

Flexible und konfigurierbare Software-Architekturen für datenintensive ontologiebasierte Informationssysteme

**Vom Promotionsausschuss der
Technischen Universität Hamburg-Harburg
zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)
genehmigte Dissertation**

von

Michael Wessel

aus Hamburg

2008

**© Logos Verlag Berlin
ISBN 978-3-8325-2162-2**

**Eine gedruckte Version dieses Buches kann online unter
<http://www.logos-verlag.de/>
oder über den Buchhandel bestellt werden.**

**Anmerkung: In der vorliegenden Version der Arbeit wurden
einige kleinere Fehler der Logos-Version korrigiert.
Diese Korrekturen sind jedoch nicht inhaltlicher Art.**

Gutachter:

Prof. Dr. Ralf Möller

Prof. Dr. Friedrich Mayer-Lindenberg

Prof. Dr. Dieter Gollmann (Drittgutachter)

Tag der mündlichen Prüfung:

14.4.2008

Kurzfassung

Ontologien werden voraussichtlich eine wichtige Rolle für die nächste Generation von Informationssystemen spielen. Heutzutage erfährt insbesondere die Vision des Semantischen Webs (Semantic Web) einige Beachtung - soll doch durch seine Realisierung nicht nur die "intelligente" Beantwortung inhaltsbasierter Suchanfragen ermöglicht werden, sondern beispielsweise auch Dienste im Web automatisch anhand funktionaler Dienstbeschreibungen ermittelt, miteinander kombiniert, und entspr. invoziert werden. Ontologien sind jedoch nicht nur im Semantischen Web relevant, sondern können in einer Vielzahl anderer Informationssystem-Domänen durch die Bereitstellung eines reichhaltigen logisch-formalen Begriffsrahmens der Domäne, einer sog. Konzeptualisierung, neuartige und innovative Zusatzdienste für Informationssysteme erbringen.

In dieser Arbeit wird ein formaler und softwaretechnischer Architekturrahmen zur Konstruktion ontologiebasierter Informationssysteme unter formalen und softwaretechnischen Gesichtspunkten entworfen und realisiert. Zur Sicherstellung einer hohen Flexibilität und Anwendbarkeit in mehreren Informationssystem-Domänen wird der Rahmen so entworfen, dass nicht nur einzelne Punkte im Informationssystem-Entwurfsraum, sondern ganze Regionen abgedeckt und adressiert werden können. Aktuelle Semantic Web-Sprachen, wie die Web Ontology Language (OWL), können hier ebenfalls Anwendung finden.

Durch durchgängige Betrachtung zweier sog. Leitdomänen für Informationssysteme werden domänenspezifische Aspekte von der Analyse über den Entwurf bis hin zur Realisierung prototypischer ontologiebasierter Informationssysteme in diesen Leitdomänen in den Rahmen einfließen. Zum einen wird die Domäne des Semantischen Webs betrachtet, und zum anderen die Domäne der geografischen bzw. räumlichen Informationssysteme. Ein wesentlicher Schwerpunkt der Arbeit liegt auf der Realisierung einer Maschine zur ontologiebasierten Anfragebeantwortung. Die Arbeit adressiert alle hierfür zu lösenden Teilproblemgruppen, insbesondere das Repräsentations-, das Anfragebeantwortungs-, und das Inferenzproblem. Die sog. Beschreibungslogiken bilden ein weiteres wichtiges Standbein, stellen sie doch die theoretische Fundierung für aktuelle Ontologie-Sprachen (wie OWL) bereit. Die Arbeit beschreibt zudem den Entwurf und die Implementierung der Beschreibungslogik-Anfragesprache nRQL, die von vielen Nutzern des bekannten Beschreibungslogiksystems RacerPro verwendet wird. nRQL ist ein wichtiger Bestandteil des in dieser Arbeit entworfenen und realisierten Gesamtrahmens.

Abstract

It is now commonly accepted that ontologies will play an important role for the next generation of information systems. Nowadays, the vision of the Semantic Web receives great attention, given that it not only promises “intelligent” answers to content-based user retrieval queries on the Web, but also automated lookup, composition and invocation of Web services, based on their functional service descriptions. Moreover, ontologies are not only relevant in the field of the Semantic Web, but also in a multitude of other information system domains. By providing a rich and logically formal frame of well-defined notions, a so-called conceptualization of a domain, new and innovative services of added value may be realized for information systems.

In this thesis, a formal software framework aimed to support the construction of ontology-based information systems is designed and implemented under formal and software technical concerns. In order to ensure a high flexibility and applicability of the framework in various information system domains, the framework was designed in such a way that it is able to cover and address regions in the design space of information systems, not merely single points in this space. Current Semantic Web languages, such as OWL, will be embraced as well.

By continues consideration of two so-called “guiding domains” for information systems, domain specific aspects and requirements influence and guide the framework under development, from the requirement analysis phase over the design phase to the implementation of the framework as well as prototypical information systems in the guiding domains, built upon and with the framework. On the one hand, the Semantic Web will be considered as one guiding domain. On the other hand, geographical / spatial information systems will be considered. A main focus of this work lies on the realization of an ontology-based query answering engine. The work covers all three main problem categories which have to be addressed for the realization of such an engine: the representation problem, the query answering problem, and the inference problem. The so-called family of description logics provide an important pillar for this work, given they provide the theoretical basis and foundation of contemporary ontology languages (e.g., OWL). This thesis also describes the design and implementation of the description logical query language nRQL, which is used by many users of the well-known description logic system RacerPro. nRQL is an integral part of the framework designed and realized in this work.

Danksagungen

Ich danke allen Personen und Institutionen, die mich auf dem langen Weg bis zur Vollendung dieser Arbeit unterstützt haben. Insbesondere danke ich meiner Familie, und natürlich meinen Lehrern: Prof. Ralf Möller, Prof. Volker Haarslev, Prof. Bernd Neumann, und auch Prof. Peter Schefe, dem ich die ersten Kontakte mit der funktionalen Programmierung im Rahmen der Informatik-Anfängervorlesung der Universität Hamburg im Wintersemester 1991/1992 zu verdanken habe, und dessen KI-Vorlesung später mein Interesse an Fragestellungen der Wissensrepräsentation geweckt hat. Vielen Dank an Prof. Mayer-Lindenberg für die Bereitschaft, die Rolle des Zweitgutachters zu übernehmen und diese doch recht umfangreiche Arbeit zu bewerten. Prof. ter Haseborg und Prof. Gollmann möchte ich für ihr Engagement in meinem Promotionsverfahren danken. Bei Prof. Leonie Dreschler-Fischer und Prof. Christian Freksa möchte ich mich für die Hilfe bei der Beantragung eines Grundstipendiums der Universität Hamburg, und bei der Universität Hamburg für das bewilligte Grundstipendium in den Jahren 1999 - 2001 bedanken. Bei den Kollegen und Studenten des Arbeitsbereichs KOGS (Fachbereich Informatik der Universität Hamburg) und des Instituts STS (Technische Universität Hamburg-Harburg) möchte ich mich für unzählige interessante Diskussionen und Anregungen bedanken, ebenso wie bei Kay Hidde von der Racer Systems GmbH & Co. KG, sowie bei Mike Fischer. Vielen Dank auch an Dr. Özgür Özcep für die vielen Korrekturhinweise. Meinem Freundeskreis danke ich für motivierende Gespräche.

Hamburg, im Juli 2007 (überarbeitet im August 2010)

Inhaltsverzeichnis

1	Einleitung und Übersicht	9
1.1	Zum Vorgehen in dieser Arbeit	21
1.2	Ergebnisse der Arbeit	22
1.3	Struktur der Arbeit und weitere Anmerkungen	23
2	Benötigte Grundlagen aus Theorie und Praxis	27
2.1	Logikbasierte Wissensrepräsentation	29
2.2	Beschreibungslogiken	62
2.3	Zusammenfassung	125
3	Ontologiebasierte Informationssysteme – Zwei Leitdomänen	127
3.1	Leitdomäne 1: Geographische Informationssysteme	133
3.2	Leitdomäne 2: Information Retrieval im SEMANTIC WEB	163
3.3	Zusammenfassung	187
4	Ein formales Rahmenwerk für ontologiebasierte Informationssysteme	191
4.1	Das Substrat-Datenmodell	193
4.2	Eine generische Anfragesprache für Substrate	202
4.3	Zusammenfassung	230
5	Softwaretechnische Realisierung des formalen Rahmenwerks	231
5.1	Realisierung des Substrat-Datenmodells	235
5.2	Realisierung der Substrat-Anfragebeantwortungsmaschine	278
5.3	Vier konkrete Substrat-Anfragesprachen	318
5.4	Zusammenfassung	398
6	Bereitstellung offener und flexibler DL-Beweiser im Rahmenwerk	399
6.1	Motivation und Anforderungen	401
6.2	Entwurf von MIDeLoRA	412
6.3	Implementation von MIDeLoRA	488
6.4	Vorstellung einiger MIDeLoRA-Instantiierungen	499
6.5	Zusammenfassung	522
7	Evaluation und Ausblick	523

Inhaltsverzeichnis

7.1	Evaluation der Anfragebeantwortungsmaschine	525
7.2	Evaluation von MiDeLoRA	542
7.3	Schlusswort und Ausblick	552
Literaturverzeichnis		555
A Anhang		575
A.1	LUBM-TBox	575
A.2	LUBM-Taxonomie	578
A.3	DISK 1	579
A.4	DISK 2	580
A.5	Verzeichnis wichtiger Akronyme, Abkürzungen und Namen	581
B Lebenslauf		585

1. Einleitung und Übersicht

Eine Dissertation mit den „Trendwörtern“ *Ontologie*, *flexibel*, *konfigurierbar*, *Softwarearchitektur* im Titel muss sich unmittelbar nach der Bedeutung dieser Begriffe fragen lassen; insbesondere, da es für diese Begriffe vielfältige und schillernde Nutzungen gibt, und es dem Autor ein Anliegen ist, dass Begriffe nicht unreflektiert verwendet werden. Wir wollen uns dem Thema und Kontext dieser Arbeit nähern, indem wir die im Titel der Arbeit verwendeten Begriffe besser ausleuchten. Eine lt. Siedersleben [Siedersleben 2004] weitgehend akzeptierte Definition des Begriffs *Softwarearchitektur* ist in [Bass u. a. 1998] gegeben:

The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components and the relationships among them.

Statt Softwarearchitektur ist lt. Siedersleben auch der Begriff *Softwarestruktur* treffend, der zudem unmittelbar verständlich ist. Die *Softwarekomponente* ist nach dieser Definition also das zentrale Software-Strukturierungsmittel. Da die Granularität der Softwarekomponenten hier nicht weiter festgelegt wird, kann man davon ausgehen, dass der Begriff der Softwarearchitektur sowohl die Struktur von Softwaresystemen im Großen (z.B. Aufteilung eines Systems in „Schichten“) wie auch im Kleinen (z.B. auf der Ebene der einzelnen Klassen) betreffen kann.

Im Folgenden verwenden wir statt *Komponente* lieber den neutraleren Begriff *Baustein*, da der Komponenten-Begriff oftmals *Black Box-Wiederverwendung* und *lose Kopplung* impliziert. In unserer Betrachtung kann ein solcher Softwarebaustein jedoch auch eine softwaretechnische Klasse bzw. eine Instanz einer solchen sein, die beispielsweise durch ein *Rahmenwerk (Framework)* nach dem durch *Vererbung* ermöglichtem „offen-geschlossen-Prinzip“ [Meyer 1990] als wiederverwendbarer und erweiterbarer Baustein bereitgestellt wird. Oftmals wird Vererbung für Komponenten (im Gegensatz zu Klassen) jedoch explizit ausgeschlossen, vgl. z.B. [Siedersleben 2004, Kap. 3.6]. Wenn wir im Folgenden Baustein sagen, so wollen wir hiermit nichts über die Art der Wiederverwendung, der Kopplung mit anderen Bausteinen, oder über die Granularität eines solchen Bausteines aussagen.

Zentrales Anliegen dieser Arbeit ist die Identifikation, der (formale und softwaretechnische) Entwurf, und die Implementation von wiederverwendbaren und flexiblen Softwarebausteinen zur *Realisierung der Basisdienste ontologiebasierter Informationssysteme*. Diese Bausteine lösen durch die Bereitstellung ihrer Dienste *unterschiedliche Probleme bzw. Problemgruppen*, die bei der Implementierung eines ontologiebasierten *Informationssystemes (ISs)* typischerweise auftreten. Es sei vorweg geschickt, dass die im Folgenden verwendeten Begriffe später in dieser Arbeit präzisiert werden (insbesondere in Kap. 2); für dieses Kapitel ist jedoch ein intuitives Verständnis erforderlich und ausreichend. Die wesentlichen Problembereiche, die beim Bau eines ontologiebasierten ISs auftreten, sind (vgl. z.B. auch [Scheffe 1991]):

1. Das *Repräsentationsproblem*: Wie und mit welchen formalen und softwaretechnischen Mitteln werden die Daten bzw. Informationen des ISs repräsentiert? Wie sehen die Datenmodelle aus?
2. Das *ontologiebasierte Anfragebeantwortungsproblem*: Wie und mit welchen formalen und softwaretechnischen Mitteln können die Daten bzw. Informationen wieder abgerufen werden?
3. Das *Inferenzproblem*: Wie und mit welchen formalen und softwaretechnischen Mitteln können die (i.d.R. schon für 2. notwendigen) *Deduktionen* durchgeführt werden?

Die Bausteine werden in einem Rahmenwerk bereitgestellt, das *flexibel, erweiterbar und konfigurierbar* ist. *Flexibilität* bedeutet in dieser Arbeit im Wesentlichen, dass das Rahmenwerk nicht nur einzelne *Punkte* im IS-Entwurfsraum, sondern *Regionen* in diesem Raum abdecken kann. Anders als oftmals im Forschungsfeld des SEMANTIC WEB [Berners-Lee u. a. 2001] ist es daher nicht unser primäres Anliegen, das Rahmenwerk auf den einen oder anderen W3C-Standard (z.B. OWL) festzulegen [Bechhofer u. a. 2006a].

In dieser Arbeit wird ein Rahmenwerk zur Erstellung (prototypischer) ontologiebasierter ISe

1. motiviert, dann
2. formal und softwaretechnisch entworfen, und schließlich
3. implementiert.

In dieser Arbeit betrachten wir durchgängig zwei *Leitdomänen für ontologiebasierte ISe*:

1. Ontologiebasierte Anfragebeantwortung in einem geographischen Informationssystem (GIS),
2. „Information Retrieval“ im SEMANTIC WEB.

Diese IS-Domänen dienen zum einen der *Gewinnung von Anforderungen* für das Rahmenwerk, aber auch der *Identifizierung und Demonstration von Potentialen* ontologiebasierter ISe. Durch Nutzung des Rahmenwerks wird in jeder Leitdomäne ein IS prototypisch entworfen und implementiert. Die erstellten prototypischen ISe werden auch als *Instantiierungen des Rahmenwerks* bezeichnet.

In dieser Arbeit geht es primär jedoch weder um das entstandene Rahmenwerk, noch um die Instantiierungen des Rahmenwerks. Dementsprechend hat der Autor auch nicht den Anspruch, dass das Rahmenwerk *für andere Entwickler* zur Wiederverwendung bereitgestellt wird. Der Autor ist jedoch der Meinung, dass die im Rahmenwerk verkörperten *pragmatischen Problemlösungen und Entwürfe* eine Art *Landkarte für ein schwieriges Terrain* für andere Entwickler liefern können. Diese Landkarte kann aufgrund der *inhärenten intellektuellen Komplexität* der beim Bau von *ontologiebasierten* ISen auftretenden und zu lösenden Probleme für andere Entwickler wertvoll sein. Aufgrund des zunehmenden Interesses an semantikbasierten Informationstechnologien ist der Autor der Meinung, dass die Relevanz solcher Arbeiten gegeben ist und sogar zunehmen wird.

Die in dieser Arbeit betrachtete Klasse von Software ist durch *datenintensive ontologiebasierte Informationssysteme* gegeben. Das zu erstellende Rahmenwerk ist für diese Klasse von Software auszulegen. Warum soll die Erstellung eines ontologiebasierten ISs nun so viel schwieriger sein (wir sprachen von inhärenter intellektueller Komplexität) als die Erstellung eines konventionellen (vielleicht webbasierten) ISs, was heutzutage oftmals als IT-Routine betrachtet wird? Zur Beantwortung dieser Frage müssen wir den Begriff *ontologiebasiert* etwas genauer ausleuchten. Ein ontologiebasiertes IS verwendet zur Erbringung seiner Dienste eine oder sogar mehrere Ontologien; eine weitgehend akzeptierte Definition des Begriffs *Ontologie* stammt von Gruber [Gruber 1993a;b]:

An ontology is a specification of a conceptualization.

Eine Ontologie ist also eine formale Beschreibung (eine Spezifikation) einer sog. Konzeptualisierung. Doch was ist eine Konzeptualisierung? Der Begriff geht auf Genesereth und Nilsson [Genesereth u. Nilsson 1987] zurück. Demzufolge handelt es sich um ein mathematisches, z.B. relationales *Modell* einer Domäne. Diese Domäne bezeichnen wir im Folgenden als die *Domäne des ISs*. Konzeptuelle Modelle werden schon seit langem auch beim Datenbankentwurf genutzt; in [Lockemann u. Schmidt 1987, Kap.5] werden diese z.B. zunächst in sog. *Miniwelten* modelliert, welche dann z.B. mittels ER-Diagrammen spezifiziert und somit sprachlich formalisiert werden. Man spricht von konzeptueller Datenmodellierung. Gemäß obiger Definition kann eine solche formale sprachliche Beschreibung auch eine Ontologie genannt werden. Die entstandenen *semantischen / konzeptuellen Datenmodelle* werden dann in *logische* und schließlich in *physische Datenmodelle* abgebildet.

Die Konzeptualisierung wird in *konventionellen ISen* durch das Schema des ISs (bzw. der Datenbank) spezifiziert. Somit wird auch festgelegt, was im IS *existieren kann* – philosophisch ist die Ontologie die *Lehre vom Seienden*. Das Modell der Domäne bzw. die Konzeptualisierung ist, wie alle Modelle, eine Abstraktion der Domäne bzw. der „repräsentierten Welt“. Statt von Konzeptualisierung kann man auch von einer *Repräsentation* sprechen. Typischerweise wird in konventionellen ISen die *Annahme der Weltabgeschlossenheit* (*Closed World Assumption, CWA*) und die *Annahme der Domänenabgeschlossenheit* (*Closed Domain Assumption, CDA*) gemacht. Die CWA besagt, dass Abwesenheit von Daten als *negative Information* interpretiert wird. Die CDA sagt aus, dass alle für das IS relevanten Objekte der Domäne explizit modelliert und entsprechende Informationen über diese im Datenbestand vorliegen.

Ontologiesprachen (die formalen Sprachen bzw. Logiken, in denen die Ontologien abgefasst werden) sind i.d.R. ausdrucksmächtiger als die für konventionelle ISe verwendeten Schemasprachen (z.B. das relationale Datenmodell). Aufgrund der erhöhten Expressivität können in Ontologien dann komplexere Domänen-Charakteristika und -Zusammenhänge modelliert und repräsentiert werden. Wir sprechen von einer *tiefen Modellierung der Domäne*. Ein ontologiebasiertes IS kann typischerweise auch *implizite und unvollständige* (z.B. *indefinite*) *Informationen* repräsentieren, d.h., die CWA und CDA müssen nicht notwendigerweise gelten. Im Gegensatz zu einem konventionellen IS wird daher nicht nur eine mögliche Ausprägung der Miniwelt als „mögliche Welt“ angenommen bzw. repräsentiert, sondern u.U. *mehrere* (insbesondere, wenn indefinite Informationen vorliegen).

Durch die tiefere Modellierung der Domäne können *Anfragen (Queries)* an das IS u.U. *vollständiger* beantwortet werden, da dem System zur Anfragebeantwortung nun mehr *Wissen* über die in der Domäne geltenden Sachzusammenhänge zur Verfügung steht. Das System ist in gewisser Weise „informierter“. Für die Anfrageformulierung wird das in der Ontologie definierte *domänenspezifische Vokabular* verwendet. Diese Art der Anfrageformulierung ist zudem als *nutzerfreundlich* bekannt. Weitere Mehrwertdienste werden durch Ontologien ermöglicht: z.B. können Zusammenhänge in den Daten entdeckt, Anfragen auf Konsistenz oder Enthaltensein geprüft werden, etc. Wir kommen auf die Vorteile der ontologiebasierten Anfragebeantwortung in Kap. 2 zurück. Ontologiebasierte Anfragebeantwortung ist eine wesentliche Kernidee des SEMANTIC WEB [Berners-Lee u. a. 2001, Baader u. a. 2003b, Antoniou u. v. Harmelen 2004].

Die von ontologiebasierten ISen ermöglichten Mehrwertdienste gibt es natürlich nicht umsonst: *Inferenz* wird erforderlich, um die u.U. nur implizit modellierten Domänen-Zusammenhänge explizit und somit für die Anwendung nutzbar zu machen. Inferenz wird oftmals als ein Prozess definiert, der in einer Daten-,

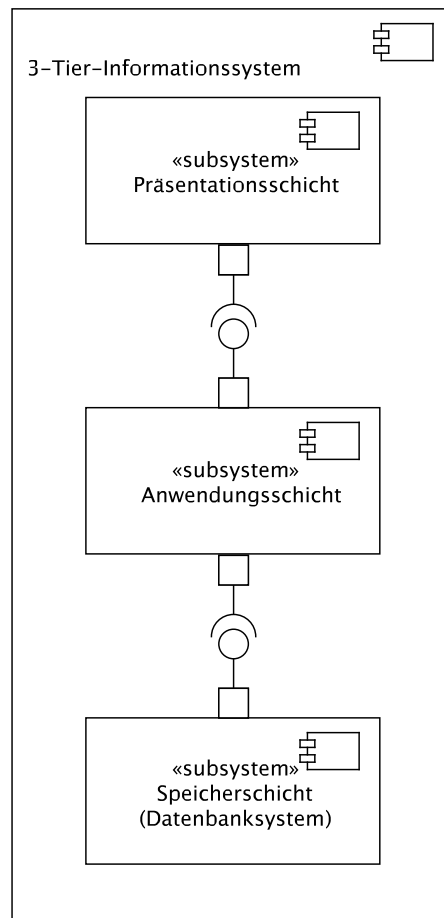


Abbildung 1.1.: Klassische 3-Tier-Architektur eines Informationssystems.

Informations- oder Wissensbasis implizit repräsentiertes Wissen mittels *logischer Deduktion* bzw. mittels eines *Inferenzverfahrens* expliziert. Ein ontologiebasiertes IS ist daher ein spezielles *wissensbasiertes System*.

Für Informationssysteme existieren heutzutage vielfältige Referenz- oder Standardarchitekturen [Alonso u. a. 2004, Singh u. Huhns 2006], z.B. die allgemeinbekannte grobgranulare 3-Schichten-Architektur, welche die klassische Aufteilung eines ISs in eine Präsentations-, eine Anwendungslogik- und eine Datenspeicher-Schicht vorsieht, s. Abb. 1.1, oder die sehr viel feingranularen durch J2EE-Rahmenwerke bzw. -Komponentenmodelle implizierten Architekturen. Wir stellen fest, dass derartige „Standardarchitekturen“ für ontologiebasierte ISe heutzutage noch nicht existieren (doch existieren speziell für das SEMANTIC WEB entspr. Toolkits, die benötigte Grundfunktionalität bereitstellen, u.a. JENA [McBride 2002], PROTÉGÉ [Musen 1998, Gennari u. a. 2003] und KAON2 [KAON 2007]). Die heute beim Bau von ontologiebasierten ISen zu lösenden Probleme erschöpfen sich daher nicht in der Auswahl, der Kon-

figuration und dem „Deployment“ von geeigneten Standardkomponenten und -Rahmenwerken. Zum einen existieren Rahmenwerke mit den für diese Arbeit geforderten Eigenschaften noch nicht. So legen z.B. Rahmenwerke wie JENA die Ontologiesprachen auf W3C-Standards (wie RDFS und OWL) fest, sodass die Flexibilität und damit ein wesentliches Anliegen dieser Arbeit, nämlich *Regionen im Entwurfsraum* abdecken zu können, verletzt wäre. Zum anderen liegen heutzutage noch zu viele Probleme *im Detail* (sodass Rahmenwerke – sofern sie existieren – entweder schlechte Performanz oder gar keine Standardimplementationen für bestimmte Problembereiche anbieten könnten). Der Bau eines ontologiebasierten ISs mit „Komponenten von der Stange“ ist daher heutzutage noch nicht möglich. Insofern ist die vorliegende Situation nicht vergleichbar mit der Situation für konventionelle ISe. Dieser Umstand macht eine Arbeit über das Thema wissenschaftlich interessant.

So kann es in einer Dissertation über Softwarearchitekturen für ontologiebasierte ISe auch nicht ausreichen, grobgranulare „generische“ Architekturen vorzuschlagen. Aufgrund der inhärenten intellektuellen Komplexität liegen die beim Bau ontologiebasierter ISe zu lösenden Probleme heutzutage noch überwiegend *in den Bausteinen*, und nicht so sehr in der Kombination dieser. In dieser Arbeit werden daher nicht nur entsprechende Architekturbausteine identifiziert, sondern auch Realisierungsprobleme in den Komponenten geschildert. Dementsprechend ist der Detaillierungsgrad der folgenden Entwürfe teilweise recht hoch. Dies erklärt auch, warum in dieser Arbeit überwiegend feingranulare UML-Klassendiagramme statt grobgranularer Komponenten- oder Architekturdiagramme verwendet werden.

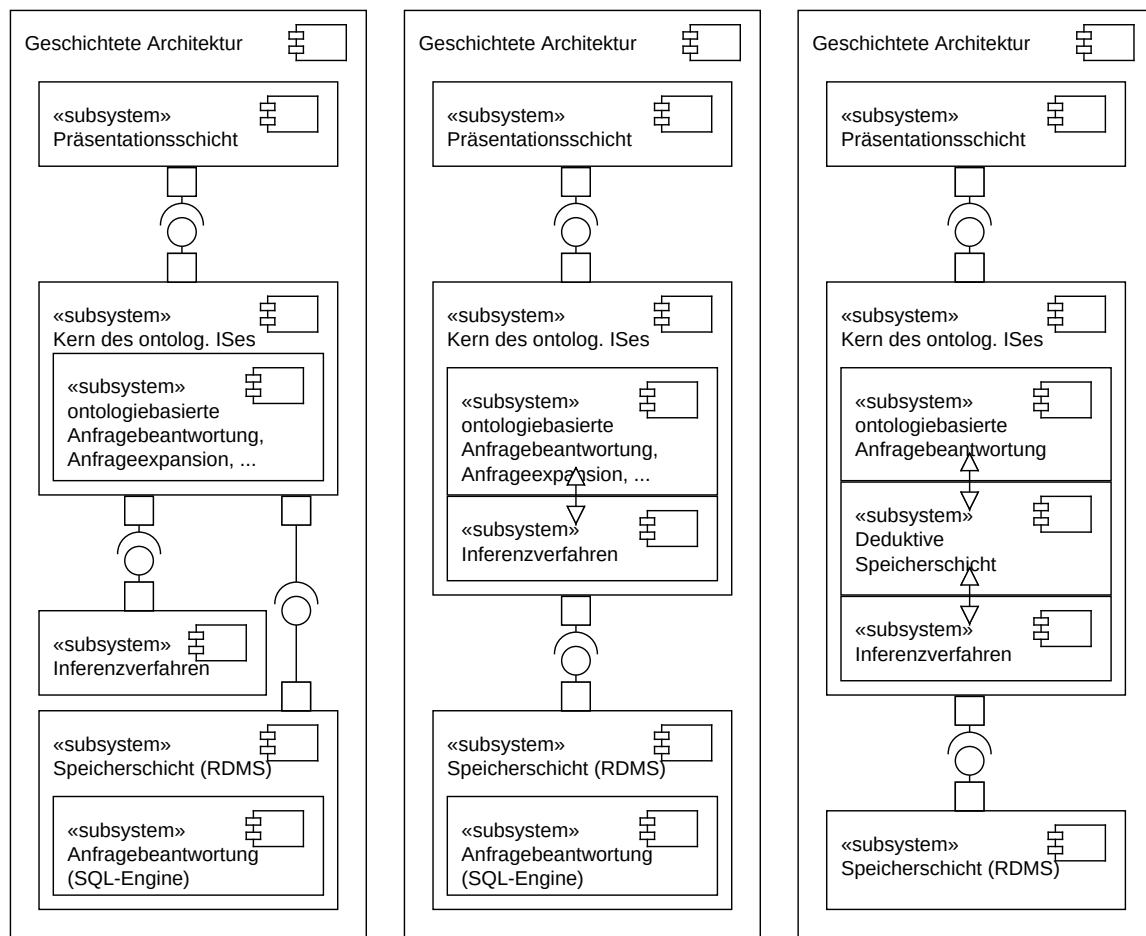
Zur Illustration eines typischen, bei der Entwicklung eines ontologiebasierten ISs auftretenden Problems betrachten wir beispielhaft das Problem, ontologiebasierte Anfragebeantwortung auf sehr großen Informations- bzw. Datenbeständen durchzuführen. Wir nennen dieses Problem im Folgenden das *Sekundärspeicher-Problem*. Eine naheliegende Idee ist, einfach eine relationale Datenbank (oder eine RDF-Datenbank) als Speicher- bzw. Repräsentationsschicht zu verwenden und über eine Standardschnittstelle (ODBC o.ä.) anzukoppeln. Dieser Speicher ist dann persistent und nicht durch die Größe des Hauptspeichers limitiert. Tatsächlich existieren SEMANTIC WEB-Rahmenwerke bzw. -Toolkits wie JENA [McBride 2002], die auch eine Datenbankbindung bieten. Da jedoch bei der Verwendung expressiver Ontologiesprachen Inferenzverfahren zur ontologiebasierten Anfragebeantwortung angewendet werden müssen, können die Anfrageergebnisse nicht alleinig von der Anfragebeantwortungsmaschine des Datenbanksystems geliefert werden. Stattdessen kann die Datenbank i.Allg. lediglich *Ergebniskandidaten* liefern. Diese müssen (aus der Perspektive der Datenbank) in die Anwendungslogik-Schicht geladen werden, sodass sie als Eingabe für Inferenzprozesse zur Verfügung stehen. Diese Inferenzprozesse müssen nun im Hauptspeicher stattfinden. Die entstehen-

den geschichteten Architekturen, s. Abb. 1.2a) und Abb. 1.2b), zeigen daher eine Reihe von Problemen auf: Zum einen ist ein hoher Kommunikationsaufwand zwischen den beiden Schichten erforderlich. Um Vollständigkeit zu garantieren, muss die zu ladende Kandidatenmenge zudem immer eine *Obermenge* der tatsächlichen Anfrageergebnisse sein. Zum anderen sind die Indexstrukturen des Datenbanksystems u.U. nicht sehr hilfreich; wir sagen im Folgenden auch *nicht ausreichend informiert*. Schlechtestenfalls muss der gesamte Datenbankbestand in den Hauptspeicher geladen werden, um die Anfrage beantworten zu können.

Zur Lösung des Problems bieten sich zwei Möglichkeiten:

1. Die Expressivität der verwendeten Ontologie- bzw. Datenbeschreibungssprachen wird soweit reduziert, dass die zur Anfragebeantwortung erforderliche Inferenz durch die Anfragebeantwortungsmaschine des Datenbanksystems allein durchgeführt werden kann (evtl. mit einigen zusätzlichen „Stored Procedures“). Wir sprechen vom *geschichteten Ansatz*. Es existieren bereits entspr. Systeme, z.B. das QUONTO-System [Acciarri u. a. 2005]. Hier wird die Technik der *Anfrage-Umformulierung (Query Rewriting)* bzw. *Anfrage-Expansion (Query Expansion)* verwendet. Das in einer Anfrage verwendete definierte Vokabular aus der Ontologie wird hier soweit expandiert, bis ausschließlich Basisvokabular in der Anfrage vorkommt. Das Basisvokabular entspricht letztlich bestimmten Relationsnamen im Datenbankschema. Es wird also ein Dienst benötigt, der Anfragen anhand der Definitionen in der Ontologie in SQL-Anfragen umschreibt. Dieser Prozess erfordert Inferenz. Das benötigte Inferenzsystem kann als externe (vgl. Abb. 1.2a)) oder interne integrierte Komponente vorgesehen werden (vgl. Abb. 1.2b)). Die Anfragebeantwortung selbst erfordert dann keine Inferenz mehr und wird vom Datenbanksystem durchgeführt.

Statt Anfragen zu expandieren und somit Inferenzprozesse in den Anfragen „zu emulieren“, können konträr hierzu auch alle folgerbaren Informationen bzw. Fakten im Voraus durch ein Inferenzverfahren außerhalb des Datenbanksystems gezogen und in der Datenbank materialisiert bzw. expliziert werden. Die Datenbank wird somit um Inferenzergebnisse angereichert. Die zur Auswertung einer ontologiebasierten Anfrage durchzuführenden Inferenzen wurden dann bereits geführt und sind in der Datenbank explizit. Anfragen können dann ohne ontologiebasiertes Umschreiben bzw. Expandieren direkt vom Datenbanksystem beantwortet werden. Diese Technik verwendet z.B. der *Instance Store* [Horrocks u. a. 2004b]. Auch hier gibt es einige Einschränkungen – so können hier lediglich unäre Relationen verwendet werden. Der Grund ist, dass zur Behandlung von mindestens zweistelligen Relationen ein Inferenzverfahren benötigt würde, welches (bei großen Datenbeständen) auf dem Se-



a) Geschichteter Ansatz b) Halbintegrierter Ansatz c) Integrierter Ansatz

Abbildung 1.2.: Architekturen für ontologiebasierte ISE im Vergleich.

kundärspeicher arbeiten müsste. Dies führt jedoch direkt zur nachfolgenden, zweiten Möglichkeit, die mit hohen Implementierungskosten verbunden ist:

2. Das Inferenzverfahren wird direkt in die Speicherschicht bzw. *in das Datenbanksystem integriert*. Die Datenbank wird dann zur *deduktiven Datenbank*. Auch hierbei handelt es sich um eine alte Idee, die primär durch die Datalog- bzw. Hornklausel-Forschungsgemeinde seit vielen Jahren verfolgt wird. Diese Systeme sind jedoch meistens Hauptspeicherbasiert.

In [Ming u. a. 2005] wird ein wesentlicher Kernalgorithmus eines zur ontologiebasierten Anfragebeantwortung benötigten Inferenzverfahrens durch SQL-Anfragen nachgebildet. Hiermit werden einige Beschränkungen des Instance Stores aufgehoben. Es ist aus der Perspektive des Autors jedoch fraglich, ob kompliziertere vollständige Inferenzverfahren auf diese Weise

(etwa mittels „Stored Procedures“ oder gar direkt auf Quellcode-Ebene) in existierende relationale (Open Source) Datenbanksysteme integriert werden können.

Statt nun das Inferenzverfahren in ein existierendes Datenbanksystem zu integrieren, gehen wir in dieser Arbeit den umgekehrten Weg: Wir *integrieren Datenbanksystem-Funktionalität in das Inferenzsystem*. Wir sprechen dann vom *integrierten Ansatz*, s. Abb. 1.2c).¹ Dieser Ansatz vermeidet zwar die mit dem geschichteten Ansatz verbundenen Nachteile (performante Anfragebeantwortung kann nur für relativ inexpressive Ontologiesprachen erreicht werden und die Flexibilität ist daher eingeschränkt), ist jedoch mit einem sehr hohen Realisierungsaufwand verbunden.

Der integrierte Ansatz bietet nach Meinung des Autors eine *höhere Flexibilität*, denn wir sind nun frei in der Wahl der Ontologiesprache, da auch expressive Ontologiesprachen unterstützt werden können; zudem können komplizierte logische Abbildungen und hieraus u.U. resultierende Modellbrüche und sog. *Impedance Mismatches* vermieden werden. Die Anfragebeantwortung kann effizient geschehen, wenn die Speicherschicht und Inferenz- bzw. Anfragebeantwortungsmaschine so aufeinander abgestimmt werden, dass gemeinsame Indexstrukturen genutzt werden. Eine *hohe Kohäsion* zwischen Inferenz und Speicherschicht erscheint hier also – ganz im Gegensatz zu einer oftmals angestrebten losen Kopplung – erforderlich. Die hohe Kohäsion ist in Abb. 1.2c) durch „zusammengeklebte“ Subsysteme visualisiert. Letztlich wollen wir diese hohe Kohäsion durch ein *gemeinsam verwendetes, flexibles Datenmodell* erreichen, sodass Modellbrüche vermieden und stattdessen gemeinsame Index-, Speicher- und Softwarestrukturen genutzt werden können. Natürlich können wir dennoch auf die Dienste existierender Datenbanksysteme zurückgreifen, wie in Abb. 1.2c) dargestellt. Ein solches Datenbanksystem ist jedoch immer in gewisser Weise „außerhalb“ des Kernsystems und kann somit höchstens Teilaufgaben lösen.

Diese Diskussion macht deutlich, dass beim Entwurf eines ontologiebasierten ISs viele Fragestellungen zu berücksichtigen sind, und dass die Grenzen zwischen dem geschichteten Ansatz und dem integrierten Ansatz fließend sind. Wir wollen eine möglichst starke Kohäsion zwischen Inferenzverfahren und Speicherschicht erreichen, sodass ineffiziente Kommunikation und vor allem Modellbrüche durch Verwendung verschiedener Datenmodelle vermieden werden. Dabei ist nach Meinung des Autors nicht die Art der Inter-Modul- bzw. Inter-Subsystem-Kommunikation das wesentliche Kriterium für das Vorliegen einer losen oder hohen Kohäsion (also z.B. lokaler vs. entfernter Prozeduraufruf), sondern ob die gekoppelten Subsysteme die gleichen Repräsentationen

¹Es besteht eine gewisse Ähnlichkeit zu einer 2-Tier-Architektur.

bzw. Datenmodelle verwenden und somit Modellbrüche vermeiden oder nicht. Derartige Modellbrüche sind jedoch nicht zu vermeiden, wenn primär ein relationales Datenbanksystem verwendet wird. Wir benötigen daher einen eigenen Architekturrahmen, der „aus einem Guss“ und für die vorliegenden Problemstellungen „gemacht“ ist.

Fassen wir die bisherige Diskussion zusammen: In dieser Arbeit wird ein durch Anforderungsanalysen für *zwei ausgewählte Leitdomänen für ontologiebasierte ISe* (s. Kap. 3) motiviertes, formal fundiertes softwaretechnisches Rahmenwerk entworfen und implementiert, das alle drei Problembereiche (Repräsentation, Anfragebeantwortung, Inferenz) abdeckt und die notwendige Flexibilität zeigt. Da die betrachteten IS-Domänen zudem recht *datenintensiv* sind, müssen wir *Skalierbarkeits-Aspekte* berücksichtigen. Auch aus diesem Grund haben wir soeben das Sekundärspeicher-Problem diskutiert. Das Sekundärspeicher-Problem ist jedoch nur eines der beim Bau eines ontologiebasierten ISs zu lösenden Probleme.

Es ist klar, dass die *softwaretechnische Realisierung des integrierten Ansatzes* mit hohem Aufwand verbunden sein wird,² denn es muss schlichtweg mehr Software entworfen und realisiert werden, als für den geschichteten Ansatz, der zumindest für die Speicherschicht und große Teile der Anfragebeantwortungsmaschine Standardkomponenten bzw. ein relationales Datenbanksystem verwenden kann. Also müssen wir in dieser Arbeit sowohl eine Repräsentations- bzw. Speicherschicht, eine Anfragebeantwortungsmaschine, als auch ein Rahmen für Inferenzverfahren bereitstellen. In dieser Arbeit können daher nur Teillösungen bzw. -erfolge erzielt werden, und die Instantiierungen des Rahmenwerks werden prototypisch bleiben.

Das Rahmenwerk ist so zu entwerfen, dass es flexible und konfigurierbare Architekturen für ontologiebasierte ISe (u.a. in den Leitdomänen) ermöglicht. Wir meinen mit einer flexiblen Architektur eine Softwarestruktur, die

1. in gewissen Grenzen an neue domänenspezifische Anforderungen angepasst werden kann („der Rahmen bricht nicht“), und
2. es ermöglicht, *Regionen im IS-Entwurfsraum* abzudecken (nicht nur Punkte).

Zur Erreichung dieses Ziels verwenden wir den klassischen Rahmenwerk-Ansatz. Tatsächlich handelt es sich um drei Rahmenwerke, die aufeinander abgestimmt werden, um die erforderliche hohe Kohäsion zu erreichen:

1. Ein Repräsentationsrahmenwerk zum Bau von Repräsentationsschichten.
2. Ein Anfragebeantwortungsrahmenwerk zum Bau von zur verwendeten Repräsentationsschicht passenden ontologiebasierten Anfragesprachen.

²Wie vielleicht schon der Umfang dieser Arbeit deutlich macht.

3. Ein Inferenzrahmenwerk zum Bau von *Beschreibungslogik-Beweisern*.

Ein prototypisches ontologiebasiertes IS kann durch *Instantiierung* oder durch *Spezialisierung und Instantiierung* von in diesen Rahmenwerken bereitgestellter Basisfunktionalität erstellt werden. Ein so gewonnenes prototypisches IS wird im Folgenden als *Instantiierung des Rahmenwerks* bezeichnet. Die Rahmenwerke stellen also nicht nur einen „leeren“ Rahmen (in Form von Schnittstellen bzw. abstrakten Klassen) zur Verfügung,³ sondern enthalten bereits umfangreiche, wiederverwendbare und konfigurierbare Basisfunktionalität in Form von Standardimplementationen.

Doch warum sollten nun insbesondere für datenintensive ontologiebasierter ISe flexible und konfigurierbare Softwarearchitekturen angestrebt werden? Hierfür gibt es nach Meinung des Autors mindestens drei gute Gründe:

1. Je nach IS-Domäne sind unterschiedliche extensionale (die Daten bzw. Informationen betreffende) Repräsentationsprobleme zu lösen, und somit unterschiedliche Repräsentationssprachen adäquat. Die Wahl eines „starren“ Repräsentationsmediums (z.B. die ausschließliche Festlegung auf sog. *Beschreibungslogik-ABoxen*, s. Kap. 2) für ein Rahmenwerk verhindert schlechtestenfalls die Nutzung des Rahmens für eine IS-Domäne, oder erfordert komplizierte Kodierungsabbildungen. Diese wiederum verschlechtern i.d.R. sowohl die Performanz als auch der Verständlichkeit sowohl des Datenbestands als auch der Implementation des ISs.

Für die Repräsentationsschicht sehen wir daher ein eigenes Datenmodell vor – das sog. *Substrat-Datenmodell*. Das Datenmodell ist sowohl durch die beiden in dieser Arbeit betrachteten Leitdomänen für ontologiebasierte ISe als auch durch die diskutierte Forderung nach einem *flexiblen* Repräsentationsmedien motiviert. Der Begriff „Substrat“ kommt aus der Biologie und bedeutet so viel wie „Nährboden“ und soll andeuten, dass durch ein Substrat eine Grundlage für die Daten- bzw. Informationsrepräsentation gegeben ist. Das Substrat-Datenmodell ist generisch genug, um z.B. *Beschreibungslogik-ABoxen* darzustellen. Mit dem Datenmodell lassen sich jedoch auch Repräsentationsprobleme angehen, für die *ABoxen* unzureichend sind (z.B. räumliche Daten in einem deduktiven GIS).

Durch Instantiierung des Datenmodells können die extensionalen Repräsentationsforderungen eines ontologiebasierten ISs befriedigt werden. Da natürlich auch eine Anfragesprache benötigt wird, sehen wir eine generische *Substrat-Anfragesprache* vor, die ebenfalls für „Spezial-Substrate“ entspr. spezialisiert und angepasst werden kann.

2. Nicht nur die verwendeten Substrate sollten IS-domänenspezifisch „pas-

³Ein Beispiel für ein solches Rahmenwerk bzw. Komponentenmodell ist das J2EE.

send“ gewählt, spezialisiert oder konfiguriert werden können, sondern auch die verwendeten *Ontologiesprachen*. Im Folgenden vertreten wir die Sichtweise, dass die Ontologiesprache *ausschließlich die intensionalen Aspekte* betrifft (für die extensionalen Aspekte sind die Substrate zuständig). Natürlich muss die Ontologiesprache „passend“ zum extensionalen Modell, also dem verwendeten Substrate-Modell, gewählt werden. Wir verwenden in dieser Arbeit insbesondere *Beschreibungslogiken (DLs)*; natürlich werden dann *Inferenzverfahren* im Rahmenwerk nicht zuletzt deshalb erforderlich, weil ontologiebasierte Anfragebeantwortung eben Deduktion erfordert. Die Beschreibungslogiken (s. Kap. 2) bilden eine ganze Familie von theoretisch wohlverstandenen Ontologiesprachen und decken somit Regionen im IS-Entwurfsraum ab.

Es ist seit langem bekannt, dass die für Beschreibungslogiken primär verwendeten Inferenzverfahren, die sog. Tableau-Verfahren, eine *hohe Modularität* aufweisen. Bestimmte „Bausteine“ in einem Tableau-Verfahren für die Sprache (bzw. Beschreibungslogik) *A* können oftmals auch in einem Tableau-Verfahren für die Sprache (bzw. Beschreibungslogik) *B* „wiederverwendet“ werden. Dies ist letztlich nicht überraschend, wenn *A* und *B* der gleichen „Familie“ von Sprachen angehören. Inferenzverfahren können daher förmlich durch Komposition und Konfiguration von Basisbausteinen *zusammengesetzt und parametrisiert* werden. Diese Wiederverwendbarkeit gilt oftmals auch für die in diesen Tableau-Verfahren angewendeten *Optimierungsverfahren*.

Wir stellen im Rahmenwerk Inferenzbausteine bereit, im Folgenden einfach *Beweiser* genannt. Ein solcher Beweiser hat eine klar definierte Aufgabe, ist für eine bestimmte Ontologiesprache (z.B. Beschreibungslogik) definiert, und arbeitet auf einer gewissen Repräsentation (einem Substrat). Beweiser müssen nicht notwendigerweise *Entscheidungsprobleme* lösen, gleichwohl dies für die meisten Beweiser der Fall sein wird. Beweiser für ausdrucksstarke DLs sind sehr komplizierte Software-Artefakte; dies ist nicht so sehr aufgrund des Umfang des Quellcodes, sondern eher aufgrund der inhärenten intellektuellen Komplexität der Algorithmen der Fall. Insbesondere integrierte Optimierungsverfahren (deren Anwendbarkeit automatisch entdeckt werden muss) verkomplizieren die Verfahren erheblich. Es stellt sich daher die Frage, wie diese Komplexitätsprobleme in einem Rahmenwerk (welches ja sogar Regionen, und nicht nur einzelne Punkte im Entwurfsraum abdecken soll) adressiert werden können. Wir werden daher entsprechende Softwareabstraktionen bzw. *domänenspezifische Sprachen zum Beweiserbau* vorsehen, sodass die Komplexität deutlich reduziert werden kann.

3. Natürlich kann durch entspr. Programmierung nahezu jedes System ir-

gendwie „angepasst“ oder „erweitert“ werden. Bei schlecht strukturierten Systemen bzw. schlechten Architekturen scheitert dieses Vorhaben jedoch oftmals an nicht überschaubarer Komplexität; die Symptome schlecht strukturierter Software sind wohlbekannt [Vogel u. a. 2005, S. 7].

Im Rahmen der *Softwarewartung* muss zudem jedes System in gewissen Zeitabständen in Grenzen angepasst und erweitert werden, um der *Softwareentropie* entgegen zu wirken. *Wartbarkeit* von Softwaresystemen wird langfristig nur durch eine „gute“ Softwarestruktur sichergestellt. Die angestrebten inneren Qualitätsmerkmale für das Rahmenwerk sind daher beileibe kein (akademischer) Selbstzweck.

Wir haben nun alle im Titel der Arbeit verwendeten Begriffe informell erläutert und motiviert, warum wir ihnen Beachtung schenken wollen. Formalere Definitionen und genauere Ausführungen folgen im weiteren Verlauf der Arbeit.

1.1. Zum Vorgehen in dieser Arbeit

Wir folgen in dieser Arbeit der *klassischen Softwareentwicklungs-Methodologie*: Anforderungsanalyse, Entwurf, Implementierung. Zur Anforderungsermittlung betrachten wir *zwei Leitdomänen für ontologiebasierte ISe*.

Anhand dieser Leitdomänen identifizieren wir zunächst Nutzungsmöglichkeiten und Probleme bei der Nutzung von Standard-Beschreibungslogiken und Standard-Beschreibungslogiksystemen zur Realisierung ontologiebasierter ISe in den Leitdomänen. Anhand der Analyse gewinnen wir Anforderungen an das Substrat-Datenmodell, die benötigte Substrat-Anfragesprache, sowie Anforderungen für das Rahmenwerk zum Bau von Beweisern. Die identifizierten Probleme motivieren jedoch auch Repräsentations- und Inferenzverfahren, die nicht aus der „Beschreibungslogik-Welt“ kommen und später realisiert werden.

Jedem Entwurf geht zunächst ein Entwurf mit formalen bzw. mathematischen Mitteln voraus, dem ein softwaretechnischer Entwurf folgt. Wir können hier mit einigem Recht von *vorausgehender Spezifikation* der zu entwerfenden Software sprechen. Der formale Entwurf vereinfacht die Überführung in einen softwaretechnischen Entwurf und garantiert eine gewisse Softwarequalität. Insbesondere sind die durch die erstellte Software angestrebten Leistungen dann klar dokumentiert und in gewisser Weise nachprüfbar, da ihre Dienste aufgrund der formalen Semantik klar spezifiziert sind. Während für die verwendeten DL-Beweiser auf größtenteils existierende Tableau-Beweiser als Spezifikationen zurückgegriffen werden können, müssen wir für das Substrat-Datenmodell und die Substrat-Anfragesprache(n) eigene Spezifikationsmittel herausarbeiten.

Schließlich werden die Entwürfe implementiert und deren Performanz anhand ausgewählter Prototypen in den beiden Leitdomänen evaluiert. Die These der Flexibilität und Konfigurierbarkeit des erstellten Rahmens wird durch die unterschiedlichen Prototypen bzw. Instantiierungen des Rahmenwerks belegt. Um *datenintensive ontologiebasierte ISe* mit dem Rahmen realisieren zu können, ist eine gewisse Performanz bzw. Skalierbarkeit (bzgl. der Größe der Daten- bzw. Informationsbestände) erforderlich. Schließlich evaluieren wir die Performanz des Rahmens bzw. seine Skalierbarkeit um somit nicht nur die erzielte Flexibilität und Konfigurierbarkeit zu belegen, sondern auch zu bestätigen, dass der Rahmen (in gewissen Grenzen) zur Realisierung *datenintensiver ontologiebasierter ISe* verwendet werden kann.

1.2. Ergebnisse der Arbeit

Worin ist nach Meinung des Autors der durch diese Arbeit erreichte wissenschaftliche Fortschritt zu sehen? Das im Rahmen dieser Arbeit erstellte Rahmenwerk hat nicht den Anspruch, von anderen Entwicklern genutzt zu werden (zumal die Nutzergemeinde der hier verwendeten Programmiersprache COMMON LISP doch sehr klein ist). Der primäre wissenschaftliche Wert dieser Arbeit ist aus der Perspektive des Autors in dem Gesamtentwurf an sich (und nicht so sehr in seiner Realisierung), sowie in den in ihm manifestierten mathematischen und softwaretechnischen Abstraktionen und Problemlösungen für viele beim Bau ontologiebasierter ISe auftretender Teilprobleme zu sehen. Die hier identifizierten Abstraktionen und Modelle können für Entwickler solcher oder ähnlicher Systeme eine Art Landkarte für ein schwieriges Terrain darstellen.

Schon der Titels der Arbeit macht deutlich, dass hier ein sehr umfassender Forschungskontext aufgespannt und verfolgt wurde. So mag auch der Umfang der Arbeit nicht verwundern. Während der Gesamtwert dieser Arbeit im Ganzen – sozusagen aus der Vogelperspektive – soeben genannt wurde, so sind doch auch viele Teilergebnisse im Rahmen dieses umfangreichen Projektes angefallen, die ebenfalls als „Fortschritt der wissenschaftlichen Erkenntnis“ bezeichnet werden können. Die Teilergebnisse dokumentieren den „roten Faden“ durch die zur Vollendung dieser Arbeit benötigten Jahre und sind in den angefallenen Publikationen festgehalten:

- Publikationen zur Komplexität von Spezial-Beschreibungslogiken zur Modellierung räumlicher Phänomene, die durch Leitdomäne 1 motiviert wurden: [Wessel u. a. 2000, Wessel 2001a; 2002; 2000b; 2001b; 2000a; 2003a].
- Publikationen zur Nutzung und Erweiterung von Beschreibungslogiksystemen zur Realisierung ontologiebasierter ISe in den Leitdomänen: [Wessel

2003b, Wessel u. Möller 2006, Wessel u. Möller 2007b].

- Publikationen zur Formalisierung und Realisierung ontologiebasierter Anfragesprachen, insbesondere von SEMANTIC WEB- und ABox-Anfragesprachen: [Haarslev u. a. 2004a;b, Wessel u. Möller 2005].
- Publikationen zur Skalierbarkeit ontologiebasierter Anfragesprachen: [Möller u. a. 2006a;b].
- Publikationen zum Bau konzeptuell fundierter Nutzeroberflächen (GUIs) für Beschreibungslogiksysteme: [Wessel u. Möller 2007a].

Als wichtigste Publikation kann der Artikel [Wessel u. Möller 2007b] angesehen werden, der eine Art Zusammenfassung der Kapitel 3, 4, 5 und Teilen von Kap. 7 darstellt. Die Arbeiten aus Kap. 6 sind noch nicht veröffentlicht, lediglich in [Wessel u. Möller 2007b] erwähnt.

Speziell die Teilergebnisse zum Thema ABox- und SEMANTIC WEB-Anfragesprachen können aus der Perspektive des Autors als erfolgreich eingestuft werden: NRQL + RACERPRO war das erste SEMANTIC WEB-Repository, welches die Anfragen aus dem sog. *Lehigh University Benchmark (LUBM)* vollständig (wenn auch mit zunächst noch relativ geringer Größe der Datenbasis) beantworten konnte. Inzwischen wurden die zur Erzielung dieses Erfolges notwendigen Optimierungstechniken (s. Kap. 6) auch von anderen Entwicklern von Beschreibungslogik- bzw. SEMANTIC WEB-Inferenzsystemen übernommen und die Evaluierung der LUBM-Anfragen gehört zum „Standard-Benchmark-Repertoire“ derartiger Systeme (s. z.B. [Sirin u. a. 2006]). Die NRQL-Anfragesprache verfügt über innovative Anfragekonstrukte, die anderen vergleichbaren Sprachen fehlen. Die durchgeführten Arbeiten können daher mit einigem Recht als „richtungweisend“ in diesem Forschungsgebiet bezeichnet werden. NRQL ist heutzutage integraler Bestandteil von RACERPRO und wird von vielen Nutzern dieses Systems verwendet. NRQL wird zudem auch als Basis für weitergehende Forschungsarbeiten im Arbeitsbereich STS der TU-Harburg genutzt, z.B. für den OWL-QL-Server RACERMANAGER [Galinski u. a. 2005, Kaplunova u. a. 2006].

1.3. Struktur der Arbeit und weitere Anmerkungen

Aus den bisherigen Erläuterungen wurde deutlich, dass es sich bei dieser Arbeit um eine *Schnittmengen-Arbeit* in den Informatik-Disziplinen bzw. -Forschungsgebieten *Beschreibungslogiken*, *Softwaretechnik* und *Informationssysteme* handelt. Da es sich um eine praktische Arbeit handelt, deren Ziel nicht zuletzt auch die Herstellung funktionierender Software ist, liegt der Schwerpunkt nicht primär im Theoretischen. Der Autor ist jedoch bestrebt, die Soft-

1.3 Struktur der Arbeit und weitere Anmerkungen

warearchitektur auf eine saubere formale Basis zu stellen.

Die zum Verständnis der formalen Basis der Arbeit notwendigen Grundlagen werden in Kap. 2 aufbereitet; zum einen, um die Arbeit in sich abgeschlossen zu machen, und zum anderen, um die vom Autor verwendete Terminologie zu präzisieren. Leser mit entsprechenden formalen Kenntnissen können dieses Kapitel evtl. überspringen.

Mit Kap. 3 beginnt der eigentliche Inhalt dieser Arbeit: Wir stellen die beiden Leitdomänen für ontologiebasierte ISe vor und analysieren diese. Hieraus gewinnen wir Anforderungen für die weitere Arbeit. Insbesondere identifizieren wir Nutzungspotentiale und -probleme von Beschreibungslogiken und existierenden Beschreibungslogiksystemen in diesen Leitdomänen. Die identifizierten Probleme motivieren die Schaffung eigener, flexibler Abstraktionen zur Adressierung des Repräsentationsproblems, des Anfragebeantwortungsproblems, und des Inferenzproblems.

Die aus Kap. 3 gewonnenen Anforderungen motivieren das bereits erwähnte generische *Substrat-Datenmodell* und die generische *Substrat-Anfragesprache*. Das Datenmodell und seine Anfragesprache werden daher in Kap. 4 zunächst formal entworfen bzw. mit logischen Mitteln „spezifiziert“. Sowohl das Substrat-Datenmodell als auch die Substrat-Anfragesprache müssen für einen speziellen Nutzungskontext bzw. eine Anwendung in einer speziellen IS-Domäne instantiiert und evtl. spezialisiert werden. Sie ermöglichen es somit, Regionen im IS-Entwurfsraum abzudecken und zeigen daher die geforderte Flexibilität.

Kap. 5 beschreibt die *softwaretechnische Realisierung* des Substrat-Datenmodells und der Substrat-Anfragebeantwortungsmaschine. Zur Darstellung einer Realisierung beschreiben wir zunächst eine verfeinerte softwaretechnische Analyse, einen softwaretechnischen Entwurf, und dann die Implementation. Zur *Kommunikation der Entwürfe* verwenden wir primär UML-Klassendiagramme (wir hatten bereits begründet, dass die Probleme in dieser Arbeit vorwiegend auf feingranularer Ebene liegen); ab und zu verwenden wir auch die sog. Entwurfsmuster [Gamma u. a. 1996], da auch durch sie eine anerkannte Kommunikationsbasis gegeben ist. Schließlich belegen wir, dass mit dem Entwurf eine gewisse Flexibilität und Streubreite erreicht wird und somit die Anforderungen umgesetzt werden, indem wir *konkrete Instantiierungen* des Datenmodells für die beiden Leitdomänen samt passender Anfragesprachen vorstellen. Die NRQL-Anfragesprache ist eine betrachtete Instantiierung.

Schließlich widmen wir uns in Kap. 6 der Bereitstellung der benötigten Inferenzbausteine. Wir entwerfen daher das MIDELOA-Baukastensystem⁴ zur Konstruktion von DL-Beweisern. Mit MIDELOA werden einige innovative Anforderungen umgesetzt, die sich in Kap. 3 ergeben.

⁴MIDELOA steht für Michael's Description Logic Reasoner.

Nachdem nun alle drei Problembereiche abgehandelt wurden (das Repräsentations-, das Anfragebeantwortungs-, und das Inferenzproblem), stellt sich die Frage, ob der entstandene Gesamtrahmen tatsächlich für ontologiebasierte ISe in den beiden Leitdomänen genutzt werden kann. Da es sich um *datenintensive* Domänen handelt, ist eine gewisse Performanz erforderlich. Der Rahmen wird daher in Kap. 7 evaluiert. Wir belegen, dass eine gewisse Skalierbarkeit (bzgl. der Größe der Daten- bzw. Informationsbestände) erreicht wurde. Schließlich kommt ein Ausblick auf mögliche zukünftige Forschungs- und Entwicklungsarbeiten.

2. Benötigte Grundlagen aus Theorie und Praxis

Inhaltsangabe

2.1	Logikbasierte Wissensrepräsentation	29
2.1.1	Elementare mathematische Logik	29
2.1.2	Wissensrepräsentation und Logik	45
2.2	Beschreibungslogiken	62
2.2.1	Historie	62
2.2.2	Die Basis-Beschreibungslogik <i>ALC</i>	64
2.2.3	Ausdrucksstarke DLs	82
2.2.4	Komplexität	92
2.2.5	Eine Beispiel-Wissensbasis	94
2.2.6	Beschreibungslogiken und Modallogiken	97
2.2.7	Inferenzverfahren und -beweiser für Beschreibungslogiken	101
2.2.8	Beschreibungslogiksysteme	109
2.2.9	Beschreibungslogiken und Datenbanken	118
2.3	Zusammenfassung	125

In diesem Kapitel werden die zum Verständnis der Arbeit benötigten Grundlagen aus der logikbasierten Wissensrepräsentation sowie der Theorie und Praxis der Beschreibungslogiken und Datenbanken dargestellt. Der Hauptzweck dieses Kapitels besteht darin, die im weiteren Verlauf der Arbeit verwendete Terminologie zu klären und die Arbeit in sich abgeschlossen machen. Mathematische Definitionen werden – falls zweckdienlich – zur Präzisierung der Begriffe verwendet.

Natürlich kann und soll diese Darstellung entspr. Lehrbücher nicht ersetzen, dennoch ist sie erforderlich. Wir gehen daher recht pragmatisch vor und formalisieren die eingeführten Begriffe nur soweit wie erforderlich. Wir halten es auch nicht für sinnvoll, Beweise aus Lehrbüchern zu replizieren.

Der Abschnitt zur logikbasierten Wissensrepräsentation bietet zunächst eine Darstellung elementarer mathematischer Logik, da diese Grundlagen erläutert werden müssen, bevor die Anwendung von Logik zur (logikbasierten) Wissensrepräsentation dargestellt werden kann. Hier werden dann Grundbegriffe aus der Wissensrepräsentation sowie der Welt der Datenbanken und Informationssysteme definiert und beleuchtet. Die Nutzung mathematischer Logik in die-

sen Gebieten bzw. Anwendungskontexten wird erläutert. Prädikatenlogik erster Stufe dient uns als „Lingua Franca“ zur Semantikdefinition; insbesondere ist Prädikatenlogik nützlich zum Verständnis von Beschreibungslogiken.

Wie in der Einleitung dargestellt, spielen *Beschreibungslogiken* (*Description Logics*, *DLs*) eine wesentliche Rolle in dieser Arbeit. Der Darstellung dieser Logiken und ihrer Implementierungen, den sog. DL-Systemen, wird daher einiger Raum gewidmet. Wir beleuchten auch die Zusammenhänge von DLs mit relationalen Datenbanken, die das traditionelle Fundament heutiger ISe bilden. Natürlich kann man DLs ebenfalls unter dem Begriff „logikbasierte Wissensrepräsentation“ einsortieren. Der Umfang der Darstellung rechtfertigt jedoch ein eigenes Unterkapitel.

2.1. Logikbasierte Wissensrepräsentation

Wissensrepräsentation mit Mitteln der mathematischen Logik hat eine lange Tradition, die nicht nur in der noch jungen *KI (Künstlichen Intelligenz)* zum Bau sog. *wissensbasierter Systeme (WBS)* angewendet wird, sondern bereits von den Alten Griechen als Form des „mechanisierten und folgerichtigen Denkens“ betrieben wurde (s. Syllogistik des Aristoteles).

Wir repetieren zunächst die elementaren Grundlagen aus der mathematischen Logik und erläutern dann die Anwendung mathematischer Logik zur Wissensrepräsentation. Zum Thema logikbasierte Wissensrepräsentation seien auch die Artikel [Baader 1996] und [Baader 1999] empfohlen.

2.1.1. Elementare mathematische Logik

Wir stellen die Grundbegriffe aus der mathematischen Logik dar, indem wir zunächst Syntax, dann Semantik und schließlich die Kalkülisierung von Logik beleuchten. Dabei beschränken wir uns auf Aussagenlogik und Prädikatenlogik erster Ordnung.

Die *Syntax* einer Logik beschreibt den strukturellen Aufbau der Formeln, während die *Semantik* die Bedeutung dieser Formeln definiert. Dem Autor dienten folgende Lehrbücher als Referenzen: [Schöning 1992, Nerode u. Shore 1993, Tuschik u. Wolter 1993, Ebbinghaus u. a. 1996, Blackburn u. a. 2001].

2.1.1.1. Aussagen (Propositionen)

Die mathematische Logik handelt von mathematischen Aussagen, ihrem Wahrheitsgehalt und wechselseitigen Beziehungen zueinander. Beispiele für Aussagen sind „Michael ist ein Mensch“, „Der Papst heißt Angela Merkel“, „Alle Frösche sind grün“, etc. Offensichtlich erachten wir die erste Aussage als wahr, die zweite als falsch, und bzgl. der dritten Aussage sind wir uns unsicher; klassische Logiken erlauben jedoch lediglich „wahr“ und „falsch“ als Wahrheitswerte von Aussagen (sog. mehrwertige Logiken werden in dieser Arbeit nicht betrachtet).

Im Gegensatz zu diesen *natürlichsprachlich* formulierten Aussagen werden Aussagen in der mathematischen Logik als wohlgeformte Sätze einer *formalen* Sprache abgefasst. Diese Sätze heißen dann *Formeln*. Derartige Aussagen können in einer gegebenen *Situation* (sog. *Interpretation*) nun entweder *wahr* (TRUE) oder *falsch* (FALSE) sein, wobei es sich um den *Wahrheitswert der Aussage* handelt. Die *Semantik* klärt, wie der Wahrheitswert einer Aussage in einer gegebenen Situation bzw. Interpretation ermittelt wird.

2.1 Logikbasierte Wissensrepräsentation

Die Aussage „Michael ist ein Mensch“ könnten wir der *Aussagenlogik* (s.u.) durch ein einzelnes Aussagensymbol, beispielsweise *michael_ist_mensch*, oder in der *Prädikatenlogik* (s.u.) durch eine Formel *mensch(michael)* oder *ist(michael, mensch)* repräsentieren. Die dritte obige Beispiel-Aussage ließe sich ebenfalls am adäquatesten prädikatenlogisch formalisieren: $\forall x.frosch(x) \rightarrow gruen(x)$. Wir halten hier gleich eine wichtige Beobachtung fest: Formeln *repräsentieren* Propositionen, vgl. auch [Brachman u. Levesque 2004, S. 4]. Wir kommen auf diese Beziehung in Kap. 2.1.2.2.1 zurück.

Hat eine Formel bzw. Aussage stets den Wahrheitswert TRUE, unabhängig von der gewählten Situation bzw. Interpretation, so handelt es sich um eine allgemeingültige Formel bzw. Aussage, eine sog. *Tautologie*. Beispielsweise ist *sein* $\vee$ $\neg$ *sein* eine solche Tautologie („ $\neg$ “ ist das Symbol für „Nicht“). Oftmals ist man nur an Formeln interessiert, die Tautologien sind, denn bei ihnen handelt es sich in gewisser Weise um „universelle Wahrheiten“. Mittels eines *Kalküls* können Tautologien bzw. *Theoreme* automatisch (also mechanisch) abgeleitet werden. Kalküle bilden die Grundlage des „mechanisierten und folgerichtigen Denkens“ und wurden schon von Aristoteles untersucht, denn ein Kalkül zieht nur Schlüsse, die *unter allen denkbaren Umständen* gültig sind.

2.1.1.2. Syntax

Eine *Logik* ist eine formale Sprache $\mathcal{L}$ und wird dementsprechend als Teilmenge $\mathcal{L} \subseteq \mathcal{A}^*$ über einem Alphabet $\mathcal{A}$ angesehen. Das Alphabet $\mathcal{A}$ wird auch *Vokabular* der Sprache genannt. Das Vokabular $\mathcal{A}$ einer *Sprache erster Ordnung* umfasst eine Menge von Konstanten, eine Menge von (Objekt-) Variablen, eine Menge von Funktionssymbolen, eine Menge von Prädikatssymbolen, sowie die Elemente $\{ (,), ., \wedge, \vee, \neg, \exists, \forall \}$; u.U. können einiger dieser Mengen auch leer sein. Wir betrachten in dieser Arbeit ausschließlich Sprachen erster Ordnung. Das Alphabet $\mathcal{A}$ ist endlich oder zumindest abzählbar; somit ist auch $\mathcal{A}^*$ abzählbar.

Die Elemente von $\mathcal{L}$ heißen *Formeln*. Nur gewisse Elemente aus $\mathcal{A}^*$ werden als Formeln und somit Elemente in $\mathcal{L}$ angesehen, nämlich solche, die *wohlgeformt* sind. Eine Formel hat daher eine gewisse Bauart bzw. *Syntax*. Die Menge $\mathcal{L}$ wird induktiv definiert (die Sprache ist dann als die kleinste Menge von Wörtern definiert, die unter Anwendung der Syntax-Regeln abgeschlossen ist):

Definition 1 (Syntax der Prädikatenlogik erster Ordnung) Sei $\mathcal{V}$ eine Menge von Variablen, $\mathcal{F}$ eine Menge von Funktionssymbolen, $\mathcal{C}$ eine Menge von Konstanten, und $\mathcal{P}$ eine Menge von Prädikatssymbolen.

Variablen werden als $x \in \mathcal{V}$, Prädikatssymbole als $P^k \in \mathcal{P}$, Funktionssymbole als $f^k \in \mathcal{F}$, und Konstanten als $c \in \mathcal{C}$ notiert.

Die *Stelligkeit* eines Prädikats- bzw. Funktionssymbols bezeichnet die Anzahl

von Argumenten (sog. Termen, s.u.) des Symbols. Für $k \in \mathbb{N}$ notieren wir bei Bedarf die Stelligkeit mittels P^k bzw. f^k . Für Funktionssymbole gilt $k \geq 1$, für Prädikatssymbole P_i^k lassen wir $k \geq 0$ zu.¹

Das Tripel $(\mathcal{V}, \mathcal{F}, \mathcal{P})$ heißt auch *Signatur*. In einer *relationalen Signatur* gilt $\mathcal{F} = \emptyset$; in einer *funktionalen (auch: algebraischen) Signatur* gilt $\mathcal{P} = \emptyset$. Wir können bei Bedarf annehmen, dass die Konstanten als 0-stellige Funktionssymbole in $\mathcal{F}$ enthalten sind.

Terme sind nun induktiv definiert:

- Jede Variable ist ein Term.
- Jede Konstante ist ein Term.
- Ist $f^k \in \mathcal{F}$ ein Funktionssymbol (mit Stelligkeit k), und sind $t_1, \dots, t_k$ Terme, so ist auch $f(t_1, \dots, t_k)$ ein Term.

Prädikatenlogische Formeln sind ebenfalls induktiv definiert:

- Ist P^k ein Prädikatssymbol mit Stelligkeit $k \geq 1$, und sind $t_1, \dots, t_k$ Terme, so ist $P(t_1, \dots, t_k)$ eine Formel, auch *Atom* genannt.
- Ist P^k ein Prädikatssymbol mit Stelligkeit $k = 0$, so ist P eine Formel, auch *Proposition* genannt.
- Ist F eine Formel, so auch $\neg F$.
- Sind F und G Formeln, so sind auch $(F \wedge G)$, $(F \vee G)$, $(F \rightarrow G)$, $(F \leftrightarrow G)$ Formeln.²
- Ist x eine Variable und F ein Formel, so sind auch $\exists x.F$ und $\forall x.G$ Formeln. $\forall$ ist der universelle Quantor oder All-Quantor, und $\exists$ der existentielle Quantor oder Existenz-Quantor.

Die Menge der prädikatenlogischen Formeln ist eine Sprache erster Ordnung und wird mit $\mathcal{L}_{\mathcal{FOPL}}$ bezeichnet.

Gilt zusätzlich noch folgende Syntaxregel:

- Sind t_1 und t_2 Terme, so ist auch $(t_1 = t_2)$ eine Formel,

so haben wir *Prädikatenlogik erster Ordnung mit Gleichheit* vorliegen, und die Sprache wird mit $\mathcal{L}_{\mathcal{FOPL},=}$ bezeichnet.

■

Oftmals benötigte syntaktische Hilfsbegriffe sind:

¹Damit die Aussagenlogik formal sauber als Spezialfall der Prädikatenlogik definiert werden kann.

²Bei der Notation von Formeln werden wir oft auf Klammern verzichten, indem wir die *Assoziativität von $\vee$ und $\wedge$* ausnutzen. Prinzipiell dürfen aber auch jederzeit redundante, zusätzliche Klammern zur Disambiguierung gesetzt werden.

2.1 Logikbasierte Wissensrepräsentation

Definition 2 (Syntaktische Hilfsbegriffe) Ein Term, der keine Variablen enthält, heißt *Grundterm*.

Eine Formel ohne Quantoren wird *offen* genannt. Eine Formel, die keine Variablen enthält, heißt *Grundformel*. Ein Atom, das keine Variablen enthält, wird *Grundatom* genannt. Ein Grundatom, das keine Funktionssymbole enthält, wird *Faktum* genannt.

Ein Vorkommen einer Variable wie x ist entweder *frei* oder *gebunden* in einer Formel:

- Das Vorkommen der Variable x heißt *gebunden* in F , wenn sie in einer Teilformel von F der Form $\exists x.G$ oder $\forall x.G$ vorkommt.
- Andernfalls heißt das Vorkommen von x *frei*.

Eine Formel ohne freie Variablen wird *geschlossen* genannt, oder auch *Satz* oder *Aussage*.

Ein *universeller Satz* ist ein Satz, der mit einem Allquantor beginnt.

Eine Formel ist in *Pränexnormalform*, wenn sie die Bauart $Q_1, \dots, Q_n.F$ hat, wobei $Q \in \{\exists, \forall\}$ und alle Variablen in F frei sind; F wird die *Matrix* genannt.

Enthält F die freien Variablen $\{x_1, \dots, x_n\}$, so schreiben wir $F[x_1, \dots, x_n]$ und meinen damit, dass dies *alle* freien Variablen in F sind. ■

Oftmals möchten wir *Namen* für komplexe Formeln verwenden:

Definition 3 (Name für eine Formel) Sei $F[x_1, \dots, x_n]$ eine *komplexe* Formel (also nicht nur ein Prädikatsname). Wir definieren dann $P =_{def} F$ und können fortan $P[x_1, \dots, x_n]$ als Abkürzung für $F[x_1, \dots, x_n]$ verwenden. ■

Oftmals sollen Variablen in Quantoren lediglich über bestimmte Mengen laufen bzw. nur an solche Individuen gebunden werden, die a priori ein bestimmtes Prädikat erfüllen. Wir definieren daher die *relativierten Quantoren* wie folgt:

Definition 4 (Relativierte Quantoren) Sei $\mathcal{M}$ eine Menge, und seien F und G beliebige Formeln mit (freier Variablen x). Die *relativierten Quantoren* sind dann wie folgt definiert:

- $\exists x, F.G =_{def} \exists x.(F \wedge G)$,
- $\forall x, F.G =_{def} \forall x.(F \rightarrow G)$,
- $\exists x \in \mathcal{M}.G =_{def} \exists x.((x \in \mathcal{M}) \wedge G)$, und
- $\forall x \in \mathcal{M}.G =_{def} \forall x.((x \in \mathcal{M}) \rightarrow G)$. ■

Ein wohlbekanntes Fragment von $\mathcal{L}_{\mathcal{FOP}\mathcal{L}}$ ist die Aussagenlogik:

Definition 5 (Syntax der Aussagenlogik) Eine prädikatenlogische Formel $F \in \mathcal{L}_{FOL}$, die ausschließliche 0-stellige Prädikatssymbole (Propositionen) verwendet, heißt aussagenlogische Formel.³

Die Menge der aussagenlogischen Formeln wird mit $\mathcal{L}_{PROP}$ bezeichnet. ■

2.1.1.3. Semantik

Der Relationsbegriff ist zentral für die Semantik; eine n -stellige Relation ist einfach eine Menge von n -Tupeln über einer Grundmenge (auch: Domäne, Universum) Δ . Funktionen sind spezielle Relationen:

Definition 6 (Relation, Funktion, Grundbegriffe) Sei Δ eine Grundmenge.

Eine n -stellige (auch: n -äre) Relation $\mathcal{R}^n$ auf Δ^n ist eine Menge $\mathcal{R}^n \subseteq \Delta^n$. Ein Element $r \in \mathcal{R}$ mit $r = (r_1, \dots, r_n)$ wird (n -stelliges) *Tupel* genannt. Zweistellige Tupel werden *Paare*, und dreistellige *Tripel* genannt.

Die *transitive Hülle* $\mathcal{R}^+$ einer binären Relation $\mathcal{R}$ ist definiert als kleinster Fixpunkt der Gleichungen

- $\mathcal{R}_0 =_{\text{def}} \mathcal{R}$
- $\mathcal{R}_i =_{\text{def}} \mathcal{R}_{i-1} \cup \{ (i, k) \mid (i, j) \in \mathcal{R}_{i-1}, (j, k) \in \mathcal{R}_{i-1} \} \text{ für } i \in \{1, \dots, \infty\}.$

Die *reflexive transitive Hülle* $\mathcal{R}^*$ einer binären Relation $\mathcal{R}$ ist definiert als $\mathcal{R}^* =_{\text{def}} \mathcal{R}^+ \cup \{ (i, i) \mid (i, j) \in \mathcal{R} \text{ oder } (j, i) \in \mathcal{R} \}.$

Eine *Funktion* ist eine Relation $\mathcal{F}^n$ für die die Funktionseigenschaft gilt: Gibt es ein $m < n$ sodass für alle $r, s \in \mathcal{F}^n$ mit $r = (r_1, \dots, r_{m-1}, r_m, \dots, r_n)$ und $s = (s_1, \dots, s_{m-1}, s_m, \dots, s_n)$ aus $\forall i \in 1 \dots m-1. r_i = s_i$ stets auch $\forall i \in m \dots n. r_i = s_i$ folgt, dann ist $\mathcal{F}$ eine $(m-1)$ -stellige Funktion, die von $\Delta^{(m-1)}$ auf $\Delta^{(n-m)}$ abbildet. Die Menge, in die abgebildet wird, wird *Bildmenge (Range)* genannt, und die andere Menge *Urbildmenge (Domain)*. Die $(m-1)$ -stellige Funktion $\mathcal{F}^n$ wird auch wie folgt notiert: $\mathcal{F}^n : \Delta^{(m-1)} \mapsto \Delta^{(n-m)}$ (statt $\mathcal{F}^n$ wird auch oft F^n verwendet, wobei $F^n \in \mathcal{F}$ ein Funktionssymbol aus der Signatur ist).

Die *Menge der Bilder* ist definiert als die Menge

$$\text{range}(\mathcal{F}^n) =_{\text{def}} \{ (r_m, \dots, r_n) \mid (r_1, \dots, r_{m-1}, r_m, \dots, r_n) \in \mathcal{F}^n \}.$$

Analog ist die *Menge der Urbilder* definiert als

$$\text{domain}(\mathcal{F}^n) =_{\text{def}} \{ (r_1, \dots, r_{m-1}) \mid (r_1, \dots, r_{m-1}, r_m, \dots, r_n) \in \mathcal{F}^n \}.$$

Ein einzelnes Element aus Δ kann auch als 0-stellige Funktion begriffen werden. ■

Funktionen können total oder partiell, injektiv oder surjektiv sein:

³Dies bedeutet natürlich, dass weder Variablen noch Terme in F vorkommen können.

2.1 Logikbasierte Wissensrepräsentation

Definition 7 (Totale, Injektive, Surjektive Funktion) Sei $\mathcal{F} : \mathcal{A} \mapsto \mathcal{B}$ eine Funktion mit $\mathcal{A}, \mathcal{B} \subseteq \Delta$. Dann heißt $\mathcal{F}$

- *total* gdw. $\mathcal{A} = \Delta$,
- *surjektiv* gdw. $\mathcal{B} = \Delta$, und
- *injektiv* gdw. wenn aus $r, s \in \mathcal{A}$ mit $r \neq s$ stets auch $\mathcal{F}^n(r) \neq \mathcal{F}^n(s)$ gilt. ■

In dieser Arbeit sind speziell binäre Relationen von Interesse; wichtige Eigenschaften binärer Relationen sind:

Definition 8 (Eigenschaften binärer Relationen) Sei $\mathcal{R}$ eine binäre Relation auf Δ . Dann heißt $\mathcal{R}$

- *transitiv* bzw. *transitiv abgeschlossen* gdw. $\mathcal{R} = \mathcal{R}^+$ (wobei $\cdot^+$ den transitiven Abschlussoperator notiert),
- *reflexiv* gdw. $\text{id}(\Delta) \subseteq \mathcal{R}$, wobei $\text{id}(\Delta) =_{\text{def}} \{(x, x) \mid x \in \Delta\}$ die sog. *Identitätsrelation* oder *Diagonale* ist,
- *irreflexiv* gdw. $\mathcal{R} \cap \text{id}(\Delta) = \emptyset$,
- *symmetrisch* gdw. $\mathcal{R} = \mathcal{R}^{-1}$ (wobei $\cdot^{-1}$ den symmetrischen Abschlussoperator notiert),
- *antisymmetrisch* gdw. $(\mathcal{R} \cap \mathcal{R}^{-1}) \setminus \text{id}(\Delta) = \emptyset$,
- *asymmetrisch* gdw. sie antisymmetrisch und irreflexiv ist.
- $\mathcal{R}$ wird *partielle Ordnung* genannt gdw. $\mathcal{R}$ reflexiv, antisymmetrisch und transitiv ist.
- $\mathcal{R}$ wird *Quasiordnung* genannt gdw. $\mathcal{R}$ reflexiv und transitiv ist. ■

Binäre Relationen können durch Graphen dargestellt werden. Wichtige Begriffe aus der Welt der Graphen sind:

Definition 9 (Graph, GAG, Beschrifteter Graph) Ein *Graph* ist ein Paar (V, E) , wobei V die Menge der *Knoten (Vertices)* und $E \subseteq V \times V$ die Menge der *Kanten (Edges)*, also eine binäre Relation auf V ist.

Ist E^+ irreflexiv, so handelt es sich um einen *gerichteten azyklischen Graphen (GAG, oder DAG für Directed Acyclic Graph)*.

Ein *beschrifteter Graph* ist ein Tripel (V, E, L) , wobei (V, E) ein Graph, und L eine Beschriftungsfunktion (Labeling Function) $L : V \cup E \mapsto \dots$ ist, deren Bildmenge hier nicht weiter festgelegt wird und daher beliebig sein kann. Ist L für V definiert, so handelt es sich um einen *knotenbeschrifteten Graphen*. Ist L für E definiert, so handelt es sich um einen *kantenbeschrifteten Graphen*. ■

Die Semantik einer Formel erster Ordnung wird durch eine *Interpretation* angegeben:

Definition 10 (Semantik der Prädikatenlogik erster Ordnung) Eine *Tarski-Interpretation* $\mathcal{I}$ (auch: *Struktur*) ist ein Paar $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. Dabei ist $\Delta^{\mathcal{I}}$ eine nichtleere Menge, die sog. Grundmenge (auch: Domäne oder Universum), und $\cdot^{\mathcal{I}}$ eine sog. *Interpretationsfunktion*, die

- jedem k -stelligen Prädikatssymbol P^k mit $k > 0$ eine k -stellige Relation (auch: k -stelliges Prädikat) über $\Delta^{\mathcal{I}}$ zuordnet,
- jedem 0-stelligen Prädikat eine Teilmenge von $\Delta^{\mathcal{I}}$,⁴
- jedem k -stelligen Funktionssymbol f^k eine k -stellige Funktion über $\Delta^{\mathcal{I}}$ zuordnet,
- jeder Konstanten c ein Element der Grundmenge $\Delta^{\mathcal{I}}$ zuordnet, und
- jeder Variablen x ein Element der Grundmenge $\Delta^{\mathcal{I}}$ zuordnet. ■

Sei α ein Term oder ein Prädikatssymbol. Dann schreibt man statt $\cdot^{\mathcal{I}}(\alpha)$ meist nur $\alpha^{\mathcal{I}}$.

Unter einer *relationalen Struktur* verstehen wir:

Definition 11 (Relationale Struktur) Eine *relationale Struktur* ist eine Tarski-Interpretation $\mathcal{I}$, deren $\cdot^{\mathcal{I}}$ lediglich für Konstanten und Prädikatsnamen definiert ist. ■

Relationale Strukturen sind in der Informatik allgegenwärtig. Beschriftete Graphen können durch relationale Strukturen dargestellt werden. Hier werden die Konstanten mit den Knoten, die unären Prädikate als Knoten-Beschriftungen, und die n -ären Prädikate als beschriftete n -äre Hyperkanten interpretiert (meist ist $n=2$). Auch *relationale Datenbanken* können als Strukturen dieser Art gedeutet werden. Natürlich kann man auch die Sichtweise annehmen, dass die Knoten nicht durch die Konstanten, sondern deren $\cdot^{\mathcal{I}}$ -Bilder in $\Delta^{\mathcal{I}}$ gegeben sind (und analog für die Kanten und Beschriftungen).

Tarski-Interpretationen werden nun verwendet, um die Semantik einer Formel F anzugeben. Dabei werden nicht beliebige Tarski-Interpretationen betrachtet, sondern lediglich solche, *die zu F passen*. Eine Interpretation heißt *zu F passend*, wenn sie mindestens die in F verwendeten Prädikatssymbole, Funktionssymbole und Variablen interpretiert (also bzgl. dieser total ist).

⁴Wir benötigen diese Klausel, damit die Aussagenlogik einfach als Teilmenge der Prädikatenlogik definiert werden kann. Alternativ könnten wir zwei ausgezeichnete Elemente TRUE, FALSE mit $\{\text{TRUE}, \text{FALSE}\} \subseteq \Delta^{\mathcal{I}}$ vorsehen und fordern, dass 0-stellige Prädikate auf $\{\text{TRUE}, \text{FALSE}\}$ abgebildet werden müssen.

2.1 Logikbasierte Wissensrepräsentation

Per Induktion über die Struktur einer Formel F wird definiert, wann F in einer Tarski-Interpretation $\mathcal{I}$ *erfüllt bzw. wahr* ist:

Definition 12 (Modell, Modellbeziehung) Sei F ein Formel, und $\mathcal{I}$ eine Tarski-Interpretation.

Zunächst wird die Definition von $\cdot^{\mathcal{I}}$ auf beliebige Terme erweitert. Sei t ein Term, dann ist $t^{\mathcal{I}}$ wie folgt definiert:

- Ist $t = x_i$, so ist $x_i^{\mathcal{I}}$ durch $\mathcal{I}$ bereits definiert.
- Ist $t = f(t_1, \dots, t_k)$, dann sei $t^{\mathcal{I}} =_{def} f^{\mathcal{I}}(t_1^{\mathcal{I}}, \dots, t_k^{\mathcal{I}})$.

Per Induktion über den Formelaufbau von F definieren wir nun, wann F *wahr in $\mathcal{I}$* ist, in Zeichen $\mathcal{I} \models F$:

- Ist $F = P^k(t_1, \dots, t_k)$ mit $k > 0$, so gilt $\mathcal{I} \models F$ gdw. $(t_1^{\mathcal{I}}, \dots, t_k^{\mathcal{I}}) \in P^{\mathcal{I}}$.
- Ist $F = P^k$ mit $k = 0$, so gilt $\mathcal{I} \models F$ gdw. $P^{\mathcal{I}} \neq \emptyset$.⁵
- Ist $F = (t_1 = t_2)$, so gilt $\mathcal{I} \models F$ gdw. $t_1^{\mathcal{I}} = t_2^{\mathcal{I}}$.
- Ist $F = \neg G$, so gilt $\mathcal{I} \models F$ gdw. $\mathcal{I} \not\models G$.
- Ist $F = (G \wedge H)$, so gilt $\mathcal{I} \models F$ gdw. $\mathcal{I} \models G$ und $\mathcal{I} \models H$.
- Ist $F = (G \vee H)$, so gilt $\mathcal{I} \models F$ gdw. $\mathcal{I} \models G$ oder $\mathcal{I} \models H$.
- Ist $F = (G \rightarrow H)$, so gilt $\mathcal{I} \models F$ gdw. $\mathcal{I} \not\models G$ oder $\mathcal{I} \models H$.
- Ist $F = (G \leftrightarrow H)$, so gilt $\mathcal{I} \models F$ gdw. $\mathcal{I} \models G$ und $\mathcal{I} \models H$ oder $\mathcal{I} \not\models G$ und $\mathcal{I} \not\models H$.
- Ist $F = \forall x.G$, so gilt $\mathcal{I} \models F$ gdw. für alle $i \in \Delta^{\mathcal{I}}$ gilt: $\mathcal{I}_{x:=i} \models G$. Hier bezeichne $\mathcal{I}_{x:=i}$ die Interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}_{x:=i})$, die durch *Redefinition* von $\cdot^{\mathcal{I}}$ für x wie folgt entsteht:

$$\cdot^{\mathcal{I}_{x:=i}}(y) =_{def} \begin{cases} i & : x = y \\ y^{\mathcal{I}} & : \text{sonst.} \end{cases}$$

- Ist $F = \exists x.G$, so gilt $\mathcal{I} \models F$ gdw. es ein $i \in \Delta^{\mathcal{I}}$ gibt, sodass $\mathcal{I}_{x:=i} \models G$ gilt.

Gilt $\mathcal{I} \models F$, so heißt $\mathcal{I}$ ein *Modell* von F , und wir sagen, dass F in $\mathcal{I}$ *erfüllt* oder *wahr* ist. Andernfalls sagen wir, dass F in $\mathcal{I}$ *unerfüllt* oder *falsch* ist: $\mathcal{I} \not\models F$. ■

2.1.1.4. Elementare semantische Begriffe und Eigenschaften

Nachdem wir nun definiert haben, wann eine Struktur *ein Modell* einer Formel ist, können wir jetzt den zentralen Begriff der mathematischen Logik nach-

⁵Mit der alternativen Interpretation 0-stelliger Prädikate als $\{\text{TRUE}, \text{FALSE}\}$ müssten wir $\mathcal{I} \models F$ gdw. $P^{\mathcal{I}} = \text{TRUE}$ fordern.

reichen. Eine Formel F heißt *erfüllbar*, wenn sie mindestens ein Modell hat. Andernfalls heißt sich *unerfüllbar*:

Definition 13 (Erfüllbarkeit, Unerfüllbarkeit) Eine Formel F heißt *erfüllbar*, falls sie ein Modell $\mathcal{I}$ hat: $\mathcal{I} \models F$. Hat F kein Modell, so ist F *unerfüllbar*. Eine unerfüllbare Formel wird auch *Kontradiktion*⁶ genannt. ■

Formeln, die in allen passenden Interpretationen wahr werden, werden *gültig* oder *tautologische* Formeln genannt:

Definition 14 (Gültigkeit) Eine Formel F heißt *gültig*, falls jede zu F passende Interpretation ein Modell von F ist. Wir schreiben dann $\models F$. Eine gültige Formel wird auch *Tautologie* genannt. Ist F keine Tautologie, so schreiben wir $\not\models F$. ■

Es gilt folgender wohlbekannter Zusammenhang zwischen Gültigkeit und Unerfüllbarkeit:

Faktum 1 (Zusammenhang von Gültigkeit und Unerfüllbarkeit) Eine Formel F ist gültig gdw. $\neg F$ unerfüllbar ist. ■

Dies bedeutet insbesondere, dass das Gültigkeitsproblem auf das Erfüllbarkeitsproblem *reduziert* werden kann, und andersrum. Voraussetzung ist jedoch, dass die entsprechende Logik den Negationsoperator bietet (dies ist z.B. nicht für alle Beschreibungslogiken der Fall, aber natürlich für Prädikaten- und Aussagenlogik).

Nachdem nun elementare semantische Eigenschaften von Formeln definiert wurden, widmen wir uns nun der wichtigsten Relation zwischen Formeln bzw. Formelmengen. Die intuitive Vorstellung, dass eine Aussage oder Formel spezieller oder stärker als eine andere Aussage ist oder diese *logisch impliziert*, wird durch den Begriff der *logischen Folgerbarkeit* formalisiert. Demnach ist eine Formel G eine logische Konsequenz oder Folgerung aus einer Formel F , wenn jedes Modell von F auch ein Modell von G ist. Dies bedeutet jedoch, dass F höchstens Modelle von G verwirft und dementsprechend „strenger“ oder „spezieller“ als G ist:

Definition 15 (Folgerbarkeit und Äquivalenz) Eine Formel G folgt logisch aus einer Formel F gdw. jedes Modell von F auch Modell von G ist: aus $\mathcal{I} \models F$ folgt $\mathcal{I} \models G$. Wir schreiben dann $F \models G$. G wird dann eine *logische Folgerung* oder *logische Konsequenz* aus F genannt.

Die Formeln F und G sind *äquivalent*, wenn $F \models G$ und $G \models F$ gilt. ■

⁶In der Literatur findet sich auch der Begriff *Oxymoron*.

2.1 Logikbasierte Wissensrepräsentation

Eine schwächere Definition von Gleichartigkeit bzw. Äquivalenz ist die sog. *Erfüllbarkeitsäquivalenz*; hier wird nicht gefordert, dass F und G die gleichen Modelle haben müssen:

Definition 16 (Erfüllbarkeitsäquivalenz) Zwei Formeln F und G heißen *erfüllbarkeitsäquivalent*, falls aus der Erfüllbarkeit von F die Erfüllbarkeit von G folgt, und andersrum. ■

Das *Deduktionstheorem* ist ein Werkzeug, welches es ermöglicht, das Folgerbarkeitsproblem für zwei Formeln F und G auf ein Erfüllbarkeitsproblem zu reduzieren, denn es gilt folgender elementarer Zusammenhang:

Faktum 2 (Deduktions-Theorem) Seien F, G Formeln. Dann gilt $F \models G$ gdw. $F \wedge \neg G$ unerfüllbar (bzw. $F \rightarrow G$ eine Tautologie) ist. ■

Für praktische Anwendungen der Logik ist man an Algorithmen interessiert, die es ermöglichen, semantische Probleme wie die Erfüllbarkeit einer Formel F oder die Folgerbarkeit von G aus F zu *entscheiden*. Die Probleme werden *Entscheidungsprobleme (Decision Problems)* genannt, und die entspr. Algorithmen *Entscheidungsalgorithmen (Decision Procedures)*. Ein Entscheidungsalgorithmus muss nun für jede mögliche Eingabe nach endlicher Zeit stets die Ausgabe TRUE oder FALSE liefern und das Problem so entscheiden. Im Falle einer *Erfüllbarkeitstesters oder -entscheiders (Satisfiability Checkers)* soll der Algorithmus genau dann TRUE als Ausgabe für die Eingabe F liefern, wenn F erfüllbar ist.

Das Erfüllbarkeitsproblem wird das *zentrale Entscheidungsproblem* einer Logik genannt. Ist das Erfüllbarkeitsproblem unentscheidbar, so wird *die Logik* als unentscheidbar bezeichnet. Dementsprechend ist der Erfüllbarkeitstester der zentrale Entscheidungsalgorithmus in einem logikbasierten System.

Seit Church ist bekannt, dass das Erfüllbarkeitsproblem für Prädikatenlogik erster Ordnung unentscheidbar ist:

Faktum 3 (Unentscheidbarkeit der Logik $\mathcal{L}_{FOL}$) Unerfüllbarkeit (und somit auch Gültigkeit und Folgerbarkeit) ist für Prädikatenlogik erster Ordnung lediglich semi-entscheidbar.

Dementsprechend sagen wir auch, dass *Prädikatenlogik erster Ordnung* unentscheidbar ist. ■

Ein entsprechendes Inferenzverfahren (s.u.) zur *Prüfung auf Unerfüllbarkeit* ist z.B. *das Resolutionsverfahren*. Dieses Verfahren ist *Widerlegungsvollständig*, d.h., es terminiert mit der Ausgabe TRUE wenn die Eingabeformel unerfüllbar ist. Andernfalls terminiert das Verfahren nicht notwendigerweise.

Die Menge der unerfüllbaren Formeln (und somit die Menge der prädikatenlogischen Tautologien) ist daher rekursiv aufzählbar. Zum Beweis kann ein Cantor-Diagonalisierungsverfahren in der Ebene $\mathbb{N} \times \mathcal{L}_{\mathcal{FOPL}}$ verwendet werden. Für ein Paar $(n, F) \in \mathbb{N} \times \mathcal{L}_{\mathcal{FOPL}}$ werden von einer Turingmaschine alle mit höchstens n Schritten durchführbaren Resolutionsbeweise für F geführt; wird F dabei als unerfüllbar nachgewiesen, so wird F auf das Ausgabeband der Turingmaschine geschrieben. Die Menge wird daher durch diese Maschine *aufgezählt*. Dieses Diagonalisierungsverfahren bzw. Aufzählungsverfahren ist für jede Menge anwendbar, deren *charakteristische Funktion* bzw. *Elementfunktion* semi-entscheidbar ist. Eine semi-entscheidbare Elementfunktion für eine Menge $\mathcal{L}$ liefert TRUE, falls $x \in \mathcal{L}$, andernfalls terminiert der Algorithmus nicht (der Funktionswert ist für solche x dann formal undefiniert).

Da eine Menge $\mathcal{L}$ entscheidbar (man sagt auch: *rekursiv*) ist gdw. $\mathcal{L}$ als auch ihr Komplement $\setminus \mathcal{L}$ rekursiv aufzählbar sind, kann die Menge der *erfüllbaren* Formeln in $\mathcal{L}_{\mathcal{FOPL}}$ *nicht* rekursiv aufzählbar sein. Dies widerspräche der Unentscheidbarkeit. Dementsprechend gibt es auch kein Semi-Entscheidungsverfahren für *erfüllbare* Formeln aus $\mathcal{L}_{\mathcal{FOPL}}$, im Gegensatz zu *unerfüllbaren* Formeln. Es ist wohlbekannt, dass Aussagenlogik entscheidbar ist:

Faktum 4 (Entscheidbarkeit der Logik $\mathcal{L}_{\mathcal{PROP}}$) Erfüllbarkeit ist für Aussagenlogik entscheidbar (und $\mathcal{NP}$ -vollständig). ■

Selbst wenn eine Logik unentscheidbar ist, können andere praxisrelevante Probleme entscheidbar sein; so ist z.B. das *Model Checking*-Problem selbst für $\mathcal{L}_{\mathcal{FOPL}}$ (auf endlichen Interpretationen) entscheidbar:

Definition 17 (Model Checking-Problem für $\mathcal{L}_{\mathcal{FOPL}}$) Gegeben eine Formel $F \in \mathcal{L}_{\mathcal{FOPL}}$ und eine *endliche* Interpretation $\mathcal{I}$, dann besteht das *Model Checking-Problem* in der Entscheidung, ob $\mathcal{I} \models F$ gilt. Das Problem ist PSPACE-vollständig [Vardi 1982], [Ebbinghaus u. Flum 1999, Kap. 7.5]. ■

Wir wollen die soeben definierten Begriffe auch auf *Mengen* von Formeln übertragen. Endliche Formelmengen können einfach durch Konjunktionen⁷, also als einzelne Formeln ausgedrückt werden. Andere Formelmengen (z.B. Theorien, s.u.) sind hingegen u.U. unendlich; das Kompaktheits-Theorem für $\mathcal{L}_{\mathcal{FOPL}}$ sichert zu, dass mit diesen genau so wie mit einzelnen Formeln „gerechnet“ werden kann:

Definition 18 ((Un)Erfüllbarkeit und Folgerbarkeit von Formelmengen) Eine Menge von Formeln $\Sigma \subseteq \mathcal{L}$ einer Logik bzw. Sprache heißt *erfüllbar*, wenn es eine Interpretation $\mathcal{I}$ gibt, die Modell aller Elemente von Σ ist: $\forall F \in \Sigma. \mathcal{I} \models F$.

⁷Sofern die entspr. Logik den Konjunktionsoperator bietet.

2.1 Logikbasierte Wissensrepräsentation

In diesem Fall heißt $\mathcal{I}$ Modell für Σ . Hat Σ kein Modell, so ist die Menge unerfüllbar.

Eine Formel G ist eine Folgerung aus Σ , wenn jedes Modell von Σ auch Modell von G ist. Wenn also mit $\mathcal{I} \models \Sigma$ auch stets $\mathcal{I} \models G$ gilt, so schreiben wir $\Sigma \models G$. Analog definieren wir Folgerbarkeit für eine Formelmenge Γ : Wenn mit $\mathcal{I} \models \Sigma$ stets $\mathcal{I} \models \Gamma$ gilt, so ist Γ eine Folgerung aus Σ , und wir schreiben $\Sigma \models \Gamma$. ■

Das Kompaktheits-Theorem besagt:

Faktum 5 (Kompaktheit) Eine Formelmenge Θ ist unerfüllbar gdw. bereits eine *endliche* Teilmenge $\Theta' \subseteq \Theta$ unerfüllbar ist. ■

Mit dem Kompaktheits-Theorems kann man z.B. für das Folgerbarkeitsproblem aus einer unendlichen Formelmenge Σ ein Semi-Entscheidungsverfahren angeben, denn es gilt $\Sigma \models \Gamma$ gdw. $\Sigma \cup \{\gamma\}$ unerfüllbar ist für alle $\gamma \in \Gamma$ (Γ sei endlich). Aufgrund des Kompaktheitstheorems wird diese Unerfüllbarkeit (falls $\Sigma \models \Gamma$ gilt) bereits nach endlicher Zeit entdeckt.

2.1.1.5. Prädikatenlogik als formaler Rahmen zur Mengenbeschreibung

Wir werden im weiteren Verlauf der Arbeit prädikatenlogische Formeln mit *freien Variablen* oftmals zur intensionalen Beschreibung von Mengen nutzen. Eine solche durch eine Formel definierte Menge wird auch *Komprehension* oder *Extension der Formel* genannt [Ebbinghaus 1994]. Intuitiv kann man sagen, dass durch die Extension das durch die Formel Repräsentierte gegeben ist.

Variablen werden durch die Interpretationsfunktion $\cdot^{\mathcal{I}}$ auf Elemente in $\Delta^{\mathcal{I}}$ abgebildet. Aus der Semantikdefinition Def. 12 ist ersichtlich, dass diese Abbildung durch $\cdot^{\mathcal{I}}$ lediglich für die *freien* Variablen einer Formel relevant ist. Gilt nun $\mathcal{I} \models F[x]$, so gibt es ein Element $i \in \Delta^{\mathcal{I}}$ mit $x^{\mathcal{I}} = i$, sodass unter dieser Variablenbelegung die Formel erfüllt ist. Dementsprechend sollte i in der Komprehension von $F[x]$ enthalten sein. Die Verallgemeinerung auf n -stellige Komprehensionen definieren wir daher wie folgt:

Definition 19 (Komprehension) Gegeben eine Formel $F[x_1, \dots, x_n]$ und eine (zu F passende) Interpretation, so ist die *Komprehension von F* definiert als

$$\{ (i_1, \dots, i_n) \mid i_1, \dots, i_n \in \Delta^{\mathcal{I}}, \mathcal{I}_{x_1:=i_1, \dots, x_n:=i_n} \models F \}.$$

■

Oftmals werden Elemente aus $\Delta^{\mathcal{I}}$ durch *Konstanten* explizit benannt sein. Gibt es eine Konstante $c_i \in \mathcal{C}$ sodass $c_i^{\mathcal{I}} = i$ (und wie bisher $\mathcal{I}_{x:=i} \models F$) gilt, so kann

x in F gegen c_i substituiert werden. Die Formel $F[x]_{x \leftarrow c_i}$ ist dann in dieser Interpretation erfüllt. Dieser Sachverhalt wird durch das Substitutions-Lemma ausgesagt:

Lemma 1 ((Vereinfachtes) Substitutions-Lemma) Gelte $\mathcal{I} \models F[x_1, \dots, x_n]$ mit $x_i^{\mathcal{I}} = j$ für ein $i \in 1 \dots n$, und $c_j^{\mathcal{I}} = j$. Dann gilt auch $\mathcal{I} \models F[x_1, \dots, x_n]_{x_i \leftarrow c_j}$. ■

In einer Komprehension sind wir i.d.R. nicht an den Elementen aus $\Delta^{\mathcal{I}}$ interessiert, sondern an ihren (durch Konstanten gegebenen) Namen:

$$\{ (c_1, \dots, c_n) \mid i_1, \dots, i_n \in \Delta^{\mathcal{I}}, c_1, \dots, c_n \in \mathcal{C}, c_1^{\mathcal{I}} = i_1, \dots, c_n^{\mathcal{I}} = i_n, \mathcal{I}_{x_1:=i_1, \dots, x_n:=i_n} \models F \}.$$

Aufgrund des Substitutions-Lemmas ergibt sich dann:

Definition 20 (Konstanten-Komprehension) Gegeben eine Formel $F[x_1, \dots, x_n]$ und eine (zu F passende) Interpretation, so ist die *Konstanten-Komprehension* von F definiert als

$$\{ (c_1, \dots, c_n) \mid \begin{array}{l} i_1, \dots, i_n \in \Delta^{\mathcal{I}}, c_1, \dots, c_n \in \mathcal{C}, \\ c_1^{\mathcal{I}} = i_1, \dots, c_n^{\mathcal{I}} = i_n, \\ \mathcal{I} \models F[x_1, \dots, x_n]_{x_1 \leftarrow c_1, \dots, x_n \leftarrow c_n} \end{array} \}.$$

In der Regel wird die Konstanten-Komprehension von F jedoch weniger Elemente enthalten als die Komprehension, da $\Delta^{\mathcal{I}}$ nichtbenannte („anonyme“ oder „implizite“) Elemente enthalten kann. Wir werden unten jedoch die sogenannte *Herbrand-Interpretation* vorstellen; in ihr enthält $\Delta^{\mathcal{I}}$ stets auch die Menge der Konstanten $\mathcal{C}$ (und noch zusätzliche Terme, die „anonyme“ Individuen denotieren), und für $c \in \mathcal{C}$ wird $c^{\mathcal{I}} = c$ gefordert. Wir können dann so tun, als ob eine Konstante ein Element aus $\Delta^{\mathcal{I}}$ wäre, denn die Existenz eines *Herbrand-Modells* ist für eine erfüllbare Formel immer garantiert (die Formel muss lediglich in *Skolem-Normalform* gebracht werden, s.u.).

Deuten wir $F[x_1, \dots, x_n]$ als eine *Anfrage (Query)*, so sind wir u.U. lediglich an den Tupeln in der Komprehension von F interessiert, die *in allen Modellen einer Menge von Formeln $\mathcal{F}$* gelten; solche Antworten werden auch *sichere Antworten* genannt:

$$\{ (c_1, \dots, c_n) \mid \mathcal{F} \models F[x_1, \dots, x_n]_{x_1 \leftarrow c_1, \dots, x_n \leftarrow c_n} \}.$$

Betrachten wir ausschließlich Herbrand-Modelle, so ist $\Delta^{\mathcal{I}}$ durch die sog. *Herbrand-Grundmenge* (s.u.) gegeben. Sei $\mathcal{M}_{\mathcal{H}}$ die Menge aller Herbrand-Modelle für F . Dann kann man leicht sehen, dass die Menge der sicheren Antworten identisch ist mit der Schnittmenge

$$\bigcap_{(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}) \in \mathcal{M}_{\mathcal{H}}} \{ (c_1, \dots, c_n) \mid c_1, \dots, c_n \in \Delta^{\mathcal{I}}, (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}) \models F[x_1, \dots, x_n]_{x_1 \leftarrow c_1, \dots, x_n \leftarrow c_n} \}.$$

2.1 Logikbasierte Wissensrepräsentation

2.1.1.6. Herbrand-Modelle

Die soeben bereits verwendeten Herbrand-Modelle sollen nun noch etwas formaler vorgestellt werden; es handelt sich in gewisser Weise um die *kanonischen Modelle der Prädikatenlogik*, wodurch ihnen eine ausgezeichnete Rolle zukommt.

Definition 21 (Herbrand-Grundmenge, Herbrand-Interpretation) Die *Herbrand-Grundmenge* $\Delta_H^{\mathcal{I}}(\mathcal{T})$ einer Menge von Termen $\mathcal{T}$ ist die Menge aller variablenfreier Terme, die sich aus $\mathcal{T}$ bilden lassen:

- Wenn $f^0 \in \mathcal{T}$ eine Konstante ist, so ist $f^0 \in \Delta_H^{\mathcal{I}}(\mathcal{T})$. Enthält $\mathcal{T}$ keine Konstanten, so ist wenigstens $a \in \Delta_H^{\mathcal{I}}(\mathcal{T})$.
- Wenn $f^n \in \mathcal{T}$ ein n -stelliges Funktionssymbol ist, und $t_1, \dots, t_n \in \Delta_H^{\mathcal{I}}(\mathcal{T})$, dann ist auch $f^n(t_1, \dots, t_n) \in \Delta_H^{\mathcal{I}}(\mathcal{T})$.

Eine *Herbrand-Interpretation* für eine Formel F ist ein Paar $(\Delta_H^{\mathcal{I}}(\mathcal{T}_F), \cdot^{\mathcal{I}}_H(\mathcal{T}_F))$, wobei $\mathcal{T}_F$ die Menge der Terme in der Formel F bezeichnet, und $\cdot^{\mathcal{I}}_H(\mathcal{T}_F)$ die folgende Bedingung erfüllt:

- für jedes n -stellige Funktionssymbol $f^n \in \mathcal{T}_F$ und $t_1, \dots, t_n \in \Delta_H^{\mathcal{I}}(\mathcal{T}_F)$ ist $f^n(t_1, \dots, t_n)^{\mathcal{I}} = f(t_1, \dots, t_n)$. ■

In einer Herbrand-Interpretation werden also Terme *durch sich selbst interpretiert*, und die Grundmenge ist die Menge aller variablenfreien Terme, die sich aus den variablenfreien Term-Bestandteilen von F bilden lassen.

Für jede erfüllbare Formel $F \in \mathcal{L}_{\mathcal{FOPL}}$ lässt sich eine erfüllbarkeitsäquivalente Formel in *Skolem-Normalform* angeben (die Details sind hier nicht relevant). Eine Formel in Skolem-Normalform ist wiederum genau dann erfüllbar, wenn sie ein Herbrand-Modell hat:

Definition 22 (Herbrand-Modell) Ein *Herbrand-Modell* für eine Formel F ist ein Modell $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ von F mit $\Delta^{\mathcal{I}} = \Delta_H^{\mathcal{I}}(\mathcal{T}_F)$, und $\cdot^{\mathcal{I}}$ erweitert die Definition von $\cdot^{\mathcal{I}}_H(\mathcal{T}_F)$ für die Variablen und Prädikatsnamen in F . ■

2.1.1.7. Theorien und Kalküle

Unter einer *Theorie* versteht man einfach eine unter der Folgerbarkeitsrelation „ $\models$ “ abgeschlossene Formelmenge. Die Formelmenge, von welcher der Abschluss gebildet wird, wird auch die Menge der *Axiome* genannt:

Definition 23 (Theorie) Eine Theorie $\mathcal{TH} \subseteq \mathcal{L}$ ist eine unter Folgerbarkeit abgeschlossene Menge von Formeln; d.h., gilt $\mathcal{TH} \models F$, so auch $F \in \mathcal{TH}$. Die Elemente von $\mathcal{TH}$ heißen *Sätze* oder *Theoreme*. ■

Eine Theorie umfasst stets alle Tautologien der zugrundeliegende Sprache $\mathcal{L}$, da ja $\{\} \models F$ für eine Tautologie F gilt, und $\{\} \subseteq \mathcal{TH}$.

Eine *inkonsistente* Theorie enthält alle Formeln: $\mathcal{TH} = \mathcal{L}$.

Eine *vollständige* Theorie enthält für eine Formel $F \in \mathcal{L}$ entweder F oder $\neg F$: $F \in \mathcal{TH}$ oder $\neg F \in \mathcal{TH}$.

Theorien können auf zwei Arten spezifiziert werden [Schöning 1992]:

- Bei der *modelltheoretischen Methode* wird eine Interpretation bzw. Struktur $\mathcal{I}$ vorgegeben, und die Theorie $\mathcal{TH} \subseteq \mathcal{L}$ wird dann als

$$\mathcal{TH} =_{def} \text{Th}(\mathcal{I}) = \{ F \mid F \in \mathcal{L}, \mathcal{I} \models F \}$$

definiert, also die Menge der Formeln aus $\mathcal{L}$, für die diese Interpretation ein Modell ist. Eine solche Theorie ist immer vollständig und unter Folgerbarkeit abgeschlossen. Ein Beispiel ist die Theorie der Struktur $(\mathbb{N}, \cdot^{\mathcal{I}})$ für die Sprache erster Ordnung mit den Konstanten $\{0, 1\}$, Prädikaten $\{<, =\}$, und den Funktionen $\{+, *\}$, die sog. Zahlentheorie. Diese Theorie enthält also die wahren Sätze der Zahlentheorie.

- Bei der *axiomatischen Methode* wird eine Menge von Axiomen $\mathcal{AX} \subseteq \mathcal{L}$ vorgegeben, und $\mathcal{TH}$ wird dann als unter Folgerbarkeit abgeschlossene Menge definiert:

$$\mathcal{TH} =_{def} \text{Th}(\mathcal{AX}) = \{ F \mid F \in \mathcal{L}, \mathcal{AX} \models F \}.$$

Eine solche Theorie ist nicht immer vollständig, da nicht notwendigerweise für jede Formel $F \in \mathcal{L}$ entweder $\mathcal{AX} \models F$ oder $\mathcal{AX} \models \neg F$ gilt.

Eine Theorie $\mathcal{TH}$ heißt (endlich) axiomatisierbar, wenn eine (endliche) Axiomenmenge $\mathcal{AX}$ für sie angegeben werden kann.

Für eine Axiomenmenge fordert man, dass sie entscheidbar („rekursiv“) ist, d.h., für beliebiges $F \in \mathcal{L}$ ist das Elementproblem bzw. die charakteristische Funktion $F \in \mathcal{AX}$ entscheidbar. Natürlich sind u.a. alle *endlichen* Axiomenmengen rekursiv.

Die Menge der Modelle einer axiomatisch spezifizierten Theorie wird auch *elementare Klasse* genannt. Beispielsweise kann die Menge *aller Graphen* als eine elementare Klasse einer Menge von Axiomen definiert werden.

Die semantischen Begriffe Erfüllbarkeit, Folgerbarkeit, Gültigkeit etc. lassen sich nicht ohne weitere „direkt“ in Algorithmen überführen. Während für die Aussagenlogik aufgrund der Endlichkeit der zu betrachtenden Modelle prinzipiell noch mit Verfahren wie Wahrheitstafeln „rein semantisch“ argumentiert werden kann, benötigt man für ausdrucksstärkere Logiken (deren Formeln

2.1 Logikbasierte Wissensrepräsentation

u.U. nur unendliche Modelle haben) sog. *Kalküle*. Ein Kalkül (auch: Inferenzverfahren, Deduktionsverfahren) ist ein mechanisches, rein syntaktisch arbeitendes Beweisverfahren, um Folgerbarkeit oder Unerfüllbarkeit einer Formel nachzuweisen. Eine von einem Kalkül abgeleitete Formel heißt auch *Konklusion*, *Ableitung*, *Inferenz* oder *Schluss*. Kann aus F mittels des Kalküls und seiner Deduktionsregeln G abgeleitet werden, so schreibt man $F \vdash G$; „ $\vdash$ “ notiert also die *Ableitbarkeits-* oder *Deduktionsrelation*. Wir hatten oben schon den sog. Resolutionskalkül für die Prädikatenlogik erwähnt. Hierbei handelt es sich um einen Kalkül, der Unerfüllbarkeit einer Formel nachweisen kann.

Zu den semantischen Begriffe *Erfüllbarkeit* und *Folgerbarkeit* („ $\models$ “) korrespondieren die syntaktischen Begriffe *Widerspruchsfreiheit* (*Konsistenz*) und *Ableitbarkeit* („ $\vdash$ “). Ein Kalkül sollte *korrekt* und *vollständig* sein:

Definition 24 (Korrektheit und Vollständigkeit) Ein Kalkül heißt *korrekt*, wenn er keine falschen Schlüsse macht, d.h., $F \vdash G$ impliziert $F \models G$.

Ein Kalkül heißt *vollständig*, wenn alle semantisch folgerbaren Schlüsse auch durch das Verfahren abgeleitet werden, d.h., $F \models G$ impliziert $F \vdash G$. ■

Ein Inferenzverfahren macht somit gemäß „ $\models$ “ Folgerbares in Form durch „ $\vdash$ “ abgeleiteter Theoreme syntaktisch explizit. Je nachdem, was durch F kodiert wird, kann ein Kalkül daher „implizite Information“ oder „implizites Wissen“ in Form von G explizit machen.

Einige Kalküle können *direkt* zur Aufzählung von Theoremen verwendet werden. Solche Kalküle sind z.B. *Hilbert-Kalküle*. Eine bekannte *Inferenzregel* ist z.B. der *Modus Ponens*, wobei es sich um ein Schlussmuster der Art

$$\frac{\alpha \quad \alpha \rightarrow \beta}{\beta}$$

handelt. Bei α und β handelt es sich um *Formelvariablen*, die gegen beliebige Formeln ersetzt werden können. So wird durch eine einzelne Schlussregel letztlich eine unendliche Axiomenmenge notiert. Sind nun beispielsweise die *Prämissen* $\{A, A \rightarrow B\} \subseteq \mathcal{F}$, so kann die *Konklusion* B von der Inferenzregel abgeleitet werden; hierbei wurde offensichtlich $\alpha = A$ und $\beta = B$ gesetzt.

Im Falle einer *unerfüllbaren Formelmenge* $\mathcal{F}$ wird ein vollständiger Kalkül sowohl $\mathcal{F} \vdash G$ als auch $\mathcal{F} \vdash \neg G$ ableiten (da eine inkonsistente Theorie ja alle Formeln enthält; anderseits wäre der Kalkül nicht vollständig). Eine Formelmengende, die einen derartigen *elementaren Widerspruch* enthält, heißt *widerspruchsvoll*, *inkonsistent* oder *kontradiktorisch*, und andernfalls *widerspruchsfrei* oder *konsistent*. Es ist offensichtlich, dass eine inkonsistente Formelmengende nicht erfüllbar sein kann.

Ist ein Kalkül korrekt und vollständig, so sind die Begriffe Erfüllbarkeit und Konsistenz bzw. Folgerbarkeit und Ableitbarkeit auswechselbar verwendbar (natürlich sollte dennoch stets der treffendere Begriff gewählt werden).

Einige Kalküle sind lediglich *widerlegungsvollständig* und können somit nicht zur direkten Generierung logischer Konsequenzen genutzt werden, z.B. der Resolutionskalkül. Hier muss zur Prüfung, ob $\mathcal{F} \models G$ gilt, die Menge $\mathcal{F}$ mit $\neg G$ *resolviert* werden, bis die sog. *leere Klausel* $\square$ (*ein elementarer Widerspruch*) auftritt (der Kalkül terminiert u.U. nicht, falls $\mathcal{F} \not\models G$ gilt).

Es gilt folgender wohlbekannter Zusammenhang für vollständige Theorien:

Faktum 6 Ein vollständige und rekursiv axiomatisierbare Theorie ist entscheidbar. ■

Hieraus folgt, dass $\mathcal{L}_{FOPC}$ nicht vollständig sein kann – denn andererseits müsste $\mathcal{L}_{FOPC}$ entscheidbar sein, da $\mathcal{L}_{FOPC}$ rekursiv axiomatisierbar ist (z.B. durch den Sequenzkalkül, vgl. [Ebbinghaus u. a. 1996, Kap. 4]). Entsprechende Resultate gelten auch für andere Theorien. Die Zahlentheorie (s.o.) ist z.B. vollständig, aber nicht entscheidbar. Also kann die Zahlentheorie nicht rekursiv axiomatisiert werden. Jedes Axiomensystem für die Zahlentheorie ist daher notwendigerweise unvollständig und „generiert“ daher nur eine Teilmenge der Theoreme der Zahlentheorie. Für einige „Wahrheiten“ gibt es daher keine Beweise (welches Axiomensystem auch gewählt wird). Dies ist der klassische *Gödelsche Unvollständigkeitssatz*.

2.1.2. Wissensrepräsentation und Logik

Wie kann nun mit Mitteln der formalen Logik *Wissen repräsentiert* werden? Was wollen wir überhaupt unter Wissen verstehen? Zur Klärung dieser Frage beleuchten wir zunächst den Begriff der *Repräsentation* sowie der elementaren Begriffe *Datum* (*Singular von Daten*), *Information* und *Wissen*. Diese stehen offensichtlich in engen Beziehungen zueinander und bauen aufeinander. Die Begriffe sind zudem unmittelbar relevant für diese Arbeit, da hier ontologiebasierte Informationssysteme betrachtet werden, wobei es sich auch um spezielle *wissensbasierte Systeme* handelt.

Nachdem der Begriff der *Wissensrepräsentation* beleuchtet wurde, wird demonstriert, wie Prädikatenlogik als Wissensrepräsentationssprache zur formalen Wissensrepräsentation genutzt werden kann, bevor dann im nächsten Abschnitt Beschreibungslogiken eingeführt und angewendet werden. Einige aus dem Forschungsfeld der *Wissensorganisation* stammenden Begriffe (wie der Begriff *Thesaurus*) werden aufgrund ihrer Relevanz für diese Arbeit ebenfalls beleuchtet. Schließlich beschreiben wir, welche unterschiedlichen Dimensionen

2.1 Logikbasierte Wissensrepräsentation

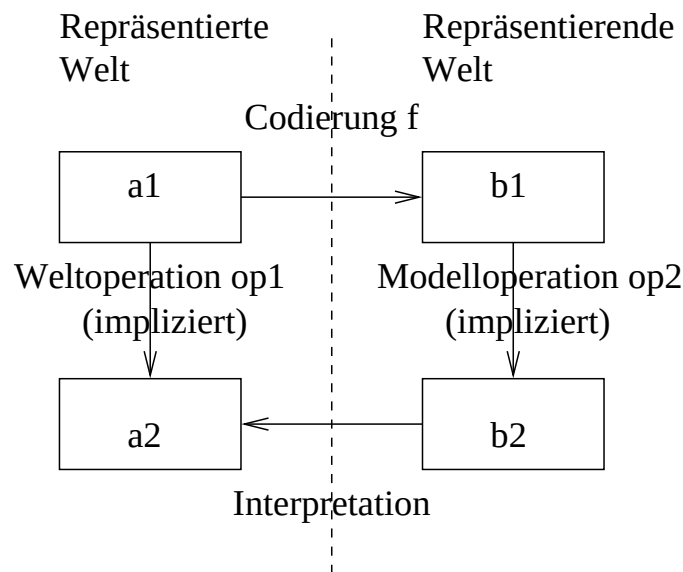


Abbildung 2.1.: Kommutatives Diagramm

zur Beurteilung bzw. zum Vergleich von Wissensrepräsentationssprachen existieren.

2.1.2.1. Modelle und Repräsentationen

Was ist eine Repräsentation? Man kann eine Repräsentation in erster Näherung mit einem *Modell* gleichsetzen, vgl. z.B. [Page 1991, Kap. 1.2], [Goos 1997, Kap. 1.2], [Rich u. Knight 1991, Kap. 4] für eine Diskussion des Modell- bzw. Repräsentations-Begriffs. Das Modell wird auch Repräsentation, und die repräsentierte Welt das Repräsentierte genannt, vgl. auch [Palmer 1978]. Durch eine Abbildungs- bzw. Kodierungsfunktion besteht ein Zusammenhang zwischen Repräsentiertem und Repräsentierendem. Die inverse Funktion wird Interpretationsfunktion genannt. Den Vorgang des Abbildens nennen wir Modellieren bzw. Repräsentieren.

Mit Hilfe eines Modells sollen Erkenntnisse bzw. Fakten und Informationen über die repräsentierte Welt gewonnen werden, die in dieser (aus welchen Gründen auch immer) so nicht zugänglich sind. Durch Anwendung von *Modelloperationen* werden im Modell Erkenntnisse gewonnen, die durch entspr. Interpretation auch gültig in der repräsentierten Welt sind, sodass entsprechende *Weltoperationen* eingespart werden können. Der Zusammenhang ist in dem klassischen und wohlbekannten „kommutativen Diagramm“ in Abb. 2.1 dargestellt. Sei f die Kodierungsfunktion, $op1$ eine Welt-, und $op2$ eine entspr. Modelloperation. Gilt $op1(a1) = a2$, so eben auch $f^{-1}(op2(f(a1))) = a2$. Da $f(a1) = b1$

kann $op2(b1) = b2$ im Modell gewonnen werden und somit auch $f^{-1}(b2) = a2$. Die Anwendung von $op1$ ist daher unnötig; allein mit Hilfe des Modells kann die „Erkenntnis“ $a2$ aus $a1$ gewonnen werden. Es ist klar, dass hierzu $op1$, $op2$ und f entspr. Eigenschaften haben müssen (natürlich gilt dieser Zusammenhang nicht für beliebige $op1$, $op2$ und f). Diese Operationen können letztlich auch *Relationen* sein (z.B. kann $op1$ eine – nicht notwendigerweise logische – Folgerungsrelation oder Kausalrelation sein).

Nach Palmer [Palmer 1978] ist eine Repräsentation als ein Tripel

(*repräsentierende Welt, repräsentierte Welt, Interpretationsvorschrift*)

zu verstehen. Im Folgenden bezeichnen wir mit Repräsentation stets die *repräsentierende Welt*, und nicht das Palmer-Tripel.

Dabei soll weder die repräsentierte noch die repräsentierende Welt näher festgelegt werden. Die repräsentierte Welt muss nicht zwangsläufig ein Ausschnitt der realen Welt sein; hier sind auch abstrakte (z.B. mathematische), gedanklich oder imaginäre Welten denkbar.⁸ So wird mit einer Wissensrepräsentation eben nicht die reale Welt, sondern *Wissen* über diese repräsentiert. Solches Wissen ist daher eher im Kopf eines Subjektes bzw. *kognitiven Agenten* zu finden (sofern es sich nicht um *Welt-* oder *Allgemeinwissen* handelt). Wir kommen hierauf in Kap. 2.1.2.2 zurück.

Ein Modell ist immer eine *Abstraktion* der repräsentierten Welt. Die Repräsentation benötigt ein *Repräsentationsmedium* und kann daher sprachlich, bildlich, gegenständlich usw. sein. Ein Globus kann z.B. als (stark abstrahierte) Repräsentation der Erde angesehen werden. Hier werden sowohl geometrische, geographische als auch thematische (z.B. politische) Sachverhalte abgebildet. Ein Modell wird immer nur so komplex sein, wie es der Zweck, für den das Modell erstellt wird, erfordert. Es werden also lediglich *problem-* bzw. *zweckrelevante* Aspekte und „Objekte“ der repräsentierten Welt im Modell Berücksichtigung finden. Die im Modell repräsentierten Aspekte stehen dann für ausgewählte Aspekte der repräsentierten Welt. Die Objekte im Modell werden *Repräsentanten*, die repräsentierten Objekte *Repräsentierte* genannt.

2.1.2.2. Daten, Informationen, Wissen

Durch *Interpretation* der repräsentierten Aspekte in einem Modell können diese zur Gewinnung von *Informationen* über die repräsentierte Welt verwendet werden, die ohne das Modell nicht ohne weiteres gewinnbar wären. Durch Information wird i.d.R. der *Wissenszustand* eines (interpretierenden) Subjektes erweitert – Information hat einen „Neuigkeitsgehalt“ und kann als die Besei-

⁸Man kann sich darüber streiten, ob z.B. mathematische Strukturen „Teil der realen Welt“ sind oder nicht.

2.1 Logikbasierte Wissensrepräsentation

tigung von Unbestimmtheit oder Unwissenheit gedeutet werden. Die Verwendung eines Globus ermöglicht z.B. die Auswertung und Erkennung geographischer Aspekte und Zusammenhänge, die einem Individuum sonst nicht ohne weiteres zugänglich wären.

Offensichtlich ist ein Globus zwar eine Repräsentation, jedoch keine rechnerverarbeitbare. Eine rechneradäquate Repräsentation nennen wir ein *Datum*. Ein Datum kodiert daher Information. Ein Datum wird oftmals als „für sich selbst stehend“ bzw. „das Gegebene“ definiert, sodass „Einigkeit bzgl. aller Beteiligten bzgl. dessen Interpretation“ besteht, vgl. [Lockemann u. Schmidt 1987, Kap. 1.1.1]. Ein Datum fixiert nach dieser Definition also bereits seine Interpretationsfunktion f . So kann man die Sichtweise einnehmen, dass beispielsweise 37 *für sich selbst* steht bzw. eine natürliche Zahl („im mathematischen Universum“) semantisch denotiert, und somit $f(37) = 37$ gilt. Diese Sichtweise ist jedoch oftmals zu schwach: So wäre in einem Informationssystem z.B. nicht klar, ob 37 eine Hausnummer oder das Alter einer Person repräsentiert (dies würde man jedoch heutzutage für sog. *selbstbeschreibenden Daten* fordern). Erst durch die Auswertung von Beziehungen zu anderen Daten wäre klar, was gemeint ist. Um also eine Bedeutung und somit Semantik bzw. Interpretation auf dieser weitgehenderen Ebene zu erreichen, müssen Daten interpretiert werden. Man mag geteilter Meinung darüber sein, ob die Interpretation von 37 als Hausnummer oder Alter einer Person nicht bereits die *pragmatische Ebene des Datumsbegriffs* adressiert, vgl. [Goos 1997, Kap. 1.1.5].

Auf syntaktischer Ebene wird ein Datum als Satz einer Daten-Beschreibungssprache betrachtet. Man unterscheidet u.a. primitive und zusammengesetzte Daten (sog. Datenstrukturen); beispielsweise ist 37 ein primitives Datum (ein numerisches Literal). Daten werden im Rechner auf Bitmuster-Ebene kodiert; auch hier spricht man weiterhin von Daten. Letztlich kann man die Sichtweise einnehmen, dass sowohl 37 als auch ein entspr. Bitmuster im Rechner nur zwei verschiedene Darstellungen für das gleiche Element in der Menge der natürlichen Zahlen sind. Während 37 unmittelbar als natürliche Zahl vom Menschen erkannt wird, sorgen im Rechenwerk eines Rechners oftmals entsprechende Kenn- bzw. Typbits dafür, dass solche Bitmuster adäquat interpretiert und die Interpretationsfunktion somit automatisch selektiert werden kann („Type Tagging“). Das Rechenwerk „weiß“ dann, dass dieses Bitmuster als natürliche Zahl zu behandeln ist. In dieser Arbeit wollen wir die Bitmuster-Ebene jedoch nicht betrachten. Ein Rechner betreibt Datenverarbeitung. Aus der Perspektive des Interpretierenden handelt es sich jedoch um Informationsverarbeitung, wenn die Datenverarbeitung in gewisser Weise „wahrheitsgetreu“ ist und es ermöglicht, die durch Datenverarbeitung gewonnenen Zusammenhänge als Wahrheiten über die repräsentierte Welt zu interpretieren, wie bereits erläutert.

Ein Datum hat daher, wie jede Repräsentation, syntaktische, semantische und pragmatische Aspekte. Zur Gewinnung der durch das Datum repräsentierten Information muss das interpretierende Subjekt die verwendete Interpretationsfunktion kennen – *Wissen* ist erforderlich. Wissen kann also als die erforderliche Information definiert werden, die benötigt wird, um Daten interpretieren, miteinander zu kombinieren und auszuwerten und somit Informationen „erzeugen“ zu können, vgl. [Goos 1997, Kap. 1.1.1]. Diese Interpretationskompetenz erfordert, dass im Subjekt Wissen repräsentiert ist, eine *Wissensrepräsentation* ist erforderlich.

Eine Wissensrepräsentation kann rein prozedural vorliegen – z.B. kann man mit einigem Recht behaupten, dass ein Rechenwerk über „Wissen“ über die natürlichen Zahlen verfügt, denn schließlich demonstriert es entsprechende Kompetenz, es rechnet richtig (und schneller als es jeder Mensch könnte). Dieses Wissen über den wahrheitsgetreuen Umgang mit natürlichen Zahlen ist im Rechenwerk (rein prozedural) repräsentiert. Die Arbeitshypothese der KI, die sog. *Wissensrepräsentationshypothese (WRH)*, verlangt jedoch eine *explizite, symbolische Repräsentation*. Wir kommen hierauf zurück.

2.1.2.2.1. Datenbanksysteme, Informationssysteme und wissensbasierte Systeme Die Begriffe *Daten*, *Information* und *Wissen* tauchen unmittelbar in den für diese Arbeit relevanten Begriffen *Datenbanksystem (DBS)*, *Informationssystem (IS)*, *wissensbasiertes System (WBS)* auf. Wo wollen wir hier die Grenzen ziehen, wie unterscheiden sich diese voneinander? Zunächst einige Definitionen [Rechenberg u. Pomberger 2006]:

Datenbankverwaltungssysteme (Database Management Systems, DBMS) sind Softwareprodukte für die dauerhafte, integere und anwendungs-unabhängige Speicherung und Verwaltung sowie die flexible und bequeme Verwendung von großen, mehrfachbenutzbaren integrierten Datenbanken.

Ein *Informationssystem* umfasst i.d.R. ein *Datenbanksystem (DBS)* (ein DBMS inkl. Datenbanken), zusätzlich stellt es jedoch domänenspezifische *Informations-Wiedergewinnungs und -Auswertungsfunktionalität* bereit [Dudenverlag 1993]:

Ein Informationssysteme (*IS*) ist ein System zur Speicherung, Wiedergewinnung, Verknüpfung und Auswertung von Informationen. Ein Informationssystem besteht aus einer Datenverarbeitungsanlage, einem Datenbanksystem und den Auswertungsprogrammen.

Die Auswertungsprogramme implementieren letztlich spezielle Interpretationsfunktionen. Ein IS ist daher spezialisierter als ein DBMS und i.d.R. für eine spezielle Klasse von Anwendungen bzw. eine spezielle IS-Domäne zugeschnitten, z.B. *geographische Informationssysteme (GIS)* [Bartelme 1995]. DBMS sind

2.1 Logikbasierte Wissensrepräsentation

daher eher als Mehrzweck-Systeme bzw. Basistechnologie zu betrachten und daher weniger zweckgebunden. Ein IS hingegen adressiert die semantischen und pragmatischen Aspekte der Daten wesentlich stärker als ein DBMS.

Alle drei Klassen von Systemen werden letztlich zur Abbildung bzw. als „Modelle“ von IS-Domänen verwendet. Einem solchen Modell liegt eine sog. *Konzeptualisierung* der IS-Domäne bzw. eine „Miniwelt“ [Lockemann u. Schmidt 1987, Kap. 5.1] zu Grunde, vgl. [Genesereth u. Nilsson 1987]:

A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose.

Eine Datenbank bzw. ein Informationssystem wird klassischerweise auf drei unterschiedlichen Abstraktionsebenen bzw. Modell-Ebenen betrachtet [Lockemann u. Schmidt 1987, Kap. 5]:

1. Konzeptuelle Ebene,
2. Implementationsebene,
3. Physische Ebene.

Beim Datenbankentwurf wird die konzeptuelle Ebene mittels abstrakter Entwurfsmittel, z.B. ER-Diagrammen, entworfen. Statt von konzeptuellen (Daten-) Modellen spricht man auch von „semantischen“ Modellen. Als Entwurfsergebnis erhält man beispielsweise ein E(E)R-Diagramm bzw. -Schema. Ein solches Schema kann als eine Konzeptualisierung betrachtet werden. Auf der Implementationsebene wird dieses konzeptuelle Modell dann auf ein logisches Datenmodell abgebildet, z.B. eine relationales Schema (dem relationalen Datenmodell entsprechend). Schließlich wird das logische Modell auf physischer Ebene durch Speicherstrukturen realisiert.

Konzeptuelle Modelle können jedoch nicht nur mittels ER-Diagrammen spezifiziert werden. Wir werden später demonstrieren, dass Logik sich ebenfalls zur Darstellung einer Konzeptualisierung eignet.

Schefe [Schefe 1991, Kap. 4.3.1] nennt prinzipiell mögliche „epistemologische Kategorien“ einer solchen Konzeptualisierung:

- Objekte bzw. Individuen.
- Eigenschaften (Attribute) von Objekten.
- Relationen bzw. Assoziationen zwischen Objekten. Ausgewählte Assoziationen sind die *Aggregation* und die *Komposition*. Eine *Mereologie* beschreibt den kompositionalen Aufbau von Dingen.
- Klassen. Eine Klasse fasst eine Menge von unter gewissen Aspekten als äquivalent betrachteten Objekten zusammen. Diese Menge wird auch die *Extension* der Klasse genannt. Klassen können mittels hinreichender und/oder notwendiger Bedingungen (z.B. prädikatenlogisch) definiert

werden. Eine *Klassendefinition* wird hinreichende und notwendige Bedingungen verwenden. Die Beschreibung wird auch die *Intension* der Klasse genannt. Genauere Aussagen hierzu sind in Kap. 2.2.2 zu finden.

- Beziehungen zwischen Klassen. Klassen können in einer *Generalisierungs-* bzw. *Spezialisierungsbeziehung* zueinander stehen, entweder explizit per Definition („Mütter sind Frauen mit Kindern“), oder notwendigerweise implizit anhand ihrer Definitionen („Großmütter sind Frauen deren Kinder Mütter sind“; hieraus folgt, dass Großmütter auch Mütter sind, obwohl dies nicht explizit gesagt wurde). Die Generalisierungshierarchie wird auch *Taxonomie* genannt; der Begriff stammt aus der Biologie.

Während die in Datenbanken und Informationssystemen verwendeten konzeptuellen Modelle am Ende nicht mehr zur Informationsverarbeitung verwendet werden, da sie in logische und physische Schemata abgebildet, ja gleichsam „wegkompiliert“ werden (wobei es sich um einen Informationsverlust handelt), wird für ein *wissensbasiertes System* angestrebt, dass eine *explizite und symbolische Repräsentation* der Konzeptualisierung erhalten bleibt, die als das im System kodierte „Wissen“ erkennbar ist und die maßgeblich für das „beobachtbare Verhalten“ des Systems verantwortlich ist.

Die sogenannte *Wissensrepräsentationshypothese (WRH)* besagt, dass das Verhalten eines „mechanisch verkörperten intelligenten Prozesses“ (wie auch immer intelligent definiert sei) bzw. eines mechanischen, kognitiven rationalen Agenten aufgrund von *explizit repräsentiertem Wissen (einer Wissensbasis)* erzeugt wird [Smith 1985]:

Any mechanically embodied intelligent process will be comprised of structural ingredients that a) we as external observers naturally take to represent a propositional account of the knowledge that the overall process exhibits, and b) independent of such external semantic attribution, play a formal but causal and essential role in engendering the behaviour that manifests the knowledge.

Zwei Punkte sind hier zentral:

1. Der Agent verfügt über Strukturen, die von externen Beobachtern als *explizite propositionale (symbolische) Repräsentationen von Wissen* angesehen werden können.
2. Diese Repräsentationen bedingen das beobachtbare Verhalten kausal. Die Existenz von Wissen wird durch das Verhalten manifestiert.

Das formal repräsentierte Wissen soll also nicht fest in das System „einkompiliert“ werden; stattdessen impliziert die Wissensrepräsentationshypothese eine Grobstruktur für die Architektur eines wissensbasierten Systems, die aus *Wissensbasis* und *Inferenzmechanismus* besteht. Ein System, welches den Design-

2.1 Logikbasierte Wissensrepräsentation

Anforderungen der Wissensrepräsentationshypothese genügt, wird *wissensbasiertes System (WBS)* genannt. Ein WBS kann daher deklarativ programmiert werden – ein Auswechseln oder Verändern der Wissensbasis ist ausreichend, um das Verhalten zu verändern. In klassischen ISen müssten die Interpretationsprogramme hingegen neu implementiert bzw. durch Programmierung angepasst werden. Wesentliches Charakteristikum von WBSen ist also der *generische Charakter*.

Gemäß Genesereth ist eine *Konzeptualisierung die Basis einer formalen Wissensrepräsentation* [Genesereth u. Nilsson 1987]:

A body of formally represented knowledge is based on a conceptualization: the objects, concepts, and other entities that are assumed to exist in some area of interest and the relationships that hold among them.

Die für WBSen betrachteten Konzeptualisierungen sind komplexer und ausdrucksstärker als die klassischer DBSe und ISe; sie verwenden i.d.R. eine tiefere Modellierung der Domäne. Dementsprechend hat ein WBS „mehr Wissen“, und es ist zu erwarten, dass es in gewisser Weise „kompetenter“ ist. Konzeptualisierungen für DBSe und ISe beschreiben im Wesentlichen *faktuelles Weltwissen*, etwa in einem Personalinformationssystem durch Fakten wie „Die Kontonummer von Michael Wessel ist 1234567“. Eine Datenbank können wir als eine einfache Faktenbasis begreifen; es handelt sich um eine sehr einfache Wissensrepräsentation [Genesereth u. Nilsson 1987].

In einer Datenbank wird typischerweise angenommen, dass alle relevanten Sachverhalten durch Fakten repräsentiert sind. Hier wird die *Annahme der Weltabgeschlossenheit (Closed World Assumption, CWA)* gemacht. Ein Faktum, das nicht in der Datenbank vorhanden ist, wird als *falsch* angesehen, vgl. [Genesereth u. Nilsson 1987, Kap. 6.1]. Dies ist in einem WBS i.d.R. nicht der Fall, da die WBSen zugrundeliegenden Konzeptualisierungen meist *unvollständige bzw. indefinite Information* zulassen. Statt der CWA gilt in WBSen also meist die *Open World Assumption (OWA)*. Wir kommen hierauf in Kap. 2.2.9.2 zurück.

Die für WBSen betrachteten Wissensrepräsentationen erlauben hingegen nicht nur Fakten bzw. faktuelle Aussagen zu repräsentieren, sondern allgemeiner, beliebige logische Aussagen zu machen, z.B. Aussagen wie „Alle Frösche sind grün“. Dementsprechend kann nicht nur Faktenwissen, sondern, in Abhängigkeit von der Expressivität der Wissensrepräsentationssprache, auch Wissen über Klassen von Objekten, Beziehungen zwischen Objekten und Klassen, Regelwissen, Wissen über Prozesse etc. formalisiert und repräsentiert werden.

Ausdrucksstarke Repräsentationssprachen erlauben auch die Repräsentation *unvollständiger und indefiniter Information* (Aussagen wie „Frösche sind grün oder braun“; die Farbe eines einzelnen Frosches muss nicht bekannt sein), „Eine Mutter hat mindestens ein Kind“ (das Kind muss nicht bekannt sein, existiert jedoch), etc.

Nach [Brachman u. Levesque 2004] ist eine (symbolische, propositionale) Wissensrepräsentation eine Menge von Sätzen in einer symbolischen Sprache (bzw. Logik), die eine Menge von *Aussagen (Propositionen)* repräsentiert. Die repräsentierte Welt ist also eine Menge von Aussagen, und die repräsentierende Welt eine Menge von (syntaktischen) Sätzen. Dementsprechend definieren sie das Forschungsfeld der *Wissensrepräsentation* in [Brachman u. Levesque 2004, S. 4] wie folgt:

Knowledge representation, then, is the field of study concerned with using formal symbols to represent a collection of propositions believed by some putative agent.

Diese Definition macht deutlich, dass hier ein *subjektiver* Wissensbegriff vorliegt. Bei der Wissensrepräsentation geht es also nicht darum, eine IS-Domäne oder die Welt abzubilden, sondern den mentalen Wissens- oder Glaubenszustand des Agenten. Nehmen wir jedoch an, dass der Agent über Weltwissen verfügt (oder vielleicht sogar allwissend ist), so wird deutlich, dass der Wissensrepräsentationsbegriff indirekt auch die bisher diskutierten Repräsentationsbegriffe subsumiert.

2.1.2.3. Rationale Agenten und die Architektur wissensbasierter Systeme

Offensichtlich benötigt der Mensch *Wissen*, um sich in der Welt zurechtzufinden, die Auswirkungen seiner Handlungen zu planen, seine Ziele zu erreichen, ja, um sich „rational und intelligent“ zu verhalten. In der KI wird die Metapher des mechanischen *rationalen Agenten* verwendet, wobei es sich um ein sog. *Leistungsmodell* (ähnlich dem Turing-Test) handelt. Ein rationaler Agent wird eine Aktion durchführen, wenn er dadurch seinen Zielen näher kommt (er handelt nicht irrational).

Im Unterschied zu einer Datenbank wird ein solcher Agent nun *mehr als eine Ausprägung der Welt* für möglich halten, insbesondere weil er aufgrund seiner sensorischen Beschränkungen mit unvollständigem Weltwissen operieren muss. Soll sich der Agent in der Welt intelligent und rational verhalten, so benötigt er nicht nur durch seine Perzeptoren/Sensoren (durch Beobachtungen) angekoppeltes Faktenwissen über die Welt, sondern auch *Hintergrundwissen*. Hierzu gehört Wissen über Gesetzmäßigkeiten in der Domäne, Wissen wie Ziele erreicht oder gewisse Welt-Zustände herbeigeführt werden können (Hand-

2.1 Logikbasierte Wissensrepräsentation

lungswissen, prozedurales Wissen), Wissen über das eigene Wissen (autoepistemisches Wissen), etc. Erst durch eine Kombination verschiedener „Wissensarten“ kann der Agent kompetent und rational handeln. Aus dieser Perspektive können wir *Wissen als erforderliche Information* definieren, die die (beobachtbare) Kompetenz des Agenten erst ermöglicht.

Die sogenannte *Wissensrepräsentationshypothese* besagt nun, wie bereits diskutiert, dass das beobachtete intelligente (rationale) Verhalten des *mechanischen*⁹ rationalen Agenten aufgrund der Wissensbasis sowie der Inferenzprozeduren erzeugt wird. Zeigt ein mechanischer Prozess bzw. rationaler Agent also intelligentes Verhalten, so sagt die WRH aus, dass dieses aufgrund der Wissensbasis erzeugt wird. Das Vorhandensein einer Wissensbasis ist nach der WRH also eine *notwendige Voraussetzung* für mechanisches intelligentes Verhalten. Die *hinreichende Bedingung* wird hingegen von der *Hypothese vom physikalischen Symbolsystem (Physical Symbol System Hypothesis)* bereitgestellt [Newell u. Simon 1976]: Diese besagt, dass ein (wie auch immer gearteter) symbolischer Kalkül sowohl die notwendigen *als auch die hinreichenden Voraussetzungen* für mechanisches intelligentes Verhalten erfüllt.

Die Grobstruktur eines WBSes wurde durch Aufteilung in „Wissensbasis + Inferenzkomponente“ bereits „festgelegt“. Newell schlägt eine grobgranulare Schichten-Architektur für ein WBS vor [Newell 1982]:

1. Eine Wissensebene (Knowledge Level).
2. Eine Symbolebene (Symbol Level).

Die Wissensebene entspricht der Ebene des konzeptuellen Schemas, während die Symbolebene die Implementierungsebene ist; man kann auch eine Analogie zu einem Abstrakten Datentypen (ADT) und seiner Implementierung ziehen.

Auf der Wissensebene kann der Wissenszustand des WBSes bzw. Agenten mittels *epistemologischer Grundoperationen* verändert werden; die Operationen „Tell“ und „Ask“ werden als Operationen eines Abstrakten Datentypen „Wissensbasis“ betrachtet. *Logik* wird als implementierungsunabhängige deklarative Beschreibungssprache für diese Ebene vorgesehen.

Die Symbolebene beschreibt hingegen die Implementierung dieses abstrakten Datentypen „Wissensbasis“. Hier werden die Sätze der Repräsentationssprache als Datenstrukturen dargestellt, und letztlich auch die Inferenzprozeduren implementiert.

Verfeinerte Ebeneneinteilungen existieren: Schefe [Schefe 1991] unterscheidet beispielsweise zwischen der Ebene der epistemologischen Repräsentation, der Ebene der logischen Rekonstruktion (der epistemologische Zustand wird hier *logisch rekonstruiert*, also mit Mitteln der Logik beschrieben), und der Ebene

⁹Die Hypothese ist nicht auf Menschen anwendbar.

der datentechnischen Realisierung.

Warum sollte nun Logik als Sprache für die Wissensebene (bzw. als Wissensrepräsentationssprache) verwendet werden? Hier gibt es eine ganze Reihe von Gründen, vgl. auch [Brachman u. Levesque 2004, Kap. 1.3]. Die wichtigsten sind nach Meinung des Autors:

1. Mathematische Logik kann kalkülisiert und somit mechanisiert werden. Die durch einen korrekten Kalkül abgeleiteten Theoreme bzw. Konsequenzen einer Wissensbasis sind „wahr“ und gelten somit in der repräsentierten Welt. Ein vollständiger Kalkül garantiert zudem eine gewisse „Handlungskompetenz“.
2. Die Leistungen eines WBSes können rein deklarativ und formal beschrieben werden. Die Leistungen sind somit nachprüfbar und prinzipiell auch *versteh- und kommunizierbar* (in Form von Beweisen). Diese Eigenschaft wurde nach Dennett als *Intentional Stance* bezeichnet; statistischen und konnektionistischen Repräsentationen mangelt es hieran. Diese erfüllen zudem die WRH nicht.
3. Logische Formeln sind explizit und symbolisch und erfüllen somit die Voraussetzungen der WRH. Sie sind insbesondere auch für Menschen lesbar, erstellbar, verstehbar, wartbar, und können somit als „Austauschformat“ für eine Schnittstelle zwischen menschlicher und maschineller Wissensrepräsentation dienen.

2.1.2.4. Konzeptualisierungen und Formelmengen

Warum sollte eine Menge von Formeln $WB \subseteq \mathcal{L}_{FOP\mathcal{L}}$ (oder einer anderen Logik) nun eine Wissensrepräsentation sein, also Wissen repräsentieren? Ausgangspunkt ist zunächst wieder die Konzeptualisierung bzw. das konzeptuelle Modell der Domäne:

A body of formally represented knowledge is based on a conceptualization: the objects, concepts, and other entities that are assumed to exist in some area of interest and the relationships that hold among them.

A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose.

Die Konzeptualisierung wird nun in eine Menge logischer Formeln WB abgebildet („a body of formally represented knowledge“).

Meist werden die Axiome der Wissensbasis WB *Sätze* sein (also Formeln ohne freie Variablen). Zunächst einmal ist durch WB nicht nur *explizites* Wissen, sondern auch *implizites* Wissen, nämlich die Theorie von WB re-

2.1 Logikbasierte Wissensrepräsentation

präsentiert. Man könnte somit auch sagen, dass eine Wissensbasis ihre Theorie repräsentiert. Treffender ist jedoch die semantische Betrachtungsweise, dass durch $\mathcal{WB}$ *mögliche Ausprägungen der Domäne* bzw. *mögliche Ausprägungen der Konzeptualisierung* repräsentiert werden, die beschreiben, wie die Domäne bzw. Welt „wohl sein könnte“. Diese möglichen Ausprägungen sind durch die *logischen Modelle* von $\mathcal{WB}$ gegeben. Wie oben diskutiert, ist es präziser zu sagen, dass hiermit nicht mögliche Ausprägungen der Welt repräsentiert werden, sondern mögliche Wissens- oder Glaubenszustände eines kognitiven Agenten (die somit den Glauben oder das Wissen des Agenten darstellen, wie die Welt wohl sein könnte oder ist). Wir ignorieren im Folgenden diese subjektive Dimension und betrachten auch nichtsubjektives, z.B. Allgemein- oder Welt-Wissen als „Wissen“ (auch dieses kann ja, wie erläutert, subjektiv sein).

Durch die logischen Modelle von $\mathcal{WB}$ ist daher Wissen über die Domäne repräsentiert, vorausgesetzt, diese logischen Modelle stehen in einer wahrheits-erhaltenden Modellierungs-Relation mit der Domäne (die Konzeptualisierung ist sinnvoll).

Ziel der logischen Wissens- bzw. Domänenmodellierung ist es nun, die logischen Modelle von $\mathcal{WB}$ soweit einzuschränken, dass lediglich solche logischen Modelle von $\mathcal{WB}$ übrig bleiben, die in Bezug auf die Konzeptualisierung und somit in Bezug auf die modellierte Domäne als *sinnvoll* bzw. *wahrheitserhaltend* interpretiert werden können. Die möglichen logischen Modelle von $\mathcal{WB}$ werden daher durch Angabe von Axiomen bzw. Sätzen in $\mathcal{WB}$ solange reduziert werden, bis nur noch solche Modelle übrig bleiben, die „sinnvoll“ in Bezug auf die Domäne bzw. Konzeptualisierung erscheinen. Eine Reduktion von möglichen Modellen entspricht daher einem *Informationszuwachs*. Betrachten wir zur Illustration folgendes Beispiel:

- Die leere Wissensbasis (leere Menge von Axiomen) hat *alle Interpretationen* als Modelle. Dies bedeutet, dass ihre Theorie (also der deduktive Abschluss) aus allen Tautologien besteht. Zunächst einmal ist hiermit keine Konzeptualisierung gegeben – wir halten somit alles für möglich in Bezug auf die Domäne. Hiermit ist *kein domänenspezifisches* Wissen kodiert, sondern lediglich Tautologien. Es wird somit „zu wenig“ repräsentiert.
- Fügen wir nun ein Faktum der Art *informatiker(michael)* hinzu, so bedeutet dies, dass nur noch die Interpretationen möglich sind, für die $michael^I \in informatiker^I$ gilt.

Offensichtlich fallen alle Interpretationen mit $michael^I \notin informatiker^I$ (und somit $michael^I \in (\neg informatiker)^I$) weg.

Durch ein solches Faktum (bzw. Axiom) wird also die Menge der möglichen logischen Modelle reduziert. Diese Reduktion von Modellen bzw. „repräsentierten möglichen Welten“ kann als *Informationszuwachs* gedeutet

werden (ähnlich wie in der Informations-Theorie).

Einige Wissensbasen werden die Anzahl ihrer möglichen Modellen sogar soweit reduzieren, dass sie in gewisser Weise als *isomorph* zur Menge der beschreibenden Formeln selbst aufgefasst werden können. Dies ist z.B. für relationale Datenbanken (s.u.) der Fall. Bei solchen Wissensbasen verschwindet in gewisser Weise der Unterschied zwischen Syntax und Semantik.

- Wir können nun noch ein weiteres sinnvolles Axiom hinzufügen, nämlich dass alle Informatiker Menschen sind:

$$\forall x.informatiker(x) \rightarrow mensch(x).$$

Offensichtlich ist hiermit ausgesagt, dass die möglichen Interpretation der Prädikate *informatiker* und *mensch* nicht unabhängig voneinander sind, sondern stets in einer Teilmengenbeziehung zueinander stehen (Spezialisierungsbeziehung).

Somit wird Domänen-Wissen repräsentiert, und *bzgl. der Domäne wahrheitserhaltende Ableitungen* werden durch logische Deduktion ermöglicht, vgl. Abb. 2.1. Nun ist $mensch(michael)$ eine logische Konsequenz aus WB . Es werden also nur noch die Interpretationen bzw. Modelle zugelassen, die sicherstellen, dass Informatiker auch Menschen sind: $michael^I \in mensch^I$.

- Eine Menge von *unerfüllbaren* Axiomen lässt schließlich gar keine Modelle zu. Es ist daher sinnvoll zu sagen, dass eine inkonsistente Menge von Axiomen *nichts* repräsentiert.

2.1.2.5. Relevante Begriffe aus der Wissensorganisation

Das Gebiet der *Wissensorganisation*, einem Teilgebiet der *Dokumentationswissenschaft*, beschäftigt sich mit der Frage, wie das durch Dokumente (welcher Art auch immer) repräsentierte „Wissen“ so zu organisieren, indexieren und archivieren ist, dass diese Dokumente (bzw. das durch sie repräsentierte Wissen) bei Bedarf möglichst „einfach“ (bzw. mit gutem „Precision and Recall“, s. Kap. 3) wieder aufzufinden ist, vgl. [Wikipedia 2006b]. Insofern bestehen viele Querbezüge zum Bibliothekswesen und dem Forschungsfeld der *Informationswiedergewinnung* (*Information Retrieval*, *IR*) und somit Dokumentensuche [Baeza-Yates 1999].

Zur Erschließung werden Dokumente (thematisch) mit *Schlagworten* (auch: *Termen*, *Begriffen*, *Kategorien*, etc.) annotiert und indexiert; es existieren jedoch auch rein statistische Verfahren, die ohne explizite Verschlagwortung auskommen, z.B. für das IR im World Wide Web, wo oftmals reine Volltext-Suche angewendet wird. Bei der manuellen Annotation stammen die Schlagworte jedoch

2.1 Logikbasierte Wissensrepräsentation

aus einem sog. *kontrollierten Vokabular*. Hierdurch wird im Wesentlichen das Problem des „Wildwuchses“ von Schlagworten entschärft. Gleichzeitig ist durch ein kontrolliertes Vokabular auch eine Art (wenn auch nicht formale) *gemeinsame Konzeptualisierung* der Domäne gegeben. Schlagworte – oder allgemeiner: Annotationen – können zudem in bestimmten Beziehungen zueinander stehen, sodass nicht nur die mit dem Schlagwort direkt annotierten Dokumente wiedergefunden werden können, sondern aufgrund der Schlagwort-Beziehungen auch *speziellere, allgemeinere* oder *ähnliche* anderen Dokumente. Hier sind beliebige weitere semantische Beziehungen zwischen Schlagworten bzw. Begriffen (und somit indirekt zwischen Dokumenten) denkbar.

Je nach Art und Struktur der im kontrollierten Vokabular verwalteten semantischen Beziehungen unterscheidet man unterschiedlich stark strukturierte Arten kontrollierter Vokabularen. Der *Thesaurus* ist traditionell am stärksten strukturiert; IR mit Hilfe von Ober- und Unterschlagworten ist heutzutage jedem Anwender nicht zuletzt durch Internet-Auktionshäuser, digitale Bibliotheksauskunftssysteme usw. geläufig:

Definition 25 (Thesaurus) Ein Thesaurus ist ein kontrolliertes Vokabular, also eine Sammlung von *Begriffen* (auch: *Termen*, *Schlagworten*, *Konzepten*). Für einen Begriff werden im Thesaurus auch seine *Synonyme*, sowie seine *Ober- und Unterbegriffe* genannt. Weitere Assoziationen können zur Repräsentation weiterer Beziehungen zwischen Begriffen vorkommen (z.B. zur Referenzierung ähnlicher oder gegensätzlicher Begriffe, sog. *Antonyme*). ■

Ein flacher (nichthierarchischer) Thesaurus wird oft *Begriffsliste* genannt. Kommt in einem Thesaurus lediglich die Generalisierungsbeziehung (und evtl. noch die Synonymbeziehung) als mögliche Assoziation vor, so spricht man auch von einer *Taxonomie* oder einem *Kategoriensystem*. Traditionell spielen diese Wissensorganisations-Strukturen eine wichtige Rolle im IR.

Aus der Perspektive der Wissensrepräsentation können diese Wissensorganisations-Strukturen jedoch auch als einfache *Ontologien* angesehen werden, vgl. [Lassila u. McGuinness 2001]. Die *Ontologie* (als philosophische Teildisziplin) ist die „Lehre vom Seienden / Existierendem“ und untersucht somit die Fragestellung, was in der Welt existiert, welche Struktur die Entitäten der Welt, und in welchen Beziehungen diese Entitäten zueinander stehen. Im Gegensatz hierzu werden in der *Epistemologie* fundamentale *erkenntnistheoretische Fragestellungen* untersucht, die sich auf den Wissenszustand eines erkennenden Subjektes bzw. rationalen Agenten beziehen, also Fragestellungen der Art „Was kann ein Agent prinzipiell über die Welt durch Wahrnehmung erfahren bzw. erkennen?“, „Was hat der Wissenszustand des Agenten mit der realen Welt zu tun?“, etc.

Eine Ontologie ist jedoch auch eine Wissensorganisationsstruktur bzw. eine

Wissensrepräsentation; wir hatten bereits in Kap. 1 die Definition von Gruber zitiert [Gruber 1993a;b]:

Definition 26 (Ontologie) Eine Ontologie ist eine Spezifikation einer Konzeptualisierung. ■

In dieser Arbeit werden ausschließlich sog. *Domänenontologien* betrachtet; andere Typen von Ontologien werden u.a. in [Guarion 1998] und [Görz u. a. 2003, Kap. 15.3.2] diskutiert. Einer Domänenontologie liegt eine Konzeptualisierung einer speziellen Domäne zugrunde, z.B. die Domäne der Genealogie, im Gegensatz zu einer generischen „Upper Ontology“, in der Allgemeinwissen oder generisches Wissen modelliert wird (die berühmte CYC-Ontologie ist beispielsweise eine „Upper Ontology“ [Lenat u. Guha 1990, Lenat 1995]).

Wird eine Ontologie von mehreren Agenten verwendet, so ist durch das gemeinsam verwendete Vokabular der Ontologie eine Art gemeinsamer Kommunikationsbasis gegeben. Die Verwendung der Ontologie bedeutet für den Agenten dann, eine *ontologische Bindung* einzugehen, da das Vokabular der Ontologie ja akzeptiert und in ihren Begriffen, Relationen und Definitionen in gewisser Weise kommuniziert, repräsentiert und inferiert werden muss. Die strengere Variante der Ontologie-Definition von Gruber betont diesen Aspekt der geteilten, gemeinsamen Verwendung einer Konzeptualisierung (*Hinzufügungen des Autors in Kursivschrift*):

Eine Ontologie ist eine formale Spezifikation einer geteilten (*gemeinsam verwendeten*) Konzeptualisierung.

Offensichtlich ist z.B. für das IR im World Wide Web eine Art „gemeinsamer Konzeptualisierung“ für die Recherche erforderlich. Diese gemeinsame Konzeptualisierung ist heutzutage durch die natürliche Sprache bzw. das allen Menschen gegebene allgemeine Hintergrundwissen gegeben (jedoch auch nicht sprachübergreifend), da heutzutage primär Volltextsuche verwendet wird. Es ist bekannt, dass Volltextsuche Defizite hat und u.U. zu schlechter Präzision und Vollständigkeit („Precision and Recall“, s. Kap. 3) bei der Recherche führt, da das menschliche Hintergrundwissen nicht maschinell auswertbar repräsentiert ist (es war gerade ein Forschungsziel des CYC-Projekts, dieses Wissen zu konzeptualisieren). Eine tiefe Erschließung der *semantischen Inhalte* von Webressourcen erfordert daher andere Techniken.

Durch die explizite Annotation von Webressourcen mittels eines kontrollierten Vokabulars kann das Problem (zu Kosten des Annotationsaufwands) gelöst werden. Das kontrollierte Vokabular stellt dann die gemeinsam verwendete Konzeptualisierung bereit. Allerdings ist auch hier die Erschließung oftmals nicht tief genug; die Vision des SEMANTIC WEBS geht von einer umfassenden Erschließung mit logikbasierten Annotation aus (s. Kap. 3), sodass maschinen-auswertbare „Inhaltsbeschreibungen“ für Webressourcen vorliegen werden, die

2.1 Logikbasierte Wissensrepräsentation

mittels logischer Deduktion maschinell verarbeitet werden können. Natürlich müssen auch hier die Annotationen eine Art „gemeinsame Vokabularbasis“ verwenden, indem sie z.B. (standardisierte) öffentlich im Web zugängliche Ontologien erweitern bzw. verwenden. Die heutzutage standardisierten generellen Vokabularien sind jedoch oftmals noch primitiv (ganz im Gegensatz zu Domänenontologien, z.B. medizinischen Ontologien wie SNOMED).

Die Verwendung von logikbasierten Annotationen erlaubt eine sehr viel tiefgehendere Erschließung der Dokumenten- bzw. Webressourcen-Inhalte. Die Vision des SEMANTIC WEBS ist nicht auf IR beschränkt; Ressourcen können z.B. auch Web Services sein, deren Dienste beschrieben wurden, sodass ontologiebasiertes Service-Matchmaking etc. stattfinden kann.

Ein Hauptschwerpunkt dieser Arbeit liegt auf ontologiebasierten Anfragebeantwortungssystemen. Ontologiebasierte Anfragebeantwortung im SEMANTIC WEB kann auch als semantisches IR gedeutet werden, woraus sich die Relevanz der soeben gemachten Ausführungen für diese Arbeit ergibt.

2.1.2.6. Wissensrepräsentationssprachen

Wir haben bisher $\mathcal{F}_{FOP\mathcal{L}}$ als konkrete Wissensrepräsentationssprache kennengelernt. Es gibt jedoch eine Fülle weiterer Wissensrepräsentationssprache, die meist auf $\mathcal{L}_{FOP\mathcal{L}}$ abgebildet werden können; $\mathcal{L}_{FOP\mathcal{L}}$ ist daher eine Art „Lingua Franca“. Tatsächlich existieren auch Sprachen, die Konstruktoren enthalten deren Abbildung Prädikatenlogik zweiter Ordnung benötigen würde (z.B. Sprachen mit Operator zum transitivem Abschluss einer Relation); solche Sprachen werden im Folgenden jedoch nicht betrachtet. In dieser Arbeit werden wir das relationale Datenmodell (das als eine sehr einfache Wissensrepräsentationssprache betrachtet werden kann), Beschreibungslogiken, und die SEMANTIC WEB-Repräsentationssprachen OWL und RDFS verwenden (s. Kap. 3).

Verschiedene (Wissens-) Repräsentationssprachen können anhand ihrer *Ausdrucksmächtigkeiten*, und anhand ihrer *ontologischen und epistemologischen Grundannahmen* miteinander verglichen werden:

- Wir sagen, dass eine Sprache $\mathcal{L}_2$ ausdrucksstärker als eine Sprache $\mathcal{L}_1$ ist, wenn für jede Formel $F \in \mathcal{L}_1$ eine Formel $G \in \mathcal{L}_2$ existiert, die die gleichen Sachverhalte repräsentiert. Wie erläutert können die repräsentierten Sachverhalte mit den *logischen Modellen* der Formeln (mehr oder weniger direkt) gleichgesetzt werden. Dementsprechend kann man z.B. elementare Klassen identifizieren, die sich zwar mit $\mathcal{L}_1$, nicht jedoch mit $\mathcal{L}_2$ axiomatisieren lassen (z.B. die Klasse der natürlichen Zahlen).

Die Ausdrucksmächtigkeit von $\mathcal{L}_i$ wird wesentlich durch die angebotenen

Operatoren (Formelkonstruktoren) bestimmt. Es ist wohlbekannt, dass Ausdrucksmächtigkeit oft mit höherer Komplexität (oder gar Unentscheidbarkeit) von Inferenzproblemen einhergeht.

Eine andere Definition verwendet lediglich Erfüllbarkeitsäquivalenz: Demnach ist die Sprache $\mathcal{L}_1$ ausdrucksstärker als $\mathcal{L}_2$, wenn für jede Formel $F \in \mathcal{L}_2$ eine erfüllbarkeitsäquivalente Formel $G \in \mathcal{L}_1$ existiert.

- Die *ontologischen Grundannahmen* [Russel u. Norvig 2003, S. 242] einer Repräsentationssprache beschreiben, wie *die Welt* prinzipiell mit der Repräsentationssprache abgebildet bzw. konzeptualisiert werden kann, und *welche Art von logisch-mathematischen Modellen* damit als prinzipiell mögliche Repräsentationen der Welt betrachtet werden.

So können wir unter Verwendung von $\mathcal{L}_{PROP}$ z.B. lediglich Aussagen (Propositionen) über die Welt repräsentieren, jedoch keine Objekte oder Relationen. Dies ist hingegen mit $\mathcal{L}_{FOP\mathcal{L}}$ möglich. Eine wiederum anderer Konzeptualisierungsrahmen ist durch *Fuzzy-Logik* gegeben, der die Repräsentation „vager“ oder „unscharfer“ Aussagen zulässt (z.B. kann hier das Prädikat *groß* auf eine Person zu 70% zutreffen - somit ist *graduelle Mitgliedschaft* in einer Menge modellierbar).

Wird die Sprache dann zur Repräsentation verwendet, so werden diese Grundannahmen zu *ontologischen Bindungen* bzw. *Verpflichtungen*. Die Welt kann dann nur auf eine gewisse Art und Weise abgebildet bzw. „durch eine gewisse Brille“ gesehen werden; prinzipiell andere Konzeptualisierungen sind dann nicht mehr möglich.

- Die *epistemologischen Grundannahmen* [Russel u. Norvig 2003, S. 244] einer Repräsentationssprache beziehen sich hingegen darauf, wie *Wissen über die Welt* mit der Sprache abgebildet werden kann. Diese Perspektive betont daher den Wissenszustand eines Agenten. So kann ein Agent, der $\mathcal{L}_{FOP\mathcal{L}}$ oder $\mathcal{L}_{PROP}$ verwendet, bestimmte Propositionen bzw. Fakten entweder für wahr oder falsch *halten*, oder kein Wissen diesbez. haben. *In der Welt* sind Propositionen jedoch nach wie vor wahr oder falsch (sofern nicht Fuzzy-Logik o.ä. verwendet wird).

Eine weitere Differenzierung dieser Wissenszustände des Agenten ist beispielsweise mit *Modallogiken* möglich, die die Modellierung verschiedener *Wissensmodalitäten* ermöglichen (z.B. Glaube und Wissen). Probabilistische Logiken ermöglichen ebenfalls eine Modellierung des „Grades des Glaubens“ (der zwischen 0 und 1 liegen kann) an einen bestimmten Sachverhalt bzw. eine Proposition.

2.2. Beschreibungslogiken

In diesem Abschnitt widmen wir uns den Beschreibungslogiken, die im weiteren Verlauf der Arbeit die Hauptrolle spielen werden. Als umfassende Referenz sei das Beschreibungslogik-Handbuch genannt [Baader u. a. 2003a].

Wir stellen zunächst die Historie der Beschreibungslogiken dar. Anhand der „einfachen“ Sprache $\mathcal{ALC}$ werden alle Grundbegriffe (Syntax, Semantik, Standard-Inferenzprobleme) eingeführt. Alsdann werden ausdrucksstärkerer DLs als $\mathcal{ALC}$ vorgestellt. Wir stellen eine wichtige Klasse von Inferenzverfahren für DLs vor, die sog. Tableau-Kalküle zur Entscheidung der Erfüllbarkeitsprobleme. Danach werden die Leistungen sowie die historische Entwicklung von DL-Systemen geschildert. Insbesondere diskutieren wir das RACERPRO-System als ein konkretes DL-System, welches im weiteren Verlauf der Arbeit eine tragende Rolle spielen wird. Schließlich folgt ein Vergleich mit dem relationalen Datenmodell und ein Vergleich der Repräsentationsdienste und -leistungen von DL-Systemen mit denen relationaler DBMSen.

2.2.1. Historie

Beschreibungslogiken (Description Logics, DLs) sind Nachfahren der sog. *Semantischen Netze*, die auf Quillian's „Semantic Memory Models“ zurückgeführt werden [Quillian 1968], und der von Minsky eingeführten *Frame-Systeme* zur Repräsentation stereotypen, prototypischen Wissens [Minsky 1974].

Sowohl Frame-Systeme als auch semantische Netze wurden durch kognitive Modelle der menschlichen Informationsverarbeitung motiviert. Rein logikbasierte Wissensrepräsentation wurde damals von vielen Forschern als kognitiv nicht adäquat und nicht effizient implementierbar angesehen. Ähnlich kontrovers wurde diskutiert, ob Wissensrepräsentation eher mit prozeduralen oder deklarativen Mitteln geschehen sollte. Minsky führte die Frames explizit als „nichtlogisches“ Repräsentationsmittel ein. Frames und Semantische Netze sollten auf der einen Seite effizient implementierbar, und auf der anderen Seite nicht mit den (oben genannten) Nachteilen logikbasierter Repräsentationen behaftet sein. Die Generalisierungsbeziehung zwischen Frames wurde als wesentliche epistemologische Primitive etabliert. Frames sind letztlich Record-ähnliche Strukturen, die in einer Spezialisierungshierarchie (Taxonomie) gehalten werden. Techniken wie *Aktivierungs-Ausbreitung* (*Spreading Activations*) sollten effiziente Inferenz ermöglichen und kognitive Adäquatheit sicherstellen. Durch Aktivierung eines Frames wurden i.d.R. auch benachbarte oder assoziierte Frames mitaktiviert.

Semantische Netze als auch Frame-Systeme werden heute generalisierend als

graphbasierte Wissensrepräsentationen bezeichnet. Bei ihnen handelt es sich letztlich um graph- bzw. netzwerkartige Strukturen, also beschriftete Knoten- und Kanten-Strukturen, in denen Knoten Individuen oder Konzepte (Mengen von Individuen) repräsentieren, die über Kanten bzw. Assoziationen in gewissen Beziehungen zueinander stehen.

Die Semantischen Netze wurden in dem einflussreichen Aufsatz „What’s in a Link“ von Woods [Woods 1975] als semantisch unzureichend und mehrdeutig kritisiert. Insbesondere kritisierte Woods, dass die Semantik der „is-a“-Kanten nicht eindeutig war. Durch eine „is-a“-Kante zwischen zwei Knoten konnte sowohl eine Spezialisierungsbeziehung zwischen zwei Konzepten, als auch eine Instanz-Beziehung zwischen Individuum und Konzept dargestellt werden. Von Hayes wurde eine prädikatenlogische Rekonstruktion der von Minsky explizit als nichtlogisch motivierten Frames angegeben, und er kam in [Hayes 1979] zu dem Schluss:

Most of ‘frames’ is just a new syntax of first-order logic.

Die *vollständige* logische Rekonstruktion von Frames stellte sich jedoch aufgrund der Möglichkeiten, *Default-Informationen* für bestimmte Record-Attribute („Slots“) anzugeben, und aufgrund von prozeduralen Erweiterungen (z.B. zur automatischen Berechnung abgeleiteter Attributwerte) in diesen Sprachen als schwierig heraus, da zur Formalisierung nichtmonotone Logiken erforderlich wurden.

Brachman und Levesque entwickelten das erste DL-System mit formaler Semantik – das KL-ONE-System [Brachman u. Schmolze 1985] [Woods u. Schmolze 1992]. KL-ONE ist der Urvater moderner DL-Systeme. In KL-ONE findet sich bereits die heute noch vorherrschende Aufteilung der Wissensbasis: Die *ABox* zur Modellierung bzw. Beschreibung eines Weltzustands mittels sog. *Zusicherungen* (*Assertionen*), und die *TBox* zur Modellierung *terminologischen Wissens* mittels primitiver und definierter *Konzepte*. Aufgrund der Aufteilung bzw. „Trennung von Wissensarten“ wurde KL-ONE als hybrid bezeichnet. Die Begriffe *Konzept* und *Rolle* wurden etabliert. KL-ONE bot bereits eine reichhaltige Konzept-Beschreibungssprache, die sich später jedoch als unentscheidbar aufgrund der sog. *Role Value Maps* herausstellte [Schmidt-Schauß 1989]. Schon früh wurde also deutlich, dass Ausdrucksmächtigkeit oftmals mit *Unentscheidbarkeit der Inferenzprobleme* einhergeht. Ein wesentlicher Beitrag von KL-ONE ist die Idee der *automatischen Konzept-Klassifikation* – im Gegensatz zu bisherigen Ansätzen zur taxonomischen Wissensrepräsentation bestimmt in KL-ONE die Definition eines Konzeptes seine *korrekte Position in der Taxonomie*. Die Definitionen werden also, anders als in Vorläufersystemen, ernst genommen.

In der Folge wurden daher verschiedenste Repräsentationskonstrukte auf ihre Kombinierbarkeit und Komplexität eingehend untersucht. Ein frühes Papier

2.2 Beschreibungslogiken

zur Identifikation der Grenze zwischen traktablen und intraktablen Konzept-Beschreibungssprachen ist [Brachman u. Levesque 1984]. Hier wird die Sprache $\mathcal{FL}^-$ als traktabel identifiziert. Heutzutage gibt es kaum noch weiße Flecken in der Komplexitätslandkarte für DLs. Entscheidbarkeit der *Standard-Inferenzprobleme* (s.u.) ist heutzutage das oberste Gebot der DL-Forschung.

Die wichtigsten Begriffe einer DL sind also das Konzept, die Rolle, die TBox und die ABox. Verschiedene DLs unterscheiden sich in der Anzahl und Ausdrucksmächtigkeit der gebotenen Konzept- und Rollenoperatoren, sowie in der Art der unterstützten TBox-Axiome.

2.2.2. Die Basis-Beschreibungslogik $\mathcal{ALC}$

Wir stellen zunächst die (aus heutiger Perspektive als einfach zu bewertende) DL $\mathcal{ALC}$ vor. $\mathcal{ALC}$ kann als eine Art „Basis-DL“ angesehen werden. Die Sprache $\mathcal{ALC}$ wurde in [Schmidt-Schauß u. Smolka 1991] durch ihre „schönen“ mathematischen Abschlusseigenschaften bzw. *aussagenlogische Abgeschlossenheit* motiviert. $\mathcal{ALC}$ steht für *Attribute Language with Complement*. Anhand von $\mathcal{ALC}$ werden wir nun die Basisbegriffe Konzept, Rolle, TBox und ABox sowie die sog. Standard-Inferenzprobleme konkretisieren.

2.2.2.1. Die Konzeptsprache

Das Basis-Repräsentationskonstrukt einer DL ist das *Konzept* (auch: *Konzeptterm*). Konzepte können atomar oder komplex bzw. zusammengesetzt sein und sind letztlich (variablenfreie) logische Formeln. Komplexe Konzepte entstehen induktiv, in dem einfachere Konzepte mit Hilfe *konzeptbildender Operatoren* zusammengesetzt werden. Ein erstes Beispiel für ein Konzept ist der Ausdruck

$$frau \sqcap \exists hat\_kind.person.$$

Hier ist „ $\sqcap$ “ ein konzeptbildender Operator, die Konzeptkonjunktion, und *frau* sowie *person* sind atomare Konzepte bzw. Konzeptnamen; *hat_kind* ist eine sog. Rolle. Der Ausdruck $\exists hat\_kind.person$ ist ein komplexes Konzept.

Die Semantik eines Konzeptes wird durch eine Tarski-Interpretation $\mathcal{I}$ angegeben; intuitiv wird durch ein Konzept eine Teilmenge der Domäne $\Delta^{\mathcal{I}}$ denotiert bzw. repräsentiert. Diese Menge enthält alle Objekte, für die das Konzept zutrifft; hierbei handelt es sich um die sog. *Extension* oder Semantik des Konzeptes. Beispielsweise denotiert das Konzept $frau \sqcap \exists hat\_kind.person$ die Schnittmenge der Menge der Frauen und der Menge der Objekte, die mit einem anderen Objekt aus der Menge der Personen in der *hat_kind*-Beziehung stehen (diese Menge wird durch $\exists hat\_kind.person$ denotiert). Um Assoziationen zwischen Objekten repräsentieren zu können, werden Rollen verwendet (*hat_kind* ist eine

Rolle). Intuitiv ist die Semantik bzw. Extension einer Rolle eine binäre Relation auf $\Delta^{\mathcal{I}}$.

Die Syntax von $\mathcal{ALC}$ ist wie folgt definiert:

Definition 27 (Definition $\mathcal{ALC}$ -Konzept) Sei $\mathcal{N}_C$ eine Menge von Konzeptnamen bzw. atomaren Konzepten, und $\mathcal{N}_R$ eine Menge von Rollennamen. Zwei ausgezeichnete Konzeptnamen seien das Top-Konzept und das Bottom-Konzept: $\{\top, \perp\} \subseteq \mathcal{N}_C$.

Ein $\mathcal{ALC}$ -Konzept (kurz: Konzept) ist induktiv definiert:

- Jeder Konzept-Name $CN \in \mathcal{N}_C$ ist ein Konzept.
- Wenn C und D Konzepte sind, und $R \in \mathcal{N}_R$ ein Rollenname ist, so sind auch die folgenden Ausdrücke Konzepte:
 - $\neg C$ (volle Konzeptnegation)
 - $C \sqcap D$ (Konzeptkonjunktion)
 - $C \sqcup D$ (Konzeptdisjunktion)
 - $\exists R.C$ (qualifizierte existentielle Rollenrestriktion)
 - $\forall R.C$ (universelle Rollenrestriktion) ■

Eine wichtige im weiteren Verlauf der Arbeit benötigte Operation extrahiert alle in einem Konzept verwendeten Unter- bzw. Subkonzepte:

Definition 28 (Subkonzepte eines Konzeptes)

$$\text{subconcepts}(C) =_{\text{def}} \begin{cases} \{C\} \cup \emptyset & \text{falls } C \in \mathcal{N}_C \\ \text{subconcepts}(D) & \text{falls } C \in \{\neg D, \exists R.D, \forall R.D\} \\ \text{subconcepts}(D) \cup \text{subconcepts}(E) & \text{falls } C \in \{(D \sqcap E), (D \sqcup E)\} \end{cases}$$

Zudem gelte $\text{subconcept\_of}(C, D)$ gdw. $C \in \text{subconcepts}(D)$. ■

So ist z.B. $\text{subconcepts}((C \sqcap (\forall R.D))) = \{(C \sqcap (\forall R.D)), C, (\forall R.D), D\}$; offensichtlich ist die subconcept_of -Relation eine partielle Ordnung.

2.2.2.2. Semantik der Konzeptsprache

Die Semantik eines Konzeptes wird wie die Semantik einer prädikatenlogischen Formel durch Angabe einer Tarski-Interpretation definiert. Anders als in der Prädikatenlogik muss die Interpretationsfunktion jedoch bereits a priori gewissen Bedingungen genügen, die die intendierte Semantik der Konzeptoperatoren

2.2 Beschreibungslogiken

wieder spiegeln (in der Prädikatenlogik wird das Gelten dieser Bedingungen erst für die Modelle gefordert):

Definition 29 (DL-Interpretation) Eine DL-Interpretation ist ein Paar $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, wobei $\Delta^{\mathcal{I}}$ eine nichtleere Grundmenge (Domäne), und $\cdot^{\mathcal{I}}$ eine Interpretationsfunktion ist, die Konzeptnamen C auf Teilmengen von $\Delta^{\mathcal{I}}$ abbildet ($C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$), und Rollennamen auf binäre Relationen, also Teilmengen von $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ abbildet ($R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$). Zudem soll $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$ und $\perp^{\mathcal{I}} = \emptyset$ gelten.

Die Interpretationsfunktion wird nun per induktiver Erweiterung für beliebige Konzepte wie folgt definiert:

- $(\neg C)^{\mathcal{I}} =_{\text{def}} \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
- $(C \sqcap D)^{\mathcal{I}} =_{\text{def}} C^{\mathcal{I}} \cap D^{\mathcal{I}}$
- $(C \sqcup D)^{\mathcal{I}} =_{\text{def}} C^{\mathcal{I}} \cup D^{\mathcal{I}}$
- $(\exists R.D)^{\mathcal{I}} =_{\text{def}} \{i \in \Delta^{\mathcal{I}} \mid \exists j \in \Delta^{\mathcal{I}}. (i, j) \in R^{\mathcal{I}} \wedge j \in D^{\mathcal{I}}\}$
- $(\forall R.D)^{\mathcal{I}} =_{\text{def}} \{i \in \Delta^{\mathcal{I}} \mid \forall j \in \Delta^{\mathcal{I}}. (i, j) \in R^{\mathcal{I}} \rightarrow j \in D^{\mathcal{I}}\}$ ■

Alternativ kann man auch fordern, dass die Interpretationsfunktion *beliebige* Konzepte (nicht nur Namen) auf Teilmengen von $\Delta^{\mathcal{I}}$ abbildet. Die Interpretationsfunktion wird dann nicht per induktiver Erweiterung für beliebige Konzepte definiert, sondern es wird gefordert, dass die Interpretationsfunktion einen Satz von *Gleichungen (oder Constraints)* erfüllt: $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$, $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$, etc. Beide Formulierungsalternativen sind gleichwertig.

Ein *Modell* für ein Konzept C ist nun eine DL-Interpretation, die C auf eine nichtleere Menge abbildet:

Definition 30 (Modell eines Konzeptes) Sei $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ eine DL-Interpretation. Dann ist $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ ein Modell für ein Konzept C , wenn $C^{\mathcal{I}} \neq \emptyset$ gilt. Wir schreiben dann auch $\mathcal{I} \models C$. ■

Ein Modell für das Beispielkonzept $\text{frau} \sqcap \exists \text{hat\_kind. person}$ muss daher für ein i aus $\Delta^{\mathcal{I}}$ erfüllen:

1. $i \in (\text{frau} \sqcap \exists \text{hat\_kind. person})^{\mathcal{I}}$,
2. $i \in \text{frau}^{\mathcal{I}} \cap (\exists \text{hat\_kind. person})^{\mathcal{I}}$,
3. $i \in \text{frau}^{\mathcal{I}} \cap \{i \in \Delta^{\mathcal{I}} \mid \exists j \in \Delta^{\mathcal{I}}. (i, j) \in \text{hat\_kind}^{\mathcal{I}} \wedge j \in \text{person}^{\mathcal{I}}\}$.

Wir könnten ein Modell wie folgt konstruieren: $\text{frau}^{\mathcal{I}} =_{\text{def}} \{i\}$, $\text{person}^{\mathcal{I}} =_{\text{def}} \{i, j\}$, $\text{hat\_kind}^{\mathcal{I}} =_{\text{def}} \{(i, j)\}$. Ein weiteres Modell ist das eigentlich unintuitive aber bzgl. der Größe von $\Delta^{\mathcal{I}}$ *kleinste Modell*: $\text{frau}^{\mathcal{I}} =_{\text{def}} \{i\}$, $\text{person}^{\mathcal{I}} =_{\text{def}} \{i\}$, $\text{hat\_kind}^{\mathcal{I}} =_{\text{def}} \{(i, i)\}$. Das Konzept hat unendlich viele weitere Modelle.

Für eine gegebene DL-Interpretation $\mathcal{I}$ definieren wir folgende Begriffe:

Definition 31 (Extension, R -Nachfolger, R -Vorgänger in $\mathcal{I}$) Sei $\mathcal{I}$ eine DL-Interpretation.

Die *Extension* eines Konzeptes C (einer Rolle R) ist die Menge $C^{\mathcal{I}}$ ($R^{\mathcal{I}}$). Wir sagen auch, dass C (R) die Menge $C^{\mathcal{I}}$ ($R^{\mathcal{I}}$) bzgl. $\mathcal{I}$ *denotiert* bzw. *repräsentiert*.

Sei R eine Rolle. Gilt $(i, j) \in R^{\mathcal{I}}$, so nennen wir j einen R -Nachfolger von i bzgl. $\mathcal{I}$, und i einen R -Vorgänger von j bzgl. $\mathcal{I}$. ■

Die semantischen Begriffe Erfüllbarkeit, Äquivalenz und Disjunktheit haben letztlich die gleiche Bedeutung wie in der Prädikatenlogik. Die wichtigste semantische Beziehung zwischen Konzepten, die *Subsumptionsbeziehung*, ist letztlich eine Implikationsbeziehung.

Definition 32 (Eigenschaften von und Relationen zwischen Konzepten) Ein Konzept C heißt *erfüllbar* gdw. es ein Modell $\mathcal{I}$ hat: $C^{\mathcal{I}} \neq \emptyset$, in Zeichen: $\mathcal{I} \models C$.

Ein Konzept D *subsumiert* ein Konzept C gdw. in allen $\mathcal{I}$ $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ gilt, in Zeichen: $C \sqsubseteq D$.¹⁰ C wird das *subsumierte Konzept* (*Subsumee*), und D das *subsumierende Konzept* (*Subsumer*) genannt.

Zwei Konzepte C und D heißen *äquivalent* wenn sowohl $C \sqsubseteq D$ als auch $D \sqsubseteq C$ und somit für alle DL-Interpretationen $\mathcal{I}$ gilt: $C^{\mathcal{I}} = D^{\mathcal{I}}$. Wir schreiben dann: $C \equiv D$. Sind $\{C, D\} \subseteq \mathcal{N}_{\mathcal{C}}$, so werden diese auch *Synonyme* genannt.

Zwei Konzepte C und D heißen *disjunkt* wenn für alle DL-Interpretation $\mathcal{I}$ gilt: $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$, in Zeichen: $C \sqcap D \sqsubseteq \perp$. ■

Die Relation $\sqsubseteq$ definiert eine *Quasiordnung* auf der Menge der Konzepte, denn sie ist reflexiv ($C \sqsubseteq C$) und transitiv (aus $C \sqsubseteq D$ und $D \sqsubseteq E$ folgt $C \sqsubseteq E$). Es handelt sich *nicht* um eine partielle Ordnung, da Konzepte zueinander äquivalent sein können (und daher ist $\sqsubseteq$ nicht antisymmetrisch). Es ist Aufgabe der *Taxonomie-Berechnung*, eine entspr. partielle Ordnung aus der (implizit vorhandenen) Quasiordnung zu errechnen.

2.2.2.3. Semantische Eigenschaften von $\mathcal{ALC}$

$\mathcal{ALC}$ hat zwei sehr wichtige semantische Eigenschaften, die auch für andere DLs wesentlich sind zur Führung von Entscheidbarkeits- und Komplexitätsbeweisen:

1. Die *Endliches-Modell-Eigenschaft* (*Finite Model Property*): Für ein $\mathcal{ALC}$ -Konzept kann stets ein *endliches* Modell angegeben werden, ähnlich wie

¹⁰Prädikatenlogisch gilt dann $C(x) \models D(x)$. Weiter unten wird erläutert, dass Konzepte Formeln aus $\mathcal{L}_{\mathcal{FOPL}}$ mit einer freien Variablen (hier x) sind.

2.2 Beschreibungslogiken

in $\mathcal{L}_{\mathcal{PROT}}$. Diese Eigenschaft gilt für einfache DLs; für ausdrucksstärkere DLs (wie auch für $\mathcal{L}_{\mathcal{FOL}}$) gilt die Eigenschaft nicht.

2. Die *Baummodell-Eigenschaft (Tree Model Property)*: Für ein erfüllbares $\mathcal{ALC}$ -Konzept kann stets ein Modell angegeben werden, welches die Form eines Baums hat.

Für nahezu alle als entscheidbar bekannten DLs gilt eine Art von (generalisierter) Baummodell-Eigenschaft; diese wird als ein Grund für die Entscheidbarkeit dieser Logiken angesehen.

2.2.2.4. Wissensbasen

Eine beschreibungslogische Wissensbasis hat zwei Komponenten: Eine *terminologische Komponente (TBox)* zur *Definition der Terminologie* (bzw. „Ontologie“) einer Domäne, sowie die *assertionale Komponente (ABox)* zur *Repräsentation eines konkreten Weltzustands* [Baader u. a. 2003a, S. 59]. Die TBox wird dementsprechend auch die *intensionale Komponente* und die ABox die *extensionale Komponente* der Wissensbasis genannt:

Definition 33 (Wissensbasis) Eine Wissensbasis $\mathcal{WB}$ ist ein Paar bestehend aus ABox $\mathcal{A}$ und einer TBox $\mathcal{T}$: $\mathcal{WB} = (\mathcal{A}, \mathcal{T})$. ■

Wir definieren nun ABox und TBox.

2.2.2.4.1. Die TBox In der TBox können Konzepte zueinander in Beziehung gesetzt werden. Die folgenden Beziehungen bzw. „epistemologischen Primitive“ können etabliert werden:

- Ein Konzept kann als Spezialisierung eines anderen Konzeptes definiert werden, sodass die beiden in einer Subsumptionsbeziehung stehen.
- Zwei Konzepte können als zueinander äquivalent definiert werden, wobei es sich um eine gegenseitige Spezialisierungs- und somit gegenseitige Subsumptionsbeziehung handelt.

Dementsprechend kann eine $\mathcal{ALC}$ -TBox zwei Arten von TBox-Axiomen enthalten:

Definition 34 (TBox, TBox-Axiom) Seine C, D Konzepte. Eine TBox $\mathcal{T}$ ist eine endliche Menge von TBox-Axiomen der Art

- $C \sqsubseteq D$, die Implikation,
- $C \equiv D$, die Äquivalenz oder Biimplikation. ■

Die Menge der in einer TBox $\mathcal{T}$ verwendeten Konzeptnamen bezeichnen wir mittels $\text{concept\_names}(\mathcal{T})$. ■

Wir lassen hier die allgemeinste Form der TBox-Axiome zu, die sog. *allgemeinen Konzept-Inklusionen* (General Concept Inclusions, GCIs). In TBoxen ausdrucksstärkerer DLs können auch sog. Rollen-Axiome vorkommen (z.B. können Spezialisierungs- bzw. Subsumptionsbeziehungen zwischen Rollen etabliert werden, s.u.).

Mittels eines TBox-Axioms $\text{mutter} \sqsubseteq \text{frau} \sqcap \exists \text{hat\_kind.person}$ wird nun das Konzept *mutter* mit dem Konzept $\text{frau} \sqcap \exists \text{hat\_kind.person}$ in Beziehung gesetzt bzw. als hierzu äquivalent erklärt. Es können nun lediglich solche DL-Interpretationen Modelle der TBox sein, die $\text{mutter}^{\mathcal{I}} = (\text{frau} \sqcap \exists \text{hat\_kind.person})^{\mathcal{I}}$ erfüllen. Die durch die TBox-Axiome erzwungenen Einschränkungen der Interpretationsfunktion werden semantisch daher wie folgt reflektiert:

Definition 35 (Modell einer TBox, Semantische Begriffe für TBoxen) Eine DL-Interpretation $\mathcal{I}$ erfüllt ein TBox-Axiom

- $C \sqsubseteq D$ gdw. $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$,
- $C \sqcap D$ gdw. $C^{\mathcal{I}} = D^{\mathcal{I}}$.

Wir schreiben für ein von $\mathcal{I}$ erfülltes Axiom $t \in \mathcal{T}$ auch $\mathcal{I} \models t$ und sagen, dass $\mathcal{I}$ ein Modell für t ist.

Eine DL-Interpretation $\mathcal{I}$ ist ein Modell einer TBox $\mathcal{T}$, in Zeichen $\mathcal{I} \models \mathcal{T}$, wenn alle Axiome erfüllt sind: $\forall t \in \mathcal{T}. \mathcal{I} \models t$.

Eine TBox, die ein Modell hat, heißt erfüllbar.¹¹

Die *inkohärenten Konzeptnamen* einer TBox $\mathcal{T}$ sind all die Konzeptnamen in $\mathcal{T}$, die äquivalent (synonym) zu $\perp$ sind.

Eine TBox, die inkohärente Konzeptnamen enthält, wird *inkohärent* genannt.¹² ■

Die DL-Interpretationen der Konzepte $\top$ und $\perp$ hatte wir in Def. 26 einfach fixiert; mit Hilfe einer TBox können $\top$ und $\perp$ nun natürlich definiert werden. Ist $C \in \mathcal{N}_C$ ein beliebiger Konzeptname, so können wir annehmen, dass eine TBox stets die beiden Definitionen $\top \sqsubseteq C \sqcup \neg C$ und $\perp \sqsubseteq C \sqcap \neg C$ enthält. Voraussetzung ist, dass die verwendete DL die benötigten Operatoren ($\sqcap, \sqcup$ und $\neg$ zumindest für Konzeptnamen) bietet.

¹¹Ein Beispiel für eine unerfüllbare TBox ist $\{A \sqsubseteq \neg A\}$; offensichtlich muss in einer DL-Interpretation für ein $i \in \Delta^{\mathcal{I}}$ stets $i \in A^{\mathcal{I}}$ oder $i \in (\neg A)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$ gelten, und lt. Definition ist $\Delta^{\mathcal{I}} \neq \emptyset$. Im ersten Fall muss dann auch $i \in (\neg A)^{\mathcal{I}}$, und im zweiten Falle $i \in A^{\mathcal{I}}$ gelten, also $i \in A^{\mathcal{I}} \cap (\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}})$ und daher $i \in \emptyset$.

¹²Kohärenz und Erfüllbarkeit sind nicht synonym! Die TBox $\{A \sqsubseteq B \sqcap \neg B, C \sqsubseteq D\}$ ist erfüllbar, aber inkohärent, da A äquivalent zu $\perp$ ist.

2.2 Beschreibungslogiken

Offensichtlich kann $C \dot{=} D$ durch $\{C \dot{\sqsubseteq} D, D \dot{\sqsubseteq} C\}$ ersetzt, und auf $\dot{=}$ somit verzichtet werden. Alternativ kann jedoch auch $C \dot{\sqsubseteq} D$ gegen $C \dot{=} D \sqcap C'$ ersetzt werden, wobei C' ein neuer bzw. „frischer“ (nirgends sonst in der TBox verwendeter Konzeptname ist); C' schneidet gleichsam den C -Anteil „aus D heraus“. Alternativ kann sogar $\top \dot{=} \neg C \sqcup D$ verwendet werden, doch ist es in vielen Systemen nicht möglich, $\top$ auf der linken Seite eines Axioms zu verwenden (s.u.).

Das Implikations-Axiom $C \dot{\sqsubseteq} D$ sollte nicht mit der Subsumptionsbeziehung $C \sqsubseteq D$ verwechselt werden: Offensichtlich muss in einem Modell einer TBox $\mathcal{T}$ mit $C \dot{\sqsubseteq} D \in \mathcal{T}$ gerade $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ und somit $C \sqsubseteq D$ bzgl. dieser TBox gelten (die Subsumptionsbeziehung ist eine *logische Konsequenz*). Gilt jedoch $C \sqsubseteq D$ für eine TBox, so muss nicht notwendigerweise $C \dot{\sqsubseteq} D \in \mathcal{T}$ gelten. Somit ist also nur eine Teilmenge der geltenden Subsumptionsbeziehungen zwischen Konzeptnamen explizit durch Axiome repräsentiert; die anderen ergeben sich als logische Konsequenzen aus diesen.

Eine TBox schränkt auch die möglichen Modelle eines Konzeptes ein: Das Konzept $mutter \sqcap \neg frau$ ist offensichtlich erfüllbar ohne TBox bzw. mit *leerer TBox*, und unerfüllbar, sobald das Konzepterfüllbarkeitsproblem bzgl. der TBox mit Axiom $mutter \dot{=} frau \sqcap \exists hat\_kind.person$ betrachtet wird. Dementsprechend definieren wir die obigen Inferenzprobleme bzgl. einer TBox:

Definition 36 (Erfüllbarkeit, Subsumption etc. bzgl. einer TBox) Ein Konzept C heißt *erfüllbar* bzgl. einer TBox $\mathcal{T}$ gdw. C ein Modell $\mathcal{I}$ hat das auch Modell von $\mathcal{T}$ ist: $\mathcal{I} \models C$ und $\mathcal{I} \models \mathcal{T}$.

Ein Konzept D *subsumiert* ein Konzept C bzgl. einer TBox $\mathcal{T}$ gdw. jedes Modell $\mathcal{I}$ von C und $\mathcal{T}$ auch Modell von D ist, bzw. gdw. in allen DL-Interpretation $\mathcal{I}$ mit $\mathcal{I} \models \mathcal{T}$ auch $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ gilt. Wir schreiben dann: $C \sqsubseteq_{\mathcal{T}} D$.

Zwei Konzepte C und D sind *äquivalent* bzgl. einer TBox $\mathcal{T}$ gdw. $C \sqsubseteq_{\mathcal{T}} D$ und $D \sqsubseteq_{\mathcal{T}} C$ gilt. Wir schreiben dann: $C \equiv_{\mathcal{T}} D$.

Zwei Konzepte C und D heißen *disjunkt* bzgl. einer TBox $\mathcal{T}$ gdw. $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$ in allen Modellen $\mathcal{I}$ von $\mathcal{T}$ gilt. ■

Bisher haben wir beliebige TBox-Axiome zugelassen. Solche TBoxen können insbesondere auch Zyklen bzgl. der $\dot{\sqsubseteq}$ -Relation enthalten, und es ist nicht sichergestellt, dass durch solche TBoxen etwas „definiert“ wird.

Aus diesem Zweck werden eingeschränkte TBoxen betrachtet, die lediglich bestimmte Arten von Axiomen enthalten und die zudem zusätzliche Bedingungen erfüllen müssen (wie Zyklenfreiheit, s.u.):

Definition 37 (Definition, Primitives Konzept, Definiertes Konzept) Ein TBox-Axiom der Art $CN \dot{=} D$ mit $CN \in \mathcal{N}_c$ heißt *Konzeptdefinition*, und CN heißt *definiertes Konzept*. Eine solche Definition stellt hinreichende und notwendige

Bedingungen für CN bereit.

Ein TBox-Axiom der Art $CN \sqsubseteq D$ mit $CN \in \mathcal{N}_C$ heißt *primitive Konzeptdefinition*, und CN heißt *primitives Konzept*. Eine solche Definition stellt notwendige Bedingungen für CN bereit. ■

Ältere DL-Systeme erlaubten oftmals lediglich sog. *einfache* oder *auffaltbare* TBoxen:

Definition 38 (Einfache TBox) Eine *einfache* (*auffaltbare*) TBox erfüllt zwei Bedingungen:

1. Sie enthält ausschließlich *Konzeptdefinitionen*, und für einen Konzeptnamen CN gibt es höchstens eine Konzeptdefinition $CN \sqsubseteq D$.
2. Sie ist nicht zyklisch:

Sei die Relation $verwendet_{\mathcal{T}}$ der transitive Abschluss der Relation $verwendet\_direkt_{\mathcal{T}}$, wobei $verwendet\_direkt_{\mathcal{T}}(C, D)$ gilt gdw. $C \sqsubseteq E \in \mathcal{T}$ und $subconcept\_of(D, E)$.

Dann ist $\mathcal{T}$ zyklisch gdw. $verwendet_{\mathcal{T}}(CN, CN)$ für ein $CN \in \mathcal{N}_C$ gilt. ■

In diesen älteren Systemen wird die TBox zunächst *normalisiert*, indem primitive Konzeptdefinitionen der Art $C \sqsubseteq D$ gegen $C \sqsubseteq C' \sqcap D$ ersetzt werden. Ist die normalisierte TBox nicht zyklisch, so kann die TBox *aufgefaltet* werden.

Die oben bzgl. einer TBox definierten Inferenzprobleme können dann auf die entspr. Inferenzprobleme *bzgl. der leeren TBox* (bzw. *ohne TBox*) reduziert werden. Hierzu werden die in dem/den Eingabe-Konzept/en verwendeten definierten Konzepte in einer Schleife solange durch ihre Definitionen aus der TBox substituiert, bis nur noch undefinierte Konzeptnamen in dem/den Eingabe-Konzept/en übrig bleiben. Da die TBox azyklisch ist, terminiert der Expansionsprozess. Die ersten DL-Systeme verwendeten diese Technik zur Berücksichtigung der TBox. Dies kann als eine Art „Makroexpansion“ verstanden werden.

Auffaltbare TBoxen haben die wichtige Eigenschaft, *definitiv* zu sein [Baader u. a. 2003a, Kap. 2.2.2]: Sei $\mathcal{T}$ eine TBox, die lediglich die 1. Bedingung aus Def. 38 erfüllt. Gegeben eine solche TBox, so partitionieren wir die Menge $concept\_names(\mathcal{T})$ in die Menge der *definierten Namen* und die Menge der *primitiven Namen*. Ein primitiver Name ist ein Konzeptname, der ausschließlich auf der rechten Seite eines Axioms auftaucht; es gibt also kein Axiom mit diesem Namen auf der linken Seite.

Sei nun eine *Basis-DL-Interpretation* dieser TBox eine DL-Interpretation, die lediglich für die primitiven Namen definiert ist (also ist $\cdot^{\mathcal{I}}$ undefiniert für die definierten Namen). Eine *Erweiterung* dieser Basis-DL-Interpretation erweitert

2.2 Beschreibungslogiken

die Definition von $\cdot^{\mathcal{I}}$ nun so, dass auch die definierten Namen interpretiert werden. Kann nun zu jeder Basis-DL-Interpretation *genau eine* Erweiterung angegeben werden, die ein Modell der TBox ist, so wird die TBox *definitorisch* genannt.

Intuitiv gesprochen bedeutet dies also, dass die Interpretationen der definierten Symbole eindeutig anhand der Interpretationen der primitiven Symbole bestimmt werden.

Wir hatten gesagt, dass die betrachtete TBox die 1. Bedingung aus Def. 38 erfüllen muss. Es gibt jedoch auch *zyklische TBoxen*, die definitorisch sind (die Namen, die in einem Zyklus auftauchen, sind dann nicht mehr primitiv). Es gilt jedoch folgendes wichtiges Theorem: Jede definitorische $\mathcal{ALC}$ -TBox ist äquivalent zu einer *einfachen* TBox [Baader u. a. 2003a, Kap. 2].

Im Falle einer nichtdefinitorischen TBox kann es mehr als eine Erweiterung geben, die Modell der TBox ist. Es wurden drei Arten von Semantiken untersucht [Nebel 1991, Baader 1990]: Die sog. *deskriptive Semantik* erlaubt *jede* Erweiterung als Modell, die die Axiome aus Def. 35 erfüllt. Hingegen fordert die *größter-Fixpunkt-Semantik*, dass die Erweiterung ein größter Fixpunkt der durch die TBox definierten Gleichungen (s. Def. 35) sein muss. Alternativ hierzu fordert die *kleinster-Fixpunkt-Semantik*, dass es sich eben um einen kleinsten Fixpunkt handeln muss.

Heutige DL-Systeme lassen allgemeine TBoxen und somit zyklische TBoxen mit GCIs zu. Die deskriptive Semantik hat sich als Standard etabliert. Allerdings erfordert die Behandlung allgemeiner TBoxen komplexere algorithmische Verfahren.

Eine TBox mit deskriptiver Semantik kann in *ausdrucksstarken DLs* (s.u.) in ein erfüllbarkeitsäquivalentes Konzept übersetzt werden (in $\mathcal{ALC}$ ist dies noch nicht der Fall). Die verwendete Technik heißt *TBox-Internalisierung* [Baader 1991]. Eine DL, die TBox-Internalisierung ermöglicht, muss zwei Bedingungen erfüllen:

1. Die Konzept-Modelle müssen *verbunden* sein (*Connected Model Property*), d.h., für ein erfüllbares Konzept C gibt es stets ein Modell $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}) \models C$ mit $i \in C^{\mathcal{I}}$, sodass von i aus stets alle anderen $j \in \Delta^{\mathcal{I}}$, $i \neq j$ „erreichbar“ sind. Erreichbar bedeutet hier, dass es einen beliebigen Pfad von i nach j gibt: $(i, j) \in R_1^{\mathcal{I}} \circ \dots \circ R_n^{\mathcal{I}}$ gibt (wobei $R_1, \dots, R_n \in \mathcal{N}_{\mathcal{R}}$ beliebige Rollen sind).
2. Es muss möglich sein, von i aus mittels eines Konzeptes etwas über alle anderen $j \in \Delta^{\mathcal{I}}$ auszusagen.

Dann kann die TBox $\mathcal{T}$ (wir nehmen o.B.d.A. an, dass diese nur Axiome der Art $C \sqsubseteq D$ enthält) zunächst in eine Konjunktion

$$C_{\mathcal{T}} =_{\text{def}} \bigwedge_{C \sqsubseteq D \in \mathcal{T}} \neg C \sqcup D$$

übersetzt werden, und mittels 2. kann dieses Konzept für alle Individuen in $\Delta^{\mathcal{I}}$ etabliert werden.

Für die Sprache $\mathcal{ALCH}_{\mathcal{R}^+}$ (s.u.) würde 2. mittels einer transitiven Oberrolle aller Rollen namens $R_T \in \mathcal{N}_{\mathcal{R}^+}$ (d.h., $\mathcal{N}_{\mathcal{R}^+} \subseteq \mathcal{N}_{\mathcal{R}}$, $R_i \sqsubseteq R_T$ für alle $R_i \in \mathcal{N}_{\mathcal{R}}$) und dem Konzept

$$I_T =_{def} C_T \sqcap \forall R_T.C_T$$

ermöglicht werden.

Nun gilt: C ist bzgl. $\mathcal{T}$ erfüllbar gdw. $C \sqcap I_T$ (bzgl. der leeren TBox) erfüllbar ist.

2.2.2.4.2. Die ABox Während die TBox zur (intensionalen) Definition des Vokabulars (der Terminologie) der Ontologie verwendet wird, wird die sog. *assertionale Box (ABox)* zur Repräsentation eines konkreten Zustands der Welt in Form von Individuen und ihrer wechselseitigen Beziehungen verwendet. Die ABox ist daher die extensionale Komponente einer Wissensbasis. ABoxen werden in dieser Arbeit eine wichtige Rolle spielen.

In der ABox gibt es zwei wesentliche epistemologische Primitive, die mittels sog. *ABox-Assertionen* erzwungen werden können:

- Mitgliedschaft eines Individuums in der durch ein Konzept denotierten Menge.
- Mitgliedschaft eines Paares von Individuen in der durch eine Rolle denotierten binären Relation.

Dementsprechend kann eine ABox zwei Arten von Axiomen enthalten:

Definition 39 (ABox, ABox-Assertion) Sei $\mathcal{N}_{\mathcal{I}}$ eine Menge von sog. *ABox-Individuen* mit $i, j \in \mathcal{N}_{\mathcal{I}}$, sei C ein Konzept, und R eine Rolle.

Eine ABox $\mathcal{A}$ ist eine endliche Menge von ABox-Assertionen (auch: ABox-Axiomen, ABox-Zusicherungen, ABox-Aussagen, Fakten) der Art

- $i : C$, die Instanz- oder Konzept-Assertion,
- $(i, j) : R$, die Rollen-Assertion.

Die Menge der in einer ABox $\mathcal{A}$ verwendeten Individuen bezeichnen wir mit $\text{inds}(\mathcal{A})$.

Die Menge der in einer ABox $\mathcal{A}$ (in Konzept-Assertionen) verwendeten Konzepte bezeichnen wir mittels $\text{concepts}(\mathcal{A})$, und die (in Rollen-Assertionen) verwendeten Rollen mittels $\text{roles}(\mathcal{A})$. ■

Bezogen auf unsere Familien-Domäne können wir mit einer ABox nun folgenden Welt-Zustand modellieren:

2.2 Beschreibungslogiken

$\{michael : person, margrit : frau, (margrit, michael) : hat\_kind\}.$

Die formale Semantik einer ABox wird sogleich angegeben. Die Semantik der Beispiel-ABox sollte jedoch offensichtlich sein.

Weitere wichtige ABox-Begriffe sind:

Definition 40 (*R*-Füller, *R*-Nachfolger, *R*-Vorgänger in einer ABox) Sei $\mathcal{A}$ eine ABox und R eine Rolle.

Gilt $(i, j) : R \in \mathcal{A}$, so wird j ein *R*-Füller oder *R*-Nachfolger von i , und i ein *R*-Vorgänger von j genannt. Ist R beliebig, so nennen wir j einfach einen Füller bzw. Nachfolger, und i einen Vorgänger von j . ■

Aus dem Kontext wird im Folgenden stets klar werden, ob der Begriff Vorgänger bzw. Nachfolger Def. 31 oder Def. 40 meint.

Durch die Assertionen einer ABox werden weitere Bedingungen an die Interpretationsfunktion formuliert. Intuitiv sollen durch die Interpretationsfunktion $\cdot^{\mathcal{I}}$ ABox-Individuen auf Domänen-Individuen (Elemente aus $\Delta^{\mathcal{I}}$) abgebildet werden, und zwar so, dass die ABox-Assertionen *erfüllt* sind:

Definition 41 (Modell einer ABox, Semantische Begriffe für ABoxen) Eine DL-Interpretation für eine ABox $\mathcal{A}$ ist eine DL-Interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, deren Interpretationsfunktion $\cdot^{\mathcal{I}} : \mathcal{N}_{\mathcal{I}}$ auf Elemente in $\Delta^{\mathcal{I}}$ abbildet.¹³

Wird zusätzlich gefordert, dass die Abbildung von ABox-Individuen auf Elemente in $\Delta^{\mathcal{I}}$ *injektiv* ist, so sagen wir, dass *Unique Name Assumption (UNA)* gilt. Unter der UNA folgt also aus $a, b \in \text{inds}(\mathcal{A})$ mit $a \neq b$ stets $a^{\mathcal{I}} \neq b^{\mathcal{I}}$.

Eine DL-Interpretation für eine ABox *erfüllt* eine ABox-Assertion

- $i : C$ gdw. $i^{\mathcal{I}} \in C^{\mathcal{I}}$, und
- $(i, j) : R$ gdw. $(i^{\mathcal{I}}, j^{\mathcal{I}}) \in R^{\mathcal{I}}$.

Wir schreiben für eine von $\mathcal{I}$ erfüllte Assertion $a \in \mathcal{A}$ auch $\mathcal{I} \models a$ und sagen, dass $\mathcal{I}$ ein Modell für a ist.

Eine DL-Interpretation $\mathcal{I}$ ist ein Modell einer ABox $\mathcal{A}$, in Zeichen $\mathcal{I} \models \mathcal{A}$, wenn alle Assertionen erfüllt sind: $\forall a \in \mathcal{A}. \mathcal{I} \models a$.

Eine ABox, die ein Modell hat, heißt erfüllbar. ■

Die Modelle einer $\mathcal{ALC}$ -ABox haben offensichtlich nicht mehr notwendigerweise die Form eines Baums. Der nichtbaumartige Teil der Modelle ist jedoch endlich, da die ABox endlich ist.

Anhand der intuitiven Semantik der verwendeten Begriffe sollte die ABox

¹³Natürlich müssen hier nur die Elemente aus $\text{inds}(\mathcal{A})$ interpretiert werden.

$\{michael : person, margrit : frau, margrit : \neg mütter, (margrit, michael) : hat\_kind\}$

unerfüllbar sein. Hierzu muss jedoch die TBox bzw. die Konzeptdefinition $mutter \hat{=} frau \sqcap \exists hat\_kind.person$ berücksichtigt werden. Eine ABox wird daher i.d.R. eine assoziierte TBox haben (die natürlich auch leer sein kann):

Definition 42 (Assoziierte TBox einer ABox) Sei $\mathcal{A}$ eine ABox. Wir bezeichnen die assoziierte TBox von $\mathcal{A}$ mit $\mathcal{T}_{\mathcal{A}}$. ■

In der Regel werden wir daher nicht nur Erfüllbarkeit von $\mathcal{A}$ fordern, sondern Erfüllbarkeit der Wissensbasis $(\mathcal{A}, \mathcal{T}_{\mathcal{A}})$. Hier ist also eine DL-Interpretation $\mathcal{I}$ gesucht, die sowohl $\mathcal{I} \models \mathcal{A}$ als auch $\mathcal{I} \models \mathcal{T}_{\mathcal{A}}$ erfüllt. Prinzipiell kann die Erfüllbarkeit einer ABox $\mathcal{A}$ auch bzgl. einer anderen TBox als $\mathcal{T}_{\mathcal{A}}$ getestet werden (dies ist i.d.R. nur dann sinnvoll, wenn das in der ABox verwendete Vokabular auch in dieser anderen TBox definiert ist).

2.2.2.5. Standard-Inferenzprobleme und -Inferenzdienste eines DL-Systems

Die Standard-Inferenzleistungen bzw. -Dienste eines DL-Systems können formal spezifiziert werden: Es handelt sich um Implementierungen von Algorithmen zur Lösung der sog. *Standard-Inferenzprobleme*. Oftmals handelt es sich bei diesen um Entscheidungsprobleme. Wir hatten bereits einige zentrale Entscheidungsprobleme für Konzepte, TBoxen und ABoxen formal definiert (Erfüllbarkeit, Subsumption, etc.). Wir unterscheiden im Folgenden *Probleme* von *Diensten*; Dienste sind Leistungen eines DL-Systems, Probleme werden hingegen mathematisch definiert. Wir wollen nun die noch fehlenden Standard-Inferenzprobleme und entspr. Dienste nachreichen und entspr. Operations- bzw. Funktionsnamen für diese vereinbaren.

Die *intensionalen Inferenzen* betreffen die TBox und Konzepte, während die *extensionalen Inferenzen* auch die ABox involvieren. All diese Inferenzprobleme sind für $\mathcal{ALC}$ entscheidbar (und auch für die weiter unten vorgestellten erweiterten DLs).

Wir definieren zunächst die *Entscheidungsprobleme*. Die Antwort/Lösung eines Entscheidungsproblems ist immer TRUE oder FALSE. *Höherwertige* Inferenzprobleme (s.u.) werden auf diese Entscheidungsprobleme reduziert werden.

Die *intensionalen Standard-Inferenzprobleme* beziehen sich auf Konzepte und TBoxen und wurden bereits definiert. Aus Gründen der Vollständigkeit wiederholen wir diese Definitionen und geben dabei die vereinbarten Operationsnamen der entspr. Dienste an. Das TBox-Argument ist optional bzw. die TBox kann auch leer sein:

- Das Konzept-Erfüllbarkeitsproblem (bzgl. einer TBox):

2.2 Beschreibungslogiken

Gegeben ein Konzept C und eine TBox $\mathcal{T}$ – gibt es eine DL-Interpretation $\mathcal{I}$, sodass $\mathcal{I} \models \mathcal{T}$ und $\mathcal{I} \models C$ gilt?

Wir sehen eine entspr. Funktion $\text{concept\_satisfiable?}(C, \mathcal{T})$ vor.¹⁴

- Das *Konzept-Subsumptionsproblem* (bzgl. einer TBox):

Gegeben zwei Konzepte C, D und eine TBox $\mathcal{T}$ – gilt für alle DL-Interpretationen mit $\mathcal{I} \models \mathcal{T}$ stets $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$? Wir schreiben dann $C \sqsubseteq_{\mathcal{T}} D$.

Wir sehen eine entspr. Funktion $\text{concept\_subsumes?}(D, C, \mathcal{T})$ vor.

- Das *Konzept-Äquivalenzproblem* (bzgl. einer TBox):

Gegeben zwei Konzepte C, D und eine TBox $\mathcal{T}$ – gilt für alle DL-Interpretationen mit $\mathcal{I} \models \mathcal{T}$ stets $C^{\mathcal{I}} = D^{\mathcal{I}}$? Wir schreiben dann $C \equiv_{\mathcal{T}} D$.

Wir sehen eine entspr. Funktion $\text{concepts\_equivalent?}(C, D, \mathcal{T})$ vor.

- Das *Konzept-Disjunktheitsproblem* (bzgl. einer TBox):

Gegeben zwei Konzepte C, D und eine TBox $\mathcal{T}$ – gilt für alle DL-Interpretationen mit $\mathcal{I} \models \mathcal{T}$ stets $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$? Wir schreiben dann $C \sqcap D \sqsubseteq_{\mathcal{T}} \perp$.

Wir sehen eine entspr. Funktion $\text{concepts\_disjoint?}(C, D, \mathcal{T})$ vor.

- Das *TBox-Erfüllbarkeitsproblem*: Hat die TBox $\mathcal{T}$ ein Modell, gibt es also eine DL-Interpretation $\mathcal{I}$, sodass $\mathcal{I} \models \mathcal{T}$?

Wir sehen eine entspr. Funktion $\text{tbox\_satisfiable?}(\mathcal{T})$ vor.

Die *extensionalen Standard-Inferenzprobleme* setzen eine ABox voraus. Da eine ABox $\mathcal{A}$ i.d.R. über eine assoziierte TBox $\mathcal{T}_{\mathcal{A}}$ verfügt, sind diese Probleme bzgl. der Wissensbasis $(\mathcal{A}, \mathcal{T}_{\mathcal{A}})$ definiert. Für die folgenden Funktionen vereinbaren wir, dass $\mathcal{A}$ und $\mathcal{T}$ optionale Argumente sind. Wird $\mathcal{A}$ nicht angegeben, so wird die *aktuelle ABox* verwendet, bzw. die ABox ergibt sich aus dem Kontext. Wird $\mathcal{T}$ nicht explizit angegeben, so ist stets $\mathcal{T} = \mathcal{T}_{\mathcal{A}}$ gemeint (auch die leere TBox kann als Argument übergeben werden):

- Das *ABox- bzw. Wissensbasis-Erfüllbarkeitsproblem*:

Hat die ABox $\mathcal{A}$ bzw. die Wissensbasis $(\mathcal{A}, \mathcal{T})$ ein Modell, gibt es also eine DL-Interpretation $\mathcal{I}$, sodass $\mathcal{I} \models \mathcal{A}$ und $\mathcal{I} \models \mathcal{T}$ gilt?

In diesem Fall heißt $\mathcal{I}$ Modell der ABox bzw. der Wissensbasis.

Wir sehen eine entspr. Funktion $\text{abox\_consistent?}(\mathcal{A}, \mathcal{T})$ vor.

Für die folgenden Funktionen bzw. extensionalen Standard-Inferenzen setzen wir voraus, dass die Eingabe-ABox $\mathcal{A}$ bzw. Eingabe-Wissensbasis $(\mathcal{A}, \mathcal{T})$ erfüllbar ist:

¹⁴ Prädikate bzw. Funktionen, die stets TRUE oder FALSE zurückgeben, enden im Folgenden mit „?“.

- Das *Instanzen-Prüfungsproblem*:

Gegeben eine ABox $\mathcal{A}$ bzw. Wissensbasis $(\mathcal{A}, \mathcal{T})$, ein Individuum $i \in \text{inds}(\mathcal{A})$, und ein Konzept C – gilt dann $i^{\mathcal{I}} \in C^{\mathcal{I}}$ in allen Modellen von $\mathcal{A}$ bzw. $(\mathcal{A}, \mathcal{T})$? In diesem Falle wird i eine *Instanz* von C oder auch *C-Instanz* genannt. Die Konzept-Assertion ist dann eine logische Konsequenz aus der Wissensbasis, in Zeichen: $(\mathcal{A}, \mathcal{T}) \models i : C$.

Wir sehen eine entspr. Funktion $\text{individual\_instance?}(i, C, \mathcal{A}, \mathcal{T})$ vor.

- Das *Rollen-Prüfungsproblem*:

Gegeben eine ABox $\mathcal{A}$ bzw. Wissensbasis $(\mathcal{A}, \mathcal{T})$, zwei Individuen $i, j \in \text{inds}(\mathcal{A})$, und eine Rolle R – gilt dann $(i^{\mathcal{I}}, j^{\mathcal{I}}) \in R^{\mathcal{I}}$ in allen Modell von $\mathcal{A}$ bzw. $(\mathcal{A}, \mathcal{T})$? Die Rollen-Assertion ist dann eine logische Konsequenz aus der Wissensbasis, in Zeichen: $(\mathcal{A}, \mathcal{T}) \models (i, j) : R$.

Wir sehen eine entspr. Funktion $\text{individuals\_related?}(i, j, R, \mathcal{A}, \mathcal{T})$ vor.

Unter einem *höherwertigen Inferenzproblem* verstehen wir im Folgenden ein Problem, dessen Lösung bzw. Ergebnis nicht in der Berechnung einer Antwort TRUE oder FALSE besteht. Stattdessen werden hier komplexere Ergebnisse gefordert; es handelt sich daher nicht mehr um reine Entscheidungsprobleme.¹⁵

Ein wichtiges *höherwertiges intensionales Inferenzproblem* ist

- Das *Taxonomie-Berechnungsproblem*:

Die *Taxonomie* einer TBox $\mathcal{T}$ ist ein GAG $(\text{concept\_names}(\mathcal{T}), \sqsubseteq'_{\mathcal{T}})$, wobei $CN \sqsubseteq'_{\mathcal{T}} DN$ gdw.

1. $CN \sqsubseteq_{\mathcal{T}} DN$ mit $\{CN, DN\} \subseteq \text{concept\_names}(\mathcal{T})$ gilt, und
2. wenn aus $EN \in \text{concept\_names}(\mathcal{T})$ mit $CN \sqsubseteq_{\mathcal{T}} EN$ und $EN \sqsubseteq_{\mathcal{T}} DN$ stets entweder $CN \equiv_{\mathcal{T}} EN$ oder $DN \equiv_{\mathcal{T}} EN$ folgt (m.a.W., dann sind entweder EN und CN oder EN und DN synonym).

Die Taxonomie ist daher gerade das Hasse-Diagramm¹⁶ der partiellen Ordnung $\sqsubseteq'_{\mathcal{T}}$, die aus der Quasiordnung $\sqsubseteq_{\mathcal{T}}$ durch Äquivalenzklassen-Faktorisierung entsteht. Die Äquivalenzklassen sind gerade die in $\mathcal{T}$ *zueinander synonymen Konzeptnamen*.

Eine Taxonomie wird durch *Klassifikation der TBox* erzeugt. Dies ist das Taxonomie-Berechnungsproblem. Das Klassifizieren geschieht inkrementell: Beginnend mit einem GAG, der lediglich Knoten für $\top$ und $\perp$ enthält, wird der GAG inkrementell um weitere Knoten (die Klassen von synonymen Konzeptnamen darstellen) erweitert. Das Einsortieren eines neuen

¹⁵Natürlich kann letztlich *jedes Problem* als reines Entscheidungsprobleme definiert werden, doch ist diese Betrachtung hier nicht hilfreich.

¹⁶Das Hasse-Diagramm einer partiellen Ordnung $(M, \leq)$ ist ein GAG (V, E) , der wie folgt gebildet wird (für $a \leq b$, $a \neq b$ schreiben wir $a < b$): $V =_{\text{def}} M, E =_{\text{def}} \{(a, b) \mid a < b, \neg \exists c. a < c, c < b\}$.

2.2 Beschreibungslogiken

Knotens an die „richtige Position“ im GAG wird das *Klassifizieren des Konzeptes* genannt.

Eine TBox, deren Taxonomie berechnet wurde, wird als *klassifiziert* bezeichnet.

Die Taxonomie einer *inkohärenten TBox* wird also Synonyme für $\perp$ enthalten.

Wir sehen entspr. Funktionen wie $\text{taxonomy}(T)$, $\text{classify\_tbox}(T)$, $\text{tbox\_coherent?}(T)$, $\text{tbox\_classified?}(T)$, $\text{incoherent\_concept\_names}(T)$ etc. vor.

In dieser Arbeit sind die wichtigsten höherwertigen *extensionalen* Inferenzprobleme die sog. *ABox-Basis-Retrieval-Probleme*. Diese setzen eine konsistente ABox $\mathcal{A}$ bzw. Wissensbasis $(\mathcal{A}, T)$ voraus. Bzgl. der $\mathcal{A}$ - und T -Argumente gilt das bereits oben Vereinbarte.

- *Das Instanzen-Retrieval-Problem:*

Gegeben eine ABox $\mathcal{A}$ bzw. Wissensbasis $(\mathcal{A}, T)$, und ein Konzept C , bestimme alle $i \in \text{inds}(\mathcal{A})$, sodass $i^{\mathcal{I}} \in C^{\mathcal{I}}$ in allen Modellen $\mathcal{I}$ von $\mathcal{A}$ und T gilt:

$$\text{concept\_instances}(C, \mathcal{A}, T) =_{\text{def}} \{ i \mid i \in \text{inds}(\mathcal{A}), (\mathcal{A}, T) \models i : C \}.$$

Offensichtlich kann $\text{concept\_instances}(C, \mathcal{A}, T)$ durch sukzessive Entscheidung des *Instanzen-Prüfungsproblems* für jedes $i \in \text{inds}(\mathcal{A})$ errechnet werden.

- *Das Rollenfüller-Retrieval-Problem:*

Gegeben eine ABox $\mathcal{A}$ bzw. Wissensbasis $(\mathcal{A}, T)$, ein Individuum i und eine Rolle R , bestimme alle $j \in \text{inds}(\mathcal{A})$, sodass $(i^{\mathcal{I}}, j^{\mathcal{I}}) \in R^{\mathcal{I}}$ in allen Modellen $\mathcal{I}$ von $\mathcal{A}$ und T gilt:

$$\text{role\_fillers}(i, R, \mathcal{A}, T) =_{\text{def}} \{ j \mid j \in \text{inds}(\mathcal{A}), (\mathcal{A}, T) \models (i, j) : R \}.$$

Offensichtlich kann $\text{role\_fillers}(i, R, \mathcal{A}, T)$ durch sukzessive Entscheidung des *Rollen-Prüfungsproblems* für jedes Paar $i, j \in \text{inds}(\mathcal{A})$ von Individuen errechnet werden.

Analog sind definiert:

$$\text{individual\_role\_successors}(i, R, \mathcal{A}, T) =_{\text{def}} \text{role\_fillers}(i, R, \mathcal{A}, T) \\ \text{und}$$

$$\text{individual\_role\_predecessors}(i, R, \mathcal{A}, T) =_{\text{def}} \{ j \mid j \in \text{inds}(\mathcal{A}), (\mathcal{A}, T) \models (j, i) : R \}.$$

- *Das Rollenpaare-Retrieval-Problem:*

Gegeben eine ABox $\mathcal{A}$ bzw. Wissensbasis $(\mathcal{A}, T)$, und eine Rolle R , bestimme alle $i, j \in \text{inds}(\mathcal{A})$, sodass $(i^{\mathcal{I}}, j^{\mathcal{I}}) \in R^{\mathcal{I}}$ in allen Modellen $\mathcal{I}$ von $\mathcal{A}$ und T gilt:

$$\text{role\_pairs}(R, \mathcal{A}, T) = \{ (i, j) \mid i, j \in \text{inds}(\mathcal{A}), (\mathcal{A}, T) \models (i, j) : R \}.$$

Wiederum könnte das Rollen-Prüfungsproblem zur Realisierung verwendet werden.

Weitere (in der Arbeit jedoch nicht so relevante) extensionale Standard-Inferenzprobleme sind:

- **Das Instanz-Realisierungs-Problem:**

Gegeben eine ABox $\mathcal{A}$ bzw. Wissensbasis $(\mathcal{A}, \mathcal{T})$ und ein Individuum $i \in \text{inds}(\mathcal{A})$, berechne die Menge der speziellsten Konzeptnamen $CN \in \text{concept\_names}(\mathcal{T})$, von denen i Instanz ist:

$$\begin{aligned} \text{direct\_types}(i, \mathcal{A}, \mathcal{T}) = \{ & CN \in \text{concept\_names}(\mathcal{T}) \mid \\ & (\mathcal{A}, \mathcal{T}) \models i : CN, \\ & \forall DN \in \text{concept\_names}(\mathcal{T}). \\ & DN \sqsubseteq_{\mathcal{T}} CN \wedge (\mathcal{T}, \mathcal{A}) \models i : DN \rightarrow \\ & CN \equiv_{\mathcal{T}} DN \}. \end{aligned}$$

Der Prozess der Berechnung von $\text{direct\_types}(i, \mathcal{A}, \mathcal{T})$ wird *Realisierung* von i genannt.

- **Das ABox-Realisierungs-Problem:**

Gegeben eine ABox $\mathcal{A}$ bzw. Wissensbasis $(\mathcal{A}, \mathcal{T})$, realisiere jedes Individuum $i \in \text{inds}(\mathcal{A})$. Die resultierende ABox wird dann als *realisiert* bezeichnet. Eine realisierte ABox wird primär zur Beschleunigung des Instanzenretrievals (also als Index) dienen. Die gefolgerten Konzept-Assertionen können auch syntaktisch zur ABox hinzugefügt werden (wodurch sie explizit würden).

2.2.2.5.1. Problemreduktionen Viele der Inferenzprobleme können wechselseitig aufeinander reduziert werden:

- Das Subsumptionsproblem bzgl. $\mathcal{T}$ kann auf das Konzept-Erfüllbarkeitsproblem bzgl. $\mathcal{T}$ reduziert werden, denn offensichtlich gilt $C \sqsubseteq_{\mathcal{T}} D$ gdw. $C \sqcap \neg D$ unerfüllbar in $\mathcal{T}$ ist. Dies ist i.Allg. jedoch nur dann möglich, wenn die DL volle Negation bietet.
- Umgekehrt kann auch das Konzept-Erfüllbarkeitsproblem bzgl. $\mathcal{T}$ auf das Subsumptionsproblem bzgl. $\mathcal{T}$ reduziert werden, da C bzgl. $\mathcal{T}$ unerfüllbar ist gdw. $C \sqsubseteq_{\mathcal{T}} \perp$ gilt.

Trivialerweise gilt daher (wir sparen uns die Referenz auf bzw. bzgl. $\mathcal{T}$):

- C und D sind disjunkt gdw. $C \sqcap D$ unerfüllbar ist (wir hatten bereits zuvor $C \sqcap D \sqsubseteq_{\mathcal{T}} \perp$ verwendet).
- C und D sind äquivalent gdw. $C \sqsubseteq D$ und $D \sqsubseteq C$ gilt.

Offensichtlich sind noch viele andere Reduktionen möglich:

2.2 Beschreibungslogiken

- Das Konzept C ist unerfüllbar gdw. die ABox $\{i : C\}$ unerfüllbar ist.

Somit können alle intensionalen Inferenzprobleme durch Reduktion auf das Konzepterfüllbarkeitsproblem und dann durch weitere Reduktion auf das ABox- bzw. Wissensbasis-Erfüllbarkeitsproblem entschieden werden.

Die *extensionalen Inferenzprobleme* können ebenfalls auf das ABox- bzw. Wissensbasis-Erfüllbarkeitsproblem reduziert werden:

- Das Instanzen-Prüfungsproblem für $\mathcal{A}$, C und i kann auf das ABox-Unerfüllbarkeitsproblem der ABox $\mathcal{A}' =_{def} \mathcal{A} \cup \{i : \neg C\}$ reduziert werden (i ist eine Instanz von C in $\mathcal{A}$ gdw. $\mathcal{A}'$ unerfüllbar ist).
- Das Rollen-Prüfungsproblem für $\mathcal{A}$, R und i, j kann auf das ABox-Unerfüllbarkeitsproblem der ABox $\mathcal{A}' =_{def} \mathcal{A} \cup \{(i, j) : R, i : \forall R.M_R, j : \neg M_R\}$ reduziert werden, wobei $M_R \notin \text{concepts}(\mathcal{A})$.
- Wir hatten oben bereits erwähnt, dass die Retrieval-Probleme auf entsprechende Prüfungsprobleme reduziert werden können.

2.2.2.5.2. Standard-Inferenzdienste und das API eines DL-Systems Die *Standard-Inferenzprobleme* sind als *Standard-Inferenzdienste* eines DL-Systems über ein API (*Application Program Interface*) verfügbar. Sie sind Teil der „Ask“-Schnittstelle des Systems. Im API finden sich Funktionen/Prozeduren/Methoden, die den mathematischen Funktionen entsprechen. Beispielsweise entspricht der mathematischen `concept_instances`-Funktion die API-Funktion `concept-instances` (wir verwenden also einen anderen Zeichensatz).¹⁷

2.2.2.6. Prädikatenlogische Deutung

Viele DLs können auf $\mathcal{L}_{\mathcal{FOPL}}$ abgebildet werden [Borgida 1992]. Für $\mathcal{ALC}$ ist zudem das sog. Zwei-Variablen-Fragment von $\mathcal{L}_{\mathcal{FOPL}}$, auch $\mathcal{FO}_2$ genannt, ausreichend. In $\mathcal{FO}_2$ ist die Menge der Variablennamen auf zwei Variablen eingeschränkt, z.B. $\mathcal{V} =_{def} \{x, y\}$.

Die sog. *Standard-Übersetzung* ist aus der Modallogik bekannt und auch für DLs bzw. $\mathcal{ALC}$ anwendbar [Blackburn u. a. 2001, Kap. 2.4]. Die Beziehungen zur Modallogik werden in Kap. 2.2.6 beleuchtet. Zunächst werden mittels einer Übersetzungsfunktion ϕ_x $\mathcal{ALC}$ -Konzepte auf prädikatenlogische Formeln in $\mathcal{FO}_2$ wie folgt abgebildet:

¹⁷Mathematische Prädikate werden mit ?-Suffix gekennzeichnet, die entspr. implementierten Funktionen/Methoden hingegen mit -p-Suffix.

$$\begin{aligned}
 \phi_x(CN) &=_{def} CN(x), \text{ für } CN \in \mathcal{N}_C \\
 \phi_x(\neg C) &=_{def} \neg \phi_x(C) \\
 \phi_x(C_1 \sqcap C_2) &=_{def} \phi_x(C_1) \wedge \phi_x(C_2) \\
 \phi_x(C_1 \sqcup C_2) &=_{def} \phi_x(C_1) \vee \phi_x(C_2) \\
 \phi_x(\exists R.D) &=_{def} \exists y. R(x, y) \wedge \phi_y(D) \\
 \phi_x(\forall R.D) &=_{def} \forall y. R(x, y) \rightarrow \phi_y(D)
 \end{aligned}$$

Die Funktion ϕ_y ist analog definiert; doch werden die beiden Variablen x und y vertauscht. Daher werden auf den rechten Seite der Definitionen auch Vorkommen von ϕ_x gegen ϕ_y und andersrum ersetzt.

Mit diesen Definitionen ist z.B.

$$\phi_x(frau \sqcap \exists hat\_kind.person) = frau(x) \wedge \exists y. hat\_kind(x, y) \wedge person(y).$$

Anhand der vorangegangenen Semantikdefinition ist auch klar, wie TBoxen und ABoxen zu übersetzen sind:

$$\begin{aligned}
 \phi(\mathcal{T}) &=_{def} \bigwedge_{t \in \mathcal{T}} \phi(t) \\
 \phi(\mathcal{A}) &=_{def} \bigwedge_{a \in \mathcal{A}} \phi(a) \\
 \phi(i : C) &=_{def} \phi_x(C)_{[x \leftarrow i]} \\
 \phi((i, j) : R) &=_{def} R(i, j) \\
 \phi(C \sqsubseteq D) &=_{def} \forall x. \phi_x(C) \rightarrow \phi_x(D) \\
 \phi(C \equiv D) &=_{def} \forall x. \phi_x(C) \leftrightarrow \phi_x(D)
 \end{aligned}$$

Es gibt jedoch auch ausdrucksstärkere DLs als $\mathcal{ALC}$, die Operatoren bieten, die nicht durch $\mathcal{L}_{\mathcal{FOPL}}$ axiomatisiert werden können. Hierzu gehört z.B. der Operator $\cdot^+$ zur Bildung des transitiven Abschlusses einer Rolle. Hingegen können *transitiv abgeschlossene Rollen* in $\mathcal{L}_{\mathcal{FOPL}}$ axiomatisiert werden, s. Kap. 2.2.3.3.

Will man das Zwei-Variablen-Fragment nicht verlassen, so werden zur prädikatenlogischen Abbildung von sog. Anzahlrestriktionen *zählenden Quantoren* benötigt, s. Kap. 2.2.3.2. Diese $\mathcal{FO}_2$ -Erweiterung heißt dann $\mathcal{C}_2$. Auch $\mathcal{C}_2$ ist als entscheidbar bekannt.

Die intensionalen und extensionalen Standard-Inferenzprobleme lassen sich prädikatenlogisch entweder als Erfüllbarkeits- oder Folgerbarkeitsprobleme verstehen:

- Das Konzept C ist erfüllbar in $\mathcal{T}$ gdw. $\phi(\mathcal{T}) \wedge \phi_x(C)$ erfüllbar ist.
- Es gilt $C \sqsubseteq_{\mathcal{T}} D$ gdw. $\phi(\mathcal{T}) \models \forall x. \phi_x(C) \rightarrow \phi_x(D)$.
- Die ABox- bzw. Wissensbasis $(\mathcal{A}, \mathcal{T}_A)$ ist konsistent gdw. $\phi(\mathcal{A}) \wedge \phi(\mathcal{T}_A)$ erfüllbar ist.
- Es gilt $(\mathcal{A}, \mathcal{T}_A) \models i : C$ gdw. $\phi(\mathcal{A}) \wedge \phi(\mathcal{T}_A) \models \phi(i : C)$.
- Es gilt $(\mathcal{A}, \mathcal{T}_A) \models (i, j) : R$ gdw. $\phi(\mathcal{A}) \wedge \phi(\mathcal{T}_A) \models \phi((i, j) : R)$.

2.2.3. Ausdrucksstarke DLs

Die Modellierungs- bzw. Ausdrucksmächtigkeit von $\mathcal{ALC}$ ist oftmals nicht ausreichend. DLs können im Wesentlichen in drei Dimensionen erweitert werden:

1. Durch zusätzliche konzeptbildende Operatoren,
2. durch Axiome zur Deklaration von Rolleneigenschaften oder gar rollenbildende Operatoren,
3. durch Berücksichtigung einer *konkreten Domäne*.

Diese Dimensionen sind nicht orthogonal. Beispielsweise setzen einige Konstrukte gewisse Rolleneigenschaften voraus. Zudem führen nicht alle Kombinationen zu entscheidbaren DLs. Wir kommen darauf in Kap. 2.2.4 zurück.

2.2.3.1. Das DL-Benennungsschema

Traditionell werden verschiedene Konzept- und Rollen-Konstrukte bzw. Ausdrucksmittel in DLs *durch Buchstaben notiert* – beispielsweise wird der volle Negationsoperator „ $\neg$ “ aus $\mathcal{ALC}$ mittels $\mathcal{C}$ (für *Complement*) notiert, die existentielle qualifizierte Rollenrestriktion mittels $\mathcal{E}$, und die Konzeptdisjunktion durch $\mathcal{U}$.

Brachman und Levesque untersuchten bereits 1984 in [Brachman u. Levesque 1984] die Sprache $\mathcal{FL}^-$, welche lediglich Konjunktion, universelle Rollenrestriktion und *unqualifizierte* existentielle Rollenrestriktion ($\exists R.\top$) bietet; sie zeigten, dass Konzeptsubsumption in $\mathcal{FL}^-$ in polynomieller Zeit entschieden werden kann (Konzepte in $\mathcal{FL}^-$ sind trivialerweise immer erfüllbar). $\mathcal{FL}$ steht für „Frame Language“.

Die Sprache $\mathcal{AL}$, die sog. *Attribute Language*, erweitert $\mathcal{FL}^-$ um *negierte* atomare Konzepte. Zusätzlich bietet $\mathcal{AL}$ zwei spezielle atomare Konzepte, das *Top-Konzept*, notiert als $\top$, und das *Bottom-Konzept*, notiert als $\perp$. In $\mathcal{ALC}$ (nicht jedoch $\mathcal{AL}$) kann $\top$ als Abkürzung für $C \sqcup \neg C$ für ein beliebiges Konzept C angesehen werden, und $\perp$ ist auch in $\mathcal{AL}$ als $C \sqcap \neg C$ schreibbar.

Erweiterungen von $\mathcal{AL}$ werden durch Anhängen von best. Kodebuchstaben gebildet bzw. benannt. Durch Erweiterung von $\mathcal{AL}$ um volle Konzeptnegation (Buchstabe $\mathcal{C}$ für *Complement*) entsteht daher $\mathcal{ALC}$. Dementsprechend können nun Disjunktionen (Buchstabe $\mathcal{U}$) aufgrund der Äquivalenz $\neg(C \sqcap D) \equiv \neg C \sqcup \neg D$ und *qualifizierte* existentielle Rollenrestriktion (Buchstabe $\mathcal{E}$) aufgrund der Äquivalenz $\neg\forall R.C \equiv \exists R.\neg C$ ausgedrückt werden. Aufgrund der Äquivalenzen ist $\mathcal{ALC}$ daher genauso ausdrucksmächtig wie $\mathcal{AL}\mathcal{E}\mathcal{U}$. Jedes $\mathcal{ALC}$ -Konzept kann in die sog. *Negationsnormalform (NNF)* gebracht werden, wobei es sich um ein

zum originalen Konzept äquivalentes $\mathcal{AL}\mathcal{EU}$ -Konzept handelt.

Eine ebenfalls in gewisser Weise „minimale“ DL ist die Sprache $\mathcal{EL}$, welche ausschließlich atomare Konzepte, das $\top$ -Konzept, Konjunktion und qualifizierte existentielle Rollenrestriktion bietet. Erweiterungen von $\mathcal{EL}$ werden sogar aktuell untersucht [Baader u. a. 2005].

2.2.3.2. Konzeptbildende Operatoren

Wie soeben erläutert, werden konzept- und rollenbildende Konstruktoren bzw. Operatoren durch Buchstaben im DL-Namen kodiert. Die wichtigsten zusätzlichen Ausdrucksmittel sind:

- *Unqualifizierte Anzahlrestriktionen, Buchstabe $\mathcal{N}$* : Ein DL mit $\mathcal{N}$ im Namen ermöglicht es, Aussagen über die Anzahl der in den Modellen erzwungenen Rollennachfolger zu machen. So kann z.B. erzwungen werden, dass pro Instanz des Konzeptes *person* genau zwei Individuen als Nachfolger der *hatElternteil*-Rolle vorhanden sind: $(\geq_2 \text{ hatElternteil}) \sqcap (\leq_2 \text{ hatElternteil})$, oder kürzer: $(=_2 \text{ hatElternteil})$.

Wir erweitern die Syntaxdefinition Def. 27 wie folgt:

- Ist R eine Rolle und $n \in \mathbb{N}$, dann sind auch die folgenden Ausdrücke Konzepte: $(\geq_n R)$ (unqualifizierte $\geq$ -Restriktion) und $(\leq_n R)$ (unqualifizierte $\leq$ -Restriktion).

Wir definieren zudem $(=_n R) =_{\text{def}} (\geq_n R) \sqcap (\leq_n R)$.

Für die Interpretationsfunktion gilt:

$$\begin{aligned} (\geq_n R)^{\mathcal{I}} &=_{\text{def}} \{ i \in \Delta^{\mathcal{I}} \mid |\{ j \mid (i, j) \in R^{\mathcal{I}} \}| \geq n \}, \\ (\leq_n R)^{\mathcal{I}} &=_{\text{def}} \{ i \in \Delta^{\mathcal{I}} \mid |\{ j \mid (i, j) \in R^{\mathcal{I}} \}| \leq n \}. \end{aligned}$$

Prädikatenlogisch können wir ϕ_x wie folgt erweitern:

$$\begin{aligned} \phi_x(\geq_n R) &=_{\text{def}} \exists^{\geq n} y. R(x, y), \\ \phi_x(\leq_n R) &=_{\text{def}} \exists^{\leq n} y. R(x, y). \end{aligned}$$

Wie bereits angedeutet, sind $\exists^{\geq n}, \exists^{\leq n}$ die zählenden Quantoren aus $\mathcal{C}_2$. Ihre Umsetzung in $\mathcal{L}_{\mathcal{FOP}\mathcal{L}}$ verlässt natürlich das Zwei-Variablen-Fragment $\mathcal{FO}_2$:

$$\begin{aligned} \exists^{\geq n} y. R(x, y) &=_{\text{def}} \exists y_1, \dots, y_n. \bigwedge_{i,j \in 1..n, i \neq j} y_i \neq y_j \wedge \bigwedge_{i \in 1..n} R(x, y_i), \\ \exists^{\leq n} y. R(x, y) &=_{\text{def}} \forall y_1, \dots, y_{n+1}. \bigwedge_{i,j \in 1..n+1, i \neq j} y_i \neq y_j \rightarrow \bigvee_{i \in 1..n+1} \neg R(x, y_i). \end{aligned}$$

- *Qualifizierte Anzahlrestriktionen, Buchstabe $\mathcal{Q}$* : Während unqualifizierte Anzahlrestriktionen nur etwas über die Anzahl von Rollennachfolgern aussagen können, ist es mit *qualifizierten* Anzahlrestriktionen auch

2.2 Beschreibungslogiken

möglich, etwas über die Konzeptzugehörigkeit dieser zu sagen. So können wir ausdrücken, dass von den beiden Nachfolgern der *hatElternteil*-Rolle einer Instanz des Konzeptes *person* genau ein Nachfolger der *Vater*, und genau ein Nachfolger die *Mutter* sein sollte: $(=_2 \text{ hatElternteil.}\top) \sqcap (=_1 \text{ hatElternteil.vater}) \sqcap (=_1 \text{ hatElternteil.mutter})$.

Wir erweitern die Syntaxdefinition Def. 27 wie folgt:

- Ist R eine Rolle, $n \in \mathbb{N}$ und C ein Konzept, dann sind auch die folgenden Ausdrücke Konzepte: $(\geq_n R.C)$ (qualifizierte $\geq$ -Restriktion) und $(\leq_n R.C)$ (qualifizierte $\leq$ -Restriktion).

Für die Interpretationsfunktion gilt:

$$\begin{aligned} (\geq_n R.C)^{\mathcal{I}} &=_{\text{def}} \{ i \in \Delta^{\mathcal{I}} \mid |\{ j \mid (i, j) \in R^{\mathcal{I}} \wedge j \in C^{\mathcal{I}} \}| \geq n \}, \\ (\leq_n R.C)^{\mathcal{I}} &=_{\text{def}} \{ i \in \Delta^{\mathcal{I}} \mid |\{ j \mid (i, j) \in R^{\mathcal{I}} \wedge j \in C^{\mathcal{I}} \}| \leq n \}. \end{aligned}$$

Wiederum ist $(=_n R.C) =_{\text{def}} (\geq_n R.C) \sqcap (\leq_n R.C)$.

Prädikatenlogisch können wir ϕ wie folgt erweitern:

$$\begin{aligned} \phi_x(\geq_n R.C) &=_{\text{def}} \exists^{\geq n} y. R(x, y) \wedge \phi_y(C), \\ \phi_x(\leq_n R.C) &=_{\text{def}} \exists^{\leq n} y. R(x, y) \wedge \phi_y(C). \end{aligned}$$

- *Funktionswert-Übereinstimmung (Feature Agreement) und -Nichtübereinstimmung (Feature Disagreement)*, Buchstabe $\mathcal{F}$: Bietet die DL funktionale Rollen (Features), wobei es sich einfach um Rollen handelt, die als partielle Funktionen interpretiert werden, so lassen sich mit diesen Konstruktoren Übereinstimmung und Nicht-Übereinstimmung von „Funktionswerten“ bzw. Rollen-Nachfolgern erzwingen. Dabei kann auch die Funktionskomposition verwendet werden.

Intuitiv repräsentiert das Konzept aus der Familien-Domäne

$$(\text{hatVater} \circ \text{hatHaarfarbe} = \text{hatHaarfarbe})$$

diejenigen Individuen, deren Haarfarbe identisch mit der ihrer Väter ist, und das Konzept

$$(\text{hatVater} \circ \text{hatHaarfarbe} \neq \text{hatHaarfarbe})$$

die Individuen, deren Väter eine andere Haarfarbe haben.

Die Syntax der beiden Konstruktoren ist:

- Sind $f = f_1 \circ \dots \circ f_n$ und $g = g_1 \circ \dots \circ g_m$ Ketten funktionaler Rollen (Feature Chains), dann sind auch die folgenden Ausdrücke Konzepte: $(f = g)$, $(f \neq g)$.

Für die Interpretationsfunktion gilt:

$$(f = g)^{\mathcal{I}} =_{\text{def}} \{ i \in \Delta^{\mathcal{I}} \mid (i, j) \in (f_1^{\mathcal{I}} \circ \dots \circ f_n^{\mathcal{I}}), (i, k) \in (g_1^{\mathcal{I}} \circ \dots \circ g_m^{\mathcal{I}}), j = k \},$$

$$(f \neq g)^{\mathcal{I}} =_{def} \{ i \in \Delta^{\mathcal{I}} \mid (i, j) \in (f_1^{\mathcal{I}} \circ \dots \circ f_n^{\mathcal{I}}), (i, k) \in (g_1^{\mathcal{I}} \circ \dots \circ g_m^{\mathcal{I}}), j \neq k \}.$$

Intuitiv ist also $i \in (f = g)^{\mathcal{I}}$, wenn durch Traversierung (Funktionskomposition) der f_j -Nachfolger (beginnend bei i) das gleiche Individuen aus $\Delta^{\mathcal{I}}$ wie durch Traversierung der g_k -Nachfolger (beginnend bei i) erreicht wird. Die durch Traversierung erhaltenen Objekte sind eindeutig, da $f_j^{\mathcal{I}}$, $g_k^{\mathcal{I}}$ (partielle) Funktionen sind.

Wir sparen uns eine prädikatenlogische Übersetzung.

- **Role Values Maps (RVMs):**

Role Value Maps können als verallgemeinerte Funktionswert-Übereinkunft für Rollen verstanden werden. So könnten wir in der Familien-Domäne für Instanzen des Konzeptes *person* fordern wollen, dass die Menge der *hatNichte*-Nachfolger stets eine Teilmenge der *hatTochter*-Nachfolger der *hatGeschwister*-Nachfolger der Instanz ist:

$$hatNichte \subseteq hatGeschwister \circ hatTochter.$$

Wir würden also $hatNichte \subseteq hatGeschwister \circ hatTochter$ als Teil der Definition von *person* fordern.

Sind $R = R_1 \circ \dots \circ R_n$ und $S = S_1 \circ \dots \circ S_m$ sog. *Rollen-Ketten*, so ist das Konzept $R \subseteq S$ eine sog. *Role Value Map (RVM)*. Die Semantik ist:

$$(R \subseteq S)^{\mathcal{I}} =_{def} \{ i \in \Delta^{\mathcal{I}} \mid \forall j. (i, j) \in (R_1^{\mathcal{I}} \circ \dots \circ R_n^{\mathcal{I}}) \rightarrow (i, j) \in (S_1^{\mathcal{I}} \circ \dots \circ S_m^{\mathcal{I}}) \}.$$

- **Nominale (Nominals), Buchstabe $\mathcal{O}$:** Nominale werden von der Interpretationsfunktion auf einelementige Teilmengen von $\Delta^{\mathcal{I}}$ abgebildet (sog. *Singletons*). Beispielsweise könnte das Konzept *deutschland* mit Hilfe des Nominals $\{berlin\}$ wie folgt definiert werden: $deutschland \doteq \exists hatHauptstadt. \{berlin\}$.

Sei $\mathcal{N}_{\mathcal{N}}$ eine Menge von Nominalen. Ist $i \in \mathcal{N}_{\mathcal{N}}$, so ist i ein Konzept. Für die Semantik fordern wir, dass $^{\mathcal{I}}$ Elemente aus $\mathcal{N}_{\mathcal{N}}$ auf Singletons in $\Delta^{\mathcal{I}}$ abbildet. Mengen von Nominalen $\{i_1, \dots, i_n\}$ können als Disjunktion $\{i_1\} \sqcup \dots \sqcup \{i_n\}$ formuliert werden (sofern die DL $\mathcal{U}$ bietet).

Mit Hilfe von Nominalen können ABoxen als Konzepte repräsentiert und ähnlich wie TBoxen „internalisiert“ werden: Eine Rollen-Assertion $(i, j) : R$ kann dann als Konzept $\{i\} \sqcap \exists R. \{j\}$ geschrieben werden, und eine Konzept-Assertion $i : C$ als $\{i\} \sqcap C$. Eine ausdrucksstarke DL mit Nominalen ermöglicht daher die Internalisierung der gesamten Wissensbasis in ein Konzept (vgl. auch [Wessel 2003a]).

2.2.3.3. Axiome zur Deklaration von Rolleneigenschaften und komplexe Rollen

Bisher haben wir jede binäre Relation als mögliche Interpretation für eine Rolle zugelassen. Für Modellierungszwecke werden jedoch oftmals funktionale, tran-

2.2 Beschreibungslogiken

sitiv abgeschlossene, symmetrische, zueinander inverse, reflexive, irreflexive, zueinander disjunkte Relationen, oder Relationskomposition benötigt. Die Liste ließe sich beliebig fortsetzen.

Standard-DLs bieten

- *Transitiv abgeschlossene Rollen, Zeichen $\mathcal{R}^+$* : Beispiele für transitiv abgeschlossene Rollen in der Familien-Domäne sind die Rollen *hatVorfahre* und *hatNachfahre*.

Statt transitiv abgeschlossene Rolle sagen wir im Folgenden auch einfach *transitive Rolle*. Für eine transitive Rolle R muss $R^{\mathcal{I}} = (R^{\mathcal{I}})^+$ gelten.

Bietet die DL TBoxen, so kann durch ein TBox-Axiom der Art *transitive*(R) die Rolle R als transitiv abgeschlossen deklariert werden, und wir definieren $\mathcal{I} \models \text{transitiv}(R)$ gdw. $R^{\mathcal{I}} = (R^{\mathcal{I}})^+$. Alternativ könnte ein solches Axiom auch in eine spezielle Rollen-Box bzw. *RBox* eingetragen werden, um die TBox nicht „zu verschmutzen“.

Oftmals wird auch einfach eine Menge transitiver Rollen $\mathcal{N}_{\mathcal{R}^+} \subseteq \mathcal{N}_{\mathcal{R}}$ vereinbart, sodass auf zusätzliche Axiome verzichtet werden kann (bzw. die *transitive*-Axiome sind dann implizit vorhanden).

Die ϕ -Funktion kann wie folgt für derartige Axiome erweitert werden:

$$\phi(\text{transitive}(R)) =_{\text{def}} \forall x, y, z. R(x, y) \wedge R(y, z) \rightarrow R(x, z).$$

- *Der transitive Abschluss-Operator, Zeichen $+$* :

Hierbei handelt es sich um einen *rollenbildenden Operator* $\cdot^+$, der, angewendet auf eine Rolle R , den transitiven Abschluss R^+ von R bildet. In der Familien-Domäne wäre z.B. *hatVorfahre* $\equiv$ *hatElternteil* $^+$ ein sinnvolles Rollenaxiom.

Die Syntax der Rollen wird daher um folgende Regel erweitert:

- Ist $R \in \mathcal{N}_{\mathcal{R}}$ eine Rolle, so auch R^+ .

Für die Interpretationsfunktion fordern wir dann:

$$(R^+)^{\mathcal{I}} = (R^{\mathcal{I}})^+.$$

In der TBox (bzw. RBox) sind wiederum Axiome der Art $R \dot{=} S^+$ denkbar, sodass $\mathcal{I} \models R \dot{=} S^+$ gdw. $R^{\mathcal{I}} = (S^+)^{\mathcal{I}} = (S^{\mathcal{I}})^+$. Natürlich kann R dann auch mittels $R \dot{=} R^+$ als transitiv abgeschlossen deklariert werden.

Es ist bekannt, dass der $\cdot^+$ -Operator aufgrund des Kompaktheitssatzes nicht prädikatenlogisch axiomatisiert werden kann.

- *Funktionale Rollen, Buchstabe f* : Funktionale Rollen müssen als partielle Funktionen interpretiert werden. In der Familien-Domäne sollten die Rollen *hatVater* und *hatMutter* funktional sein. Wiederum können wir eine Menge von funktionalen Rollen $\mathcal{N}_{\mathcal{F}} \subseteq \mathcal{N}_{\mathcal{R}}$ annehmen, oder entspr. TBox-

bzw. RBox-Axiome $functional(F)$ vorsehen.

Für $F \in \mathcal{N}_{\mathcal{F}}$ verschärfen wir in Def. 29 dann die Bedingungen an $\cdot^{\mathcal{I}}$, indem wir fordern: „... und funktionale Rollen auf einstellige Funktionen abbildet“. Alternativ kann auch

$$\mathcal{I} \models functional(F) \text{ gdw. } \{i \mid (i, j) \in F^{\mathcal{I}}, (i, k) \in F^{\mathcal{I}}, j \neq k\} = \emptyset$$

gefordert werden.

Die ϕ -Funktion kann wie folgt erweitert werden:

$$\phi(functional(F)) =_{def} \forall x, y, z. F(x, y) \wedge F(x, z) \rightarrow y = z.$$

- **Rollenhierarchien, Buchstabe $\mathcal{H}$ (und h , s. Text):** Ein **Rolleninklusionsaxiom** $R \sqsubseteq S$ deklariert S als **Oberrolle** von R , bzw. R als **Unterrolle** von S . Sinnvolle Rolleninklusionen in der Familien-Domäne sind $hatVater \sqsubseteq hatElternteil$ und $hatMutter \sqsubseteq hatElternteil$.

Eine DL-Interpretation $\mathcal{I}$ erfüllt ein Rolleninklusionsaxiom $\mathcal{I} \models R \sqsubseteq S$ gdw. $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ gilt.

Die ϕ -Funktion kann wie folgt erweitert werden:

$$\phi(R \sqsubseteq S) =_{def} \forall x, y. R(x, y) \rightarrow S(x, y).$$

Die **Rollenhierarchie** muss genau wie die Taxonomie anhand der gegebenen Rolleninklusionsaxiome berechnet werden. Wie bereits für die Konzept-Subsumptionsrelation besprochen, hat $\sqsubseteq$ die Eigenschaften einer Quasiordnung. Die Äquivalenzklassen sind die zueinander synonymen Rollen, z.B. werden die Rollen R und S durch $R \sqsubseteq S$, $S \sqsubseteq R$ als synonym deklariert. Viele DL-Systeme erlauben keine zyklischen Rolleninklusionsaxiome. Die Rollenhierarchie ist also die partielle Ordnung, die durch Äquivalenzklassen-Faktorisierung entsteht. Einige DL-Systeme (z.B. CLASSIC) lassen nur *Bäume* für die Rollenhierarchie zu – wir verwenden hierfür den Kodebuchstaben h (statt $\mathcal{H}$).

- **Inverse Rollen, Buchstabe $\mathcal{I}$:** Mittels des rollenbildenden Operators $\cdot^{-1}$ kann die inverse Rolle R^{-1} einer Rolle R gebildet werden. So kann beispielsweise $hatKind^{-1}$ für die (evtl. nicht vorhandene) Rolle $hatElternteil$ genutzt werden, oder ein Axiom $hatKind \dot{=} hatElternteil^{-1}$ verwendet werden.

Die Syntax der Rollen wird um folgende Regel erweitert:

- Ist $R \in \mathcal{N}_{\mathcal{R}}$ eine Rolle, so auch R^{-1} .

Für die Interpretationsfunktion fordern wir dann:

$$(R^{\mathcal{I}})^{-1} = (R^{-1})^{\mathcal{I}}$$

Für die entspr. Axiome können wir wieder definieren, dass $\mathcal{I} \models R \dot{=} S^{-1}$ gdw. $R^{\mathcal{I}} = (S^{-1})^{\mathcal{I}} = (S^{\mathcal{I}})^{-1}$ gilt.

Es ist klar, dass die ϕ -Funktion wie folgt erweitert werden kann:

2.2 Beschreibungslogiken

$$\phi(R \dot{=} S^{-1}) =_{def} \forall x, y. R(x, y) \leftrightarrow S(y, x).$$

- *Bild- und Urbild-Restriktionen für Rollen:*

Urbild und *Bild* einer Relation (s. auch Def. 6) bzw. interpretierten Rolle R sind wie folgt definiert:

$$\begin{aligned} \text{domain}(R^{\mathcal{I}}) &=_{def} \{ i \mid (i, j) \in R^{\mathcal{I}} \}, \\ \text{range}(R^{\mathcal{I}}) &=_{def} \{ j \mid (i, j) \in R^{\mathcal{I}} \}. \end{aligned}$$

Eine Bild-Einschränkung der Rolle R auf D erzwingt $\text{range}(R^{\mathcal{I}}) \subseteq D^{\mathcal{I}}$, und eine Urbild-Einschränkung von R auf C erzwingt $\text{domain}(R^{\mathcal{I}}) \subseteq C^{\mathcal{I}}$, sodass $R^{\mathcal{I}} \subseteq C^{\mathcal{I}} \times D^{\mathcal{I}}$.

In der Familien-Domäne wäre z.B. $\text{hatMutter}^{\mathcal{I}} \subseteq \text{person}^{\mathcal{I}} \times \text{Frau}^{\mathcal{I}}$ eine sinnvolle Urbild- und Bild-Restriktion.

Um nun für eine Rolle R in allen Modellen $R^{\mathcal{I}} \subseteq C^{\mathcal{I}} \times D^{\mathcal{I}}$ erzwingen zu können, können entweder spezielle Axiome der Art $\text{domain}(R) \dot{\subseteq} C$ und $\text{range}(R) \dot{\subseteq} D$ oder aber, falls die DL allgemeine TBoxen bietet, GCIs der Form $\exists R. \top \dot{\subseteq} C$ und $\top \dot{\subseteq} \forall R. D$ verwendet werden.

- *Axiome für Anwendungen von DLs im qualitativen räumlichen Schließen:*

Im Rahmen dieser Arbeit und im Vorgriff auf Kap. 3 hat der Autor auch DLs mit Rollen-Axiomen untersucht, die durch Anwendungen von DLs im qualitativen räumlichen Schließen motiviert wurden, und die den folgenden prädikatenlogischen Sätzen entsprechen:

- Rollendisjunktheit: $\forall x, y. R(x, y) \wedge S(x, y) \rightarrow \perp$,
- Kompositionsbasierte Rolleninklusionsaxiome: $\forall x, y, z. R(x, y) \wedge S(y, z) \rightarrow T_1(x, z) \vee \dots \vee T_n(x, z)$, und
- Globale Verbundenheit: $\forall x, y. R_1(x, y) \vee \dots \vee R_n(x, y)$.

Mehr hierzu in Kap. 3.

Wir haben die Operatoren $\cdot^+$ und $\cdot^{-1}$ *rollenbildende Operatoren* bzw. *Rollenkonstruktoren* genannt, während Operatoren wie $\mathcal{H}, \mathcal{R}^+$ etc. globalen Deklarationen bzw. Axiomen entsprechen. Eine Rolle, die nicht nur ein Rollenname, sondern durch einen Rollenausdruck mittels rollenbildender Operatoren beschrieben ist (z.B. R^{-1} und R^+), wird auch *komplexe Rolle* genannt. Eine sehr allgemeine Definition der Rollen-Syntax könnte daher wie folgt aussehen; die entspr. Konstrukturen wurden in der Literatur untersucht:

- Jedes $R \in \mathcal{N}_{\mathcal{R}}$ ist eine Rolle.
- Sind R und S Rollen, so sind auch die folgenden Ausdrücke Rollen:
 - R^{-1} (Rolleninversion, Buchstabe $\mathcal{I}$, Operator $\cdot^{\mathcal{I}}$),
 - R^+ (transitiver Rollenabschluss, Operator $\cdot^+$),

- $\neg R$ (Rollennegation),
- $R \sqcup S$ (Rollendisjunktion),
- $R \sqcap S$ (Rollenkonjunktion, Buchstabe $\mathcal{R}$),
- $R \circ S$ (Rollenkomposition, Operator $\circ$).

Die *regulären Operatoren* sind durch die Menge $\{\sqcup, \circ, \cdot^*\}$ gegeben. Der Operator $\cdot^*$ konstruiert die *reflexive transitive Hülle* der Rolle. $\mathcal{ALC}$ erweitert um diese Rollenkonstruktoren wird $\mathcal{ALC}_{\mathcal{REG}}$ genannt.

Die entsprechenden Anforderungen an die Interpretationsfunktion sind offensichtlich (z.B. muss $(R \sqcap S)^{\mathcal{I}} = R^{\mathcal{I}} \cap S^{\mathcal{I}}$ gelten). Die rollenbildenden Operatoren Rollendisjunktion und Rollenkomposition sind z.B. in $\mathcal{ALC}$ bereits „implizit vorhanden“, da die folgenden Äquivalenzen gelten: $\exists(R \sqcup S).C \equiv \exists R.C \sqcup \exists S.C$, $\forall(R \sqcup S).C \equiv \forall R.C \sqcap \forall S.C$, $\exists(R \circ S).C \equiv \exists R.\exists S.C$, $\forall(R \circ S).C \equiv \forall R.\forall S.C$. Zudem gelten die Äquivalenzen $\forall R^*.C \equiv C \sqcap \forall R^+.C$ und $\exists R^*.C \equiv C \sqcap \exists R^+.C$. Daher ist bereits die Menge $\{\sqcup, \circ, \cdot^+\}$ ausreichend für $\mathcal{ALC}_{\mathcal{REG}}$.

2.2.3.4. Konkrete Domänen

Für viele Konzeptualisierungen werden sog. *konkrete Domänen* benötigt [Baader u. Hanschke 1991a]. In der Familien-Domäne wird eine adäquate Konzeptualisierung des Begriffs *Erwachsener* u.a. implizieren, dass das Alter größer als 18 sein muss. Um nun mathematische Strukturen wie die natürlichen Zahlen und die in ihnen geltenden Gesetzmäßigkeiten in die Konzeptualisierung einzubeziehen, kann eine DL um sog. *konkrete Domänen* erweitert werden. Die $<$ -Relation in $\mathbb{N}$ ist z.B. eine Striktordnung und hat daher gewisse mathematische Eigenschaften. Ohne konkrete Domänen könnten die Eigenschaften dieser Prädikate nur unvollständig mit beschreibungslogischen Mitteln abgebildet werden. Im Gegensatz zu den Konzept- und Rollennamen, welche ja unären und binären Prädikatsnamen entsprechen, müssen die Prädikatsnamen aus einer konkreten Domäne also auf eine feste, „konkrete“ Weise interpretiert werden. Beispielsweise wird $<$ in $\mathbb{N}$ zwingend als die $<$ -Relation auf $\mathbb{N}$ interpretiert.

Eine konkrete Domäne ist formal wie folgt definiert [Baader u. Hanschke 1991a]:

Definition 43 (Konkrete Domäne) Eine *konkrete Domäne (KD)* $\mathcal{D}$ ist ein Paar $(\Delta^{\mathcal{D}}, \mathcal{P}^{\mathcal{D}})$, wobei $\Delta^{\mathcal{D}}$ der Grundbereich (Domäne) der konkreten Domäne, und $\mathcal{P}^{\mathcal{D}}$ eine Menge von Prädikatsnamen ist. Jeder Prädikatsname P hat eine Stelligkeit n (wir notieren diese bei Bedarf mittels P^n) und ist mit einer n -stelligen Relation assoziiert: $P^{\mathcal{D}} \subseteq (\Delta^{\mathcal{D}})^n$. ■

Man beachte, dass die Semantik eines Prädikates $P \in \mathcal{P}^{\mathcal{D}}$ durch $P^{\mathcal{D}} \subseteq (\Delta^{\mathcal{D}})^n$ festgelegt (also „konkret“) ist. Damit *entscheidbare* Inferenz mit konkreten

2.2 Beschreibungslogiken

Domänen sichergestellt ist, muss eine konkrete Domäne *zulässig* sein:

Definition 44 (Zulässigkeit) Eine konkrete Domäne $\mathcal{D} = (\Delta^{\mathcal{D}}, \mathcal{P}^{\mathcal{D}})$ heißt *zulässig* (*admissible*), wenn die folgenden Bedingungen erfüllt sind:

1. Die Menge $\mathcal{P}^{\mathcal{D}}$ ist abgeschlossen unter Negation.
2. Die Menge $\mathcal{P}^{\mathcal{D}}$ enthält einen Prädikatsnamen $\top_{\Delta^{\mathcal{D}}}$.
3. Das *Erfüllbarkeitsproblem* für endliche Konjunktionen der Form $\bigwedge_{i=1}^n P_i(\vec{x}_i)$ ist entscheidbar. Hierbei sind $\vec{x}_i$ die Variablenvektoren der $P_i \in \mathcal{P}^{\mathcal{D}}$. ■

Eine nützliche zulässige konkrete Domäne ist beispielsweise $\mathcal{D}_{\mathbb{N}} = (\mathbb{N}, \{<, \leq, >, \geq, <_n, \leq_n, >_n, \geq_n\})$. Hier sind $<_n$ etc. unäre Prädikate, für alle $n \in \mathbb{N}$. Eine weitere bekannte zulässige konkrete Domäne ist die sog. *Tarski-Domäne*, $\mathcal{D}_{\mathbb{R}}$: Hier ist $\Delta_{\mathbb{R}}^{\mathcal{D}} = \mathbb{R}$, und $\mathcal{P}_{\mathbb{R}}^{\mathcal{D}}$ ist durch die Menge aller Formeln gegeben, die mit Sprachmitteln erster Ordnung über Gleichungen und Ungleichungen von Ganzzahl-Polynomen mit beliebig vielen Variablen definiert werden können. Sind beispielsweise $P_1[x] =_{\text{def}} (x+x=x)$ und $P_2[x] =_{\text{def}} x > 0$ solche Gleichungen, so kann entschieden werden, dass $P_1 \wedge P_2$ unerfüllbar ist.

Konkrete Domänen sind nicht auf numerische Domänen beschränkt – sie werden auch zur Repräsentation von „Daten“ bzw. Literalen wie Zeichenketten etc. verwendet. Wir verwenden in dieser Arbeit auch die konkrete Domäne der Zeichenketten, welche Prädikate für Gleichheit und Ungleichheit von Zeichenketten enthält, $\mathcal{D}_{\mathbb{S}} = (\Delta_{\mathbb{S}}^{\mathcal{D}}, \{=, \neq\})$.

Um nun eine DL um eine konkrete Domäne zu erweitern, wird ein neuer konzeptbildender Operator benötigt: Der sog. *prädikatsbasierte Existenz-Operator*. Mit Hilfe dieses Operators kann die Existenz von n Elementen in der konkreten Domäne $\mathcal{D}$ gefordert werden, die ein gewisses n -stelliges Prädikat P^n erfüllen. Die Assoziation von Individuen aus der abstrakten Domäne $\Delta^{\mathcal{I}}$ mit Elementen in der konkreten Domäne $\Delta^{\mathcal{D}}$ geschieht über funktionale Rollen. Die Interpretation einer funktionalen Rolle $F \in \mathcal{N}_{\mathcal{F}}$ wird dann so erweitert, dass für $F^{\mathcal{I}}$ eine partielle Funktion $\Delta^{\mathcal{I}} \mapsto (\Delta^{\mathcal{I}} \cup \Delta^{\mathcal{D}})$ zulässig ist. Aus Gründen der Einfachheit wird oftmals jedoch einfach eine neue Art von funktionalen Rollen, die ausschließlich in die konkrete Domäne abbilden, eingeführt (z.B. im RACERPRO-System, [Haarslev u. Möller 2001c]). Wir nennen diese *Attribute*. Wir sehen eine Menge von Attributnamen $\mathcal{N}_{\mathcal{A}}$ vor; für $A \in \mathcal{N}_{\mathcal{A}}$ fordern wir dann, dass $A^{\mathcal{I}}$ eine partielle Funktion vom Typ $\Delta^{\mathcal{I}} \mapsto \Delta^{\mathcal{D}}$ ist.

Wir erweitern die Syntaxdefinition Def. 27 wie folgt:

- Sei $F_i \in \mathcal{N}_{\mathcal{F}}$, $A \in \mathcal{N}_{\mathcal{A}}$. Eine *KD-Kette* ist ein Ausdruck der Form $F_1 \circ \dots \circ F_n \circ A$. Sind $f_1, \dots, f_m$ KD-Ketten, und ist $P^n \in \mathcal{P}^{\mathcal{D}}$ ein n -stelliges Prädikat, so ist der Ausdruck $\exists(f_1, \dots, f_m).P$ ein Konzept (sog. prädikatsbasierter Existenz-Operator).

Die Anwesenheit einer konkreten Domäne (und des prädikatsbasierten Existenz-Operators) wird im Namen der DL durch einen Postfix „...($\mathcal{D}$)“ angezeigt, z.B. $\mathcal{ALC}(\mathcal{D})$.

Ist $f = F_1 \circ \dots \circ F_n \circ A$ nun eine KD-Kette, so ist $f^{\mathcal{I}} =_{def} F_1^{\mathcal{I}} \circ \dots \circ F_n^{\mathcal{I}} \circ A^{\mathcal{I}}$. Die Semantik von $\exists(f_1, \dots, f_n).P$ ist nun wie erwartet definiert:

$$(\exists(f_1, \dots, f_n).P^n)^{\mathcal{I}} =_{def} \{ i \mid i \in \Delta^{\mathcal{I}}, \\ \exists cd_1, \dots, cd_n \in \Delta^{\mathcal{D}}. (i, cd_1) \in f_1^{\mathcal{I}}, \\ \vdots \\ (i, cd_n) \in f_n^{\mathcal{I}}, \\ (cd_1, \dots, cd_n) \in P^{\mathcal{D}} \}.$$

Oftmals werden (z.B. aus Gründen der Entscheidbarkeit) lediglich *eingeschränkte Versionen* des prädikatsbasierten Existenz-Operators zugelassen; diese werden dann durch einen entspr. modifizierten Postfix (z.B. „...($\mathcal{D}^-$)“) angezeigt. Die vom RACERPRO-System implementierte DL $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ lässt aus Gründen der Entscheidbarkeit z.B. lediglich einzelne Attribute statt KD-Ketten im prädikatsbasierten Existenz-Operator zu.

Bietet die DL ABoxen, so werden für konkrete Domänen i.d.R. weitere ABox-Assertionen zugelassen. Seien $a_1, \dots, a_n$ eine weitere Art von Individuen, die sog. *KD-Individuen*. Wir nehmen eine Menge von KD-Individuen $\mathcal{N}_{\mathcal{CD}}$ an, d.h., $a_i \in \mathcal{N}_{\mathcal{CD}}$. Hierbei handelt es sich um Namen (Konstanten) für Elemente in $\Delta^{\mathcal{D}}$. Die Mengen der KD-Individuen und ABox-Individuen sind disjunkt: $\mathcal{N}_{\mathcal{CD}} \cap \mathcal{N}_{\mathcal{I}} = \emptyset$. Für eine gegebene ABox $\mathcal{A}$ bezeichnen wir die Menge der in ihr verwendeten KD-Individuen mit $cd\_inds(\mathcal{A})$.

Wir erweitern Def. 39 um neue ABox-Assertionen wie folgt:

- Ist $i \in \mathcal{N}_{\mathcal{I}}$, $a \in \mathcal{N}_{\mathcal{CD}}$, $A \in \mathcal{N}_{\mathcal{A}}$, so ist $(i, a) : A$ eine ABox-Assertion, die *Attributs-Assertion*.
- Sind $a_i \in \mathcal{N}_{\mathcal{CD}}$ und $P^n \in \mathcal{P}^{\mathcal{D}}$, so ist $(a_1, \dots, a_n) : P^n$ eine ABox-Assertion, die *Prädikats-Assertion*.

Semantisch fordern wir nun, dass $\cdot^{\mathcal{I}} \mathcal{N}_{\mathcal{CD}}$ (bzw. wenigstens $cd\_inds(\mathcal{A})$) auf $\Delta^{\mathcal{D}}$ abbildet. Für die neuen Assertionen definieren wir Erfülltsein (Wahrheit) in $\mathcal{I}$ wie folgt:

- $\mathcal{I} \models (i, a) : A$ gdw. $(i^{\mathcal{I}}, a^{\mathcal{I}}) \in A^{\mathcal{I}}$, und
- $\mathcal{I} \models (a_1, \dots, a_n) : P$ gdw. $(a_1^{\mathcal{I}}, \dots, a_n^{\mathcal{I}}) \in P^{\mathcal{D}}$.

Zwei neue Inferenzprobleme werden wie folgt definiert:

- Das *Constraint-Prüfungsproblem*: Gegeben ein Tupel von Attributen $(A_1, \dots, A_n)$ mit $A_i \in \mathcal{N}_{\mathcal{A}}$, ein Tupel $(i_1, \dots, i_n)$ von ABox-Individuen mit

2.2 Beschreibungslogiken

$i_j \in \text{inds}(\mathcal{A})$ und ein Prädikat $P^n \in \mathcal{P}^{\mathcal{D}}$, gilt dann in allen Modellen $\mathcal{I}$ der WB $(A_1^{\mathcal{I}}(i_1^{\mathcal{I}}), \dots, A_n^{\mathcal{I}}(i_n^{\mathcal{I}})) \in P^{\mathcal{D}}$?

- Das *eingeschränkte Constraint-Prüfungsproblem*: Gegeben ein Tupel von KD-Individuen $(a_1, \dots, a_n)$ mit $a_i \in \text{cd\_inds}(\mathcal{A})$ und ein Prädikat $P^n \in \mathcal{P}^{\mathcal{D}}$, gilt dann in allen Modellen $\mathcal{I}$ der WB $(a_1^{\mathcal{I}}, \dots, a_n^{\mathcal{I}}) \in P^{\mathcal{D}}$? Wir schreiben dann $WB \models (a_1, \dots, a_n) : P$.

Anhand dieser beiden Entscheidungsprobleme lassen sich wieder entsprechende Retrieval-Probleme definieren.

2.2.4. Komplexität

Die Komplexität von DLs ist heutzutage sehr gut untersucht. Die *Komplexität einer Logik* ist die Komplexität (Worst Case Complexity) des Erfüllbarkeitsproblems (wobei es sich um das zentrale Inferenzproblem handelt). In einfachen Sprachen wie $\mathcal{FL}^-$, deren Konzepte stets erfüllbar sind, wird bzw. wurde stattdessen die Komplexität des Subsumptionsproblems untersucht. Wir möchten hier nur die wichtigsten „Eckpfeiler“ der für diese Arbeit relevanten DLs zitieren [Baader u. a. 2003a, Kap. 3]:

- $\mathcal{ALC}$: PSPACE-vollständig ohne TBoxen [Schmidt-Schauß u. Smolka 1991] oder mit einfachen TBoxen [Lutz 1999], aber EXPTIME-vollständig für allgemeine TBoxen (dies gilt bereits für $\mathcal{AL}$).
- $\mathcal{ALCF}$: PSPACE-vollständig ohne TBoxen [Hollunder u. Nutt 1990], NEXPTIME-vollständig für einfache TBoxen, unentscheidbar für allgemeine TBoxen (das berühmte unentscheidbare Domino-Problem kann sehr einfach auf das $\mathcal{ALCF}$ -Konzepterfüllbarkeitsproblem reduziert werden) [Lutz 1999].
- $\mathcal{ALC}_{\mathcal{R}^+}(S)$, $\mathcal{ALC}_+$, $\mathcal{ALC}_{\mathcal{REG}}$: EXPTIME-vollständig mit allgemeinen TBoxen. Dies gilt auch für das ABox- bzw. Wissensbasis-Erfüllbarkeitsproblem. Die Sprache $\mathcal{ALC}_{\mathcal{R}^+}$ wird aufgrund der engen Verwandtschaft zur Multimodallogik S_4 auch S genannt, s.u.
- $\mathcal{ALCH}_{\mathcal{R}^+}(S)$: EXPTIME-vollständig (unabhängig von der Art der TBox). Dies gilt auch für das ABox-Erfüllbarkeitsproblem.
- $\mathcal{ALCNH}_{\mathcal{R}^+}(SHN)$: Unentscheidbar (Beweis durch Reduktion des Domino-Problems), *wenn nicht* die Anzahlrestriktionen so eingeschränkt werden, dass in diesen lediglich sog. *einfache Rollen* verwendet werden dürfen [Horrocks u. a. 2000b].

Eine *einfache Rolle* R ist eine Rolle, die keine transitive Unterrolle S hat:

$$\neg \exists S. S \in \mathcal{N}_{\mathcal{R}^+}, S \sqsubseteq R.^{18}$$

Diese Annahme wird im Folgenden stets gemacht. Die so eingeschränkte Sprache $\mathcal{ALCHN}_{\mathcal{R}^+}$ ist dann EXPTIME-vollständig (unabhängig von der Art der TBox). Dies gilt auch für das ABox-Erfüllbarkeitsproblem.

- $\mathcal{ALCQHI}_{\mathcal{R}^+}$ (auch $\mathcal{SHIQ}$ genannt): Selbst wenn $\mathcal{ALCHN}_{\mathcal{R}^+}$ um inverse Rollen und qualifizierte Anzahlrestriktionen erweitert wird, bleibt die Sprache noch in EXPTIME; sowohl $\mathcal{SHIQ}$ als auch $\mathcal{SHIN}$ sind EXPTIME-vollständig [Tobies 2001]. Dies gilt auch für das ABox-Erfüllbarkeitsproblem.
- Die nächste Komplexitätsklasse NEXPTIME wird erst für Sprachen wie $\mathcal{SHOIN}$ und $\mathcal{SHOIQ}$ erreicht. Diese Sprachen sind NEXPTIME-vollständig [Tobies 2000]. Dies gilt auch für das ABox-Erfüllbarkeitsproblem.
- $\mathcal{ALC}(\mathcal{D})$ ist PSPACE-vollständig. $\mathcal{ALC}(\mathcal{D})$ mit einfachen TBoxen ist bereits NEXPTIME-vollständig [Lutz 2001]. $\mathcal{S}$ mit konkreter Domänen, also $\mathcal{S}(\mathcal{D})$, ist unentscheidbar [Baader u. Hanschke 1991b].

Wie schon erwähnt, werden daher lediglich eingeschränkte Versionen des prädikatsbasierten Existenzoperators $\exists(f_1, \dots, f_n).P$ in DL-Systemen angeboten. So lässt z.B. das RACERPRO-System lediglich Attribute A_i für die f_i zu. Die so eingeschränkte konkrete Domäne wird dann mit $\mathcal{D}^-$ bezeichnet [Haarslev u. a. 2001b].

- Sprachen mit *Role Value Maps* (RVMs) sind i.d.R. unentscheidbar, z.B. bereits $\mathcal{FL}^-$ mit RVMs. KL-ONE wurde aufgrund der RVMs in [Schmidt-Schauß 1989] als unentscheidbar identifiziert.
- Die vom Autor untersuchten Sprachen $\mathcal{ALC}_{\mathcal{RA}^\oplus}$, $\mathcal{ALC}_{\mathcal{RA}}$ (s. Kap. 3) sind unentscheidbar; eine entscheidbare $\mathcal{ALC}_{\mathcal{RA}}$ -Spezialisierung namens $\mathcal{ALC}_{\mathcal{RASG}}$ wurde als EXPTIME-vollständig nachgewiesen. $\mathcal{ALCN}_{\mathcal{RASG}}$ ist jedoch wieder unentscheidbar (per Reduktion des Domino-Problems). Die sog. $\mathcal{ALCI}_{\mathcal{RCC}}$ -DLs sind spezielle $\mathcal{ALCI}_{\mathcal{RA}}$ -Instantiierungen und werden ebenfalls in Kap. 3 vorgestellt.

Höhere Expressivität kann nur zum Preis erhöhter Berechnungskomplexität erzielt werden. Zudem ist die Endliches-Modell-Eigenschaft für ausdrucksstarke DLs nicht mehr gegeben, sobald inverse, funktionale und transitive Rollen möglich sind. So hat z.B. das $\mathcal{ALCHIf}_{\mathcal{R}^+}$ - bzw. $\mathcal{ALCHIN}_{\mathcal{R}^+}$ -Konzept

$$\neg C \sqcap \exists F^{-1}.(C \sqcap \leq_1 F \sqcap \forall R^{-1}.\exists F^{-1}.(C \sqcap \leq_1 F))$$

mit $F \in \mathcal{N}_{\mathcal{F}}$, $R \in \mathcal{N}_{\mathcal{R}^+}$ und $F \sqsubseteq R$ lediglich unendliche Modelle [Horrocks u. a. 1999b].

¹⁸Wie beschrieben, ist „ $\sqsubseteq$ “ die *inferierte* Rollenhierarchie, nicht zu verwechseln mit der per „ $\sqsubseteq^+$ “ *zugesicherten* Rollenhierarchie. Daher gilt z.B. auch $R \sqsubseteq R$; m.a.W., R selbst darf ebenfalls nicht transitiv sein.

2.2.5. Eine Beispiel-Wissensbasis

Zur Illustration von DL-Wissensbasen und -Standardinferenzen verwenden wir eine $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ -Wissensbasis, deren TBox in Abb. 2.2, und deren ABox in Abb. 2.3 angegeben ist. Wir modellieren bzw. konzeptualisieren und formalisieren eine Familien-Domäne.

Die erste Gruppe von TBox-Axiomen in Abb. 2.2 sind Rollenaxiome. Die zweite Gruppe enthält die konzeptspezifischen Axiome. Die Bedeutung der einzelnen Axiome sollte aufgrund der vorangegangenen Erläuterungen klar sein. Wir verwenden hier die konkreten Domänen $(\Delta_{\mathbb{R}}^{\mathcal{D}}, \mathcal{P}_{\mathbb{R}}^{\mathcal{D}})$ (Tarski-Domäne) und $(\Delta_{\mathbb{S}}^{\mathcal{D}}, \mathcal{P}_{\mathbb{S}}^{\mathcal{D}})$ (String-Domäne). Prädikate wie $\geq_{18}$ sind unäre Prädikate der Tarski-Domäne (also $\geq_{18} \in \mathcal{P}_{\mathbb{R}}^{\mathcal{D}}$).

Eine „konkrete Familie“ repräsentieren wir in der ABox in Abb. 2.3: Bei der ersten Gruppe von Axiomen handelt es sich um Konzept-Assertionen, dann kommen die Rollen-Assertionen, schließlich die Attributs-Assertionen, und letztlich eine Gruppe von Prädikats-Assertionen: $\{\text{michael\_age}, \text{michael\_name}, \text{birgit\_age}\} \subseteq \mathcal{N}_{\mathcal{CD}}$, $\text{=}'\text{'Michael Wessel}'\text{'}' \in \mathcal{P}_{\mathbb{S}}^{\mathcal{D}}$, und $\text{=}_{35} \in \mathcal{P}_{\mathbb{R}}^{\mathcal{D}}$. Dabei gilt $x \in (\text{=}'\text{'Michael Wessel}'\text{'})^{\mathcal{I}}$ gdw. $x = \text{'\text{'Michael Wessel}'\text{'}}$, und analog für =_{35} und 35.

2.2.5.1. Die Standard-Inferenzen am Beispiel

- Das Konzept *special_mother* ist *unerfüllbar* bzgl. der TBox.
- Das Konzept *mother* *subsumiert* das Konzept *grandmother* ($\text{grandmother} \sqsubseteq_{\mathcal{T}} \text{mother}$).
- Das Konzept *mammal* ist *äquivalent* zu $\top$.
- Die Konzepte *woman* und *man* sind *disjunkt* ($\text{woman} \sqcap \text{man} \sqsubseteq_{\mathcal{T}} \perp$).
- Die TBox ist *inkohärent*, da *special_mother* synonym zu $\perp$ ist.
- Die TBox ist *erfüllbar* (allerdings muss in allen Modellen $\text{special\_mother}^{\mathcal{I}} = \emptyset$ gelten).
- Die *Taxonomie* dieser TBox ist in Abb. 2.4 zu sehen. Ein Knoten in diesem GAG stellt eine Menge von zueinander synonymen Konzeptnamen dar.

Das Konzept *mammal* wurde als synonym zu $\top$ erkannt, und *special_mother* als synonym zu $\perp$.

Die transitive reflexive Hülle der dargestellten Kanten-Relation stellt *alle* zwischen Konzeptnamen geltenden Subsumptionsbeziehungen dar, nicht nur die syntaktisch (anhand von TBox-Axiomen) expliziten.

- Die ABox bzw. Wissensbasis ist *erfüllbar*.

		$attribute(age)$
		$attribute(name)$
		$transitive(has\_descendant)$
has_parent	$\doteq$	has_child^{-1}
$has\_ancestor$	$\doteq$	$has\_descendant^{-1}$
has_child	$\sqsubseteq$	$has\_descendant$
has_son	$\sqsubseteq$	has_child
$has\_daughter$	$\sqsubseteq$	has_child
$\top$	$\sqsubseteq$	$\forall has\_child.parent$
$\exists has\_child.\top$	$\sqsubseteq$	$person$
$\top$	$\sqsubseteq$	$\forall has\_son.male\_person$
$\top$	$\sqsubseteq$	$\forall has\_daughter.female\_person$
$\top$	$\sqsubseteq$	$mammal \sqcap (male \sqcup female)$
$male$	$\doteq$	$\neg female$
$person$	$\sqsubseteq$	$\exists (age).\top_{\mathbb{R}} \sqcap$ $\exists (name).\top_{\mathbb{S}} \sqcap$ $=_1 has\_parent.female\_person \sqcap$ $=_1 has\_parent.male\_person \sqcap$ $=_2 has\_parent.person$
$male\_person$	$\doteq$	$person \sqcap male$
$female\_person$	$\doteq$	$person \sqcap female$
$adult$	$\doteq$	$person \sqcap \exists (age). \geq_{18}$
man	$\doteq$	$male\_person \sqcap adult$
$woman$	$\doteq$	$female\_person \sqcap adult$
$parent$	$\doteq$	$person \sqcap \exists has\_child.\top$
$mother$	$\doteq$	$parent \sqcap female\_person$
$father$	$\doteq$	$parent \sqcap male\_person$
$mother\_of\_son$	$\doteq$	$mother \sqcap \exists has\_son.\top$
$mother\_of\_daughter$	$\doteq$	$mother \sqcap \exists has\_daughter.\top$
$mother\_having\_only\_daughters$	$\doteq$	$mother \sqcap \exists has\_daughter.\top \sqcap$ $\forall has\_child.female\_person$
$grandmother$	$\doteq$	$woman \sqcap \exists has\_child.parent$
$special\_mother$	$\doteq$	$mother\_of\_son \sqcap mother\_having\_only\_daughters$

Abbildung 2.2.: Eine $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ -Beispiel-Wissensbasis: Die TBox.

- Die Antwort des *Instanzen-Prüfungsproblems* für *margrit* und *grandmother* ist TRUE; d.h., *margrit* ist Instanz von *grandmother*.

Diese Inferenz kann wie folgt begründet werden. Gegeben ist *margrit* : *woman*. Aufgrund von $(birgit, mara) : has\_daughter$ und $birgit : person \sqcap (=2$

2.2 Beschreibungslogiken

```

{  margrit : woman,
   rudolf : man,
   birgit : person  $\sqcap$  ( $=_2$  has_child),

   (margrit, birgit) : has_daughter,
   (margrit, michael) : has_son,
   (rudolf, birgit) : has_daughter,
   (rudolf, michael) : has_son,
   (birgit, mara) : has_daughter,
   (birgit, alina) : has_daughter,

   (michael, michael_age) : age,
   (michael, michael_name) : name,
   (birgit, birgit_age) : age,

   (michael_name) : = 'Michael Wessel',
   (michael_age) : =35,
   (birgit_age, michael_age) : > }

```

Abbildung 2.3.: Eine $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ -Beispiel-Wissensbasis: Die ABox.

has_child) folgt nun *birgit* : *parent* aufgrund von $\text{parent} \sqsubseteq \text{person} \sqcap \exists \text{has\_child}.\top$, da $(\text{birgit}, \text{mara}) : \text{has\_child}$ aufgrund von $(\text{birgit}, \text{mara}) : \text{has\_daughter}$ und $\text{has\_daughter} \sqsubseteq \text{has\_child}$. Somit folgt *margrit* : *grandmother*, denn $\text{grandmother} \sqsubseteq \text{woman} \sqcap \exists \text{has\_child}.\text{parent}$ und $(\text{margrit}, \text{birgit}) : \text{has\_daughter}$ und somit auch $(\text{margrit}, \text{birgit}) : \text{has\_child}$.

- Die Antwort des Rollenfüller-Prüfungsproblems für *rudolf*, *alina* und *has_descendant* ist TRUE.

Die Inferenz kann wie folgt begründet werden. Da $(\text{rudolf}, \text{birgit}) : \text{has\_daughter}$, $\text{has\_daughter} \sqsubseteq \text{has\_child}$ und $\text{has\_child} \sqsubseteq \text{has\_descendant}$, ist $(\text{rudolf}, \text{birgit}) : \text{has\_descendant}$ eine logische Konsequenz, ebenso $(\text{birgit}, \text{alina}) : \text{has\_descendant}$. Da *has_descendant* als transitiv abgeschlossen deklariert wurde (mittels $\text{transitive}(\text{has\_descendant})$) folgt somit $(\text{rudolf}, \text{alina}) : \text{has\_descendant}$.

- Die Antwort des Instanzen-Retrieval-Problems für *woman* ist die Menge $\{\text{margrit}, \text{birgit}\}$.

Das *birgit* : *woman* folgt, kann wie folgt gesehen werden. Aufgrund der Bildrestriktion der Rolle *has_daughter* muss für *birgit* : *female-person* gelten, da $(\text{margrit}, \text{birgit}) : \text{has\_daughter}$. Die Definition von *woman* ist jedoch $\text{woman} \sqsubseteq \text{female\_person} \sqcap \text{adult}$, wobei $\text{adult} \sqsubseteq \text{person} \sqcap \exists(\text{age}). \geq_{18}$. Um

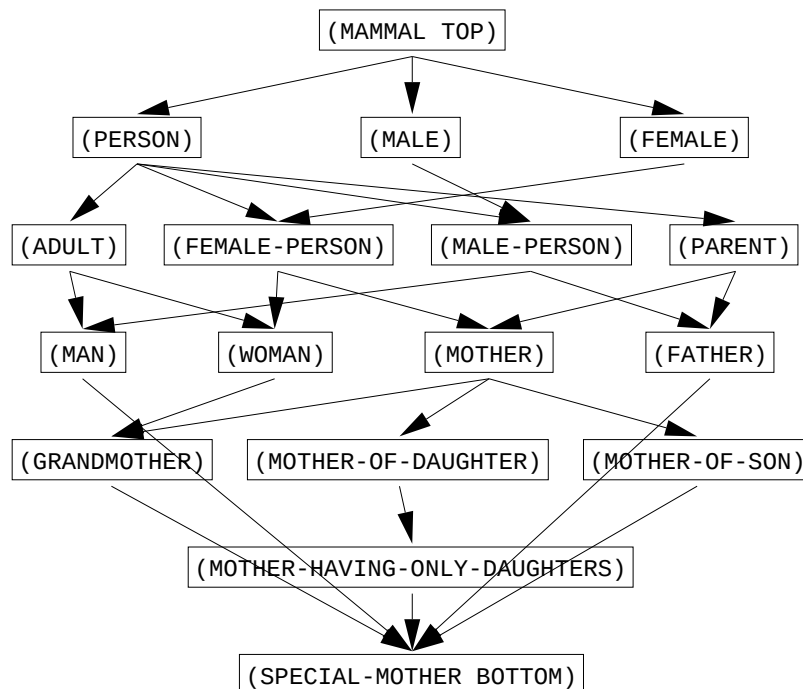


Abbildung 2.4.: Taxonomie der TBox.

$\exists(\text{age}). \geq_{18}$ für *birgit* zu zeigen, müssen nun die Prädikats-Assertionen in der konkreten Domäne betrachtet werden. Das Alter von *michael* ist durch $(\text{michael\_age}) : =_{35}$ bekannt ist, und somit gilt *michael* : *adult*. Da nun aber auch $(\text{birgit\_age}, \text{michael\_age}) : >$ vorliegt, muss offensichtlich auch *birgit* : *adult* gelten.

- Die Antwort des *Rollenfüller-Retrieval-Problems* für *margrit* und *has_descendant* liefert die Menge $\{\text{michael}, \text{birgit}, \text{mara}, \text{alina}\}$.
- Die Antwort des *Rollenpaare-Retrieval-Problems* für *has_parent* liefert die Menge $\{(\text{michael}, \text{margrit}), (\text{birgit}, \text{margrit}), (\text{alina}, \text{birgit}), (\text{mara}, \text{birgit})\}$.

2.2.6. Beschreibungslogiken und Modallogiken

DLs und Modallogiken sind eng verwandt. Schild zeigte in [Schild 1991], dass *ALC* letztlich nur eine andere Syntax für die Multi-Modallogik K_n (mit n Modalitäten) bereitstellt, also lt. Schild eine „syntaktische Variante“ von K_n ist. In der Folge konnten eine ganze Reihe von Komplexitätsergebnissen und Techniken aus der Modallogik-Forschung übernommen werden. Umgekehrt konnten jedoch auch Erkenntnisse aus der DL-Forschung in die Modallogik-Forschung transferiert werden (z.B. bzgl. ABoxen).

2.2 Beschreibungslogiken

Sei Φ eine Menge von Symbolen, die sog. *Propositionen* (wie in der Aussagenlogik), und $\mathcal{M}$ eine Menge von sog. *Modalitäten* mit $|\mathcal{M}| = n$. Die Menge der Formeln der Logik K_n bzw. die Logik K_n selbst (als Formelmengende betrachtet) ist induktiv wie folgt definiert:

- Jede Proposition $p \in \Phi$ ist eine K_n -Formel.
- Wenn Φ, Φ_1, Φ_2 K_n -Formeln sind, und $R \in \mathcal{M}$, dann sind auch $\neg\Phi, \Phi_1 \wedge \Phi_2, \Diamond_R \Phi, \Box_R \Phi$ K_n -Formeln.

Die Basis-Beobachtung von Schild war, dass eine $\mathcal{ALC}$ -Rolle $R \in \mathcal{N}_R$ einer der n Modalitäten $R \in \mathcal{M}$ in K_n entspricht, und dass $\mathcal{N}_C = \Phi$ ist. In [Schild 1991] wird eine Übersetzungsfunktion ϕ von $\mathcal{ALC}$ nach K_n wie folgt angegeben:

- $\phi(CN) =_{def} CN$ für $CN \in \mathcal{N}_C$,
- $\phi(\neg C) =_{def} \neg\phi(C)$,
- $\phi(C \sqcap D) =_{def} \phi(C) \wedge \phi(D)$,
- $\phi(\exists R.C) =_{def} \Diamond_R \phi(C)$,
- $\phi(\forall R.C) =_{def} \Box_R \phi(C)$.

Mit dieser Funktion ist z.B. $\phi(\exists R.C \sqcap \forall R.\exists S.C) = \Diamond_R C \wedge \Box_R \Diamond_S C$.

Zwischen C und $\phi(C)$ gilt eine Erfüllbarkeitsäquivalenz. Für Modallogiken werden in der Literatur mehrere Semantiken definiert. Betrachten wir die sog. *Kripke-Semantik* für K_n , so können die Modelle von $\phi(C)$ durch triviale Abbildungen in $\mathcal{ALC}$ -Modelle von C überführt werden, und andersrum (die Modelle können in gewisser Weise sogar als zueinander isomorph angesehen werden). Die Kripke-Semantik-Definition, die man in Standard-Modallogikbüchern wie [Hughes u. Cresswell 1968, Blackburn u. a. 2001] findet, unterscheidet sich von der hier gegebenen Definition nur unwesentlich:

Definition 45 (Kripke-Semantik, (relativierte) Erfüllbarkeit) Ein *Kripke-Rahmen* (*Kripke Frame*) für K_n mit $\mathcal{M} = \{R_1, \dots, R_n\}$ ist eine Tarski-Interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, sodass $R_i^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ für alle $i \in 1 \dots n$ definiert ist. Die Relation $R_i^{\mathcal{I}}$ wird auch *Erreichbarkeitsrelation* genannt, und die Elemente in $\Delta^{\mathcal{I}}$ *Punkte* oder *Zustände*.

Eine *Kripke-Interpretation* basiert auf einem *Kripke-Rahmen* und erweitert diesen so, dass $CN_i^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ für alle (relevanten bzw. benötigten) $CN_i \in \Phi$ definiert ist.

Erfülltsein einer Formel F in einer Kripke-Interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ an einem Punkt $i \in \Delta^{\mathcal{I}}$, geschrieben $((\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}), i) \models F$, ist wie folgt definiert:

- $((\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}), i) \models CN$ gdw. $i \in CN^{\mathcal{I}}$
- $((\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}), i) \models \neg\Phi$ gdw. $((\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}), i) \not\models \Phi$.

- $((\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}), i) \models \Phi_1 \wedge \Phi_2$ gdw. $((\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}), i) \models \Phi_1$ und $((\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}), i) \models \Phi_2$.
- $((\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}), i) \models \Diamond_R \Phi$ gdw. es ein $(i, j) \in R^{\mathcal{I}}$ gibt, sodass $((\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}), j) \models \Phi$.
- $((\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}), i) \models \Box_R \Phi$ gdw. für alle j mit $(i, j) \in R^{\mathcal{I}}$ gilt: $((\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}), j) \models \Phi$.

Eine K_n -Formel F ist erfüllbar wenn es eine Kripke-Interpretation gibt und einen Punkt $i \in \Delta^{\mathcal{I}}$, sodass $((\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}), i) \models F$.

Oftmals ist man nicht an beliebigen Kripke-Interpretationen interessiert, sondern nur an solchen, deren Kripke-Rahmen bzw. deren Erreichbarkeitsrelationen bestimmte Bedingungen erfüllen. Diese werden *Rahmen-Bedingungen* (*Frame Conditions*) genannt. Eine Menge von Kripke-Rahmen, die bestimmte Rahmen-Bedingungen erfüllen, wird eine *Kripke-Klasse* genannt.

Das *relativierte Erfüllbarkeitsproblem* fragt daher nach einer Kripke-Interpretation, die auf einem Rahmen basiert, der in einer bestimmten Klasse von Kripke-Rahmen enthalten ist.

Eine Formel ist *gültig* in einer Klasse von Rahmen wenn sie erfüllt ist in jeder Interpretation, die auf einem Rahmen dieser Klasse basiert. ■

Ist nun $\mathcal{I}$ ein DL-Modell von C mit $i \in C^{\mathcal{I}}$, so gilt offensichtlich auch $((\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}), i) \models \phi(C)$, und andersrum.

So folgt beispielsweise aus $i \in (D \sqcap E)^{\mathcal{I}}$ mit $D, E \in \mathcal{N}_C$ für $\Theta(D \sqcap E) = D \wedge E$ auch $((\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}), i) \models D \wedge E$, denn offensichtlich gilt $((\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}), i) \models D$ und $((\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}), i) \models E$ aufgrund von $i \in D^{\mathcal{I}}$ und $i \in E^{\mathcal{I}}$, da $\mathcal{I}$ eine DL-Interpretation ist. Gilt umgekehrt nun $((\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}), i) \models D \wedge E$, so muss $i \in D^{\mathcal{I}}$ und $i \in E^{\mathcal{I}}$ gelten. Per induktiver Erweiterung wird nun eine DL-Interpretation konstruiert: $(D \sqcap E)^{\mathcal{I}} =_{\text{def}} D^{\mathcal{I}} \cap E^{\mathcal{I}}$. Es ist offensichtlich, dass dann auch $i \in (D \sqcap E)^{\mathcal{I}}$ gilt. Mit den so definierten Modellen liegt also bereits eine Äquivalenz von C und $\phi(C)$ vor, und nicht nur eine Erfüllbarkeitsäquivalenz (der Beweis für beliebige Konzepte C erfordert nur eine einfache strukturelle Induktion über den Aufbau von C bzw. $\phi(C)$).

Diese Übersetzung funktioniert zunächst nur für Konzepte. Modallogiken kennen kein direktes Äquivalent zu TBoxen oder ABoxen. Diese müssen daher mit anderen modallogischen Mitteln (erfüllbarkeitsäquivalent) „simuliert“ werden. Einfache TBoxen können natürlich auch hier aufgefaltet werden; in ausdrucksstarken DLs bzw. Modallogiken können Techniken wie *TBox-Internalisierung* verwendet werden. ABoxen können in *hybriden Modallogiken* (diese stellen das modallogische Äquivalent der Nominalen bereit) repräsentiert werden.

Die Rahmen-Bedingungen sind eng verwandt mit den Eigenschaften, die für bzw. zwischen Rollen (durch Rollenaxiome) deklariert werden können. Ist R eine transitive Rolle und C ein $\mathcal{ALC}_{\mathcal{R}^+}$ -Konzept, so sind für $\phi(C)$ nun nur noch solche Modelle zulässig, in denen die Erreichbarkeitsrelation $R^{\mathcal{I}}$ transitiv abgeschlossen ist. Das Erfüllbarkeitsproblem wird daher relativiert auf die Kripke-

2.2 Beschreibungslogiken

Klasse der bzgl. R transitiv abgeschlossenen Kripke-Rahmen.

Eine Modallogik wird auch als eine *Menge von in einer Kripke-Klasse gültigen Formeln* angesehen [Blackburn u. a. 2001]; insofern handelt es sich um die Theorie einer Struktur (s. Kap. 2.1.1.7). Die Modallogik K_n entspricht der Menge der auf *beliebigen* Kripke-Rahmen gültigen Formeln (also wird die Kripke-Klasse aller Rahmen zugelassen). Die Logik $K4$ entspricht der Menge der in der Kripke-Klasse mit transitiv abgeschlossener Erreichbarkeitsrelation gültigen Formeln. $K4$ entspricht daher $\mathcal{ALC}_{\mathcal{R}^+}$ mit einer einzigen transitiven Rolle R (und keinen weiteren Rollen); da die Logik $S4 = TK4$ mit *reflexiver* (T) und *transitiver* ($K4$) Erreichbarkeitsrelation jedoch sehr prominent ist, wurde $\mathcal{ALC}_{\mathcal{R}^+}$ wohl S genannt.

Modallogisch lassen sich die Rahmen-Bedingungen durch sog. *Rahmen-Axiome* (*Frame Axioms*) erzwingen. Offensichtlich wird kein Rahmen-Axiom für K benötigt (da ja alle Kripke-Rahmen zulässig sind). Transitiv abgeschlossene Erreichbarkeitsrelationen und somit $K4$ wird durch das Axiom namens 4 erzwungen: $\Box p \rightarrow \Box \Box p$ (bzw. dual: $\Diamond \Diamond p \rightarrow \Diamond p$). Viele Kripke-Klassen lassen sich so axiomatisieren. Es gibt jedoch auch Klassen, für die keine Axiome existieren – so sind z.B. weder Rollendisjunktheit noch Rollenirreflexivität modallogisch axiomatisierbar. Auch die diskutierten Standard-DLs ermöglichen dies nicht.

Modallogiken sind als *robust entscheidbar* bekannt [Grädel 1999]. Die Ursache hierin wird in [Grädel 1999] durch die im Vergleich zur Prädikatenlogik eingeschränkten Quantoren und in der Baummodell-Eigenschaft gesehen. Die gleiche Beobachtung kann für DLs gemacht werden.

Einige bekannte Modallogiken sind:

1. $\mathcal{ALC}_{\mathcal{R}^+}$ entspricht der Vereinigung von K_n und $K4_n$, wobei es sich um die multimodalen Varianten von K und $K4$ handelt; mehrere Modalitäten werden benötigt, da $\mathcal{ALC}_{\mathcal{R}^+}$ natürlich mehrere (transitive und nichttransitive) Rollen erlaubt [Sattler 1996b].
2. $\mathcal{ALC}_{\mathcal{REG}}$ ist eine syntaktische Variante von *Propositional Dynamic Logic* (*PDL*) [Harel 1984, Harel u. a. 2000].
3. $\mathcal{ALC}\mathcal{I}_{\mathcal{REG}}$ ist eine syntaktische Variante von *Converse PDL* (*CPDL*) [Schild 1991].
4. *Deterministic PDL* (*DPDL*) ermöglicht *funktionale Erreichbarkeitsrelationen*. $\mathcal{ALC}f\mathcal{I}_{\mathcal{REG}}$ ist eine syntaktische Variante von *DCPDL*. Diese Sprache ist EXPTIME-vollständig [Vardi u. Wolper 1986].

2.2.7. Inferenzverfahren und -beweiser für Beschreibungslogiken

Inferenzverfahren für DLs sind Algorithmen bzw. mathematische Spezifikationen zur Lösung der Standard-Inferenzprobleme. Wie in Kap. 2.2.2.5 dargestellt, handelt es sich meist um *Entscheidungsverfahren*. Unter einem *Beweiser* verstehen wir im Folgenden eine Implementierung eines solchen Algorithmus’.

Da viele Inferenzprobleme wechselseitig aufeinander reduziert werden können, implementieren die meisten DL-Systeme lediglich einen Kern-Inferenzalgorithmus, z.B. einen ABox-Erfüllbarkeitstester im Falle eines ABox-DL-Systems. Heutzutage sind jedoch auch optimierte Spezial-Verfahren zur Lösung bestimmter Inferenzprobleme bekannt, z.B. für das Taxonomie-Berechnungsproblem oder das Instanzen-Retrieval-Problem [Haarslev u. Möller 2004].

Zwei *prominente Klassen* von DL-Inferenzverfahren sind durch die *Klasse der strukturellen Verfahren* und die *Klasse der Tableau-Verfahren* gegeben. Wir werden beide erläutern. Neuerdings werden jedoch auch Beweisverfahren aus der Logikprogrammierung, z.B. Reduktion auf disjunktives Datalog und Verwendung von Resolutionsverfahren [Hustadt u. a. 2004], sowie Beweiser für $\mathcal{L}_{FOP\mathcal{L}}$ wie VAMPIRE [Riazanov u. Voronkov 2002] in [Tsarkov u. a. 2004], oder gar Beweiser für monadische Prädikatenlogik zweiter Ordnung wie MONA [Henriksen u. a. 1995], s. [Karabaev u. Lutz 2004], für DLs untersucht.

Wir hatten in Kap. 2.1.1.7 bereits den Unterschied zwischen Folgerbarkeit und Beweis- bzw. Ableitbarkeit sowie die Begriffe Vollständigkeit und Korrektheit dargestellt. Zur Verdeutlichung dieser Begriffe im DL-Kontext betrachten wir das Konzept-Subsumptionsproblem:

- Das Verfahren ist *korrekt*, wenn bei Ausgabe von $\text{subsumes?}(D, C) = \text{TRUE}$ auch tatsächlich $C \sqsubseteq D$ gilt. Gilt also (Kontraposition) $C \not\sqsubseteq D$, so ist $\text{subsumes?}(D, C) = \text{FALSE}$. Das heißt, TRUE-Antworten eines korrekten Verfahrens kann vertraut werden.
- Das Verfahren ist *vollständig*, wenn im Falle von $C \sqsubseteq D$ stets $\text{subsumes?}(D, C) = \text{TRUE}$ gilt. Gilt also (Kontraposition) $\text{subsumes?}(D, C) = \text{FALSE}$, so muss $C \not\sqsubseteq D$ gelten. Das heißt, FALSE-Antworten eines vollständigen Verfahrens kann vertraut werden.

Korrektheit und Vollständigkeit für die extensionalen Retrieval-Probleme definieren wir am Beispiel des Instanzen-Retrieval-Problems wie folgt (die Definition für andere Retrieval-Probleme ist hierzu analog):

- Das Verfahren ist *korrekt*, wenn es keine falschen Ergebnisse liefert, d.h. wenn

$$\text{concept\_instances}(C) \subseteq \{i \mid i \in \text{inds}(\mathcal{A}), (\mathcal{A}, T) \models i : C\}$$
 gilt.

2.2 Beschreibungslogiken

Ist also $i \in \text{concept\_instances}(C)$, so auch $i \in \{i \mid i \in \text{inds}(\mathcal{A}), (\mathcal{A}, \mathcal{T}) \models i : C\}$, d.h., positiven Antworten eines korrekten Verfahrens kann vertraut werden.

- Das Verfahren ist *vollständig*, wenn keine Ergebnisse fehlen, d.h. wenn $\{i \mid i \in \text{inds}(\mathcal{A}), (\mathcal{A}, \mathcal{T}) \models i : C\} \subseteq \text{concept\_instances}(C)$

gilt.

Gilt also $i \notin \text{concept\_instances}(C)$, so auch $i \notin \{i \mid i \in \text{inds}(\mathcal{A}), (\mathcal{A}, \mathcal{T}) \models i : C\}$, d.h., negativen Antworten eines vollständigen Verfahrens kann vertraut werden.

Korrektheit ist in der Regel einfacher zu erreichen als Vollständigkeit. Wir werden in dieser Arbeit ausschließlich korrekte Verfahren betrachten und wollen dies im Folgenden stets stillschweigend annehmen.

2.2.7.1. Strukturelle Verfahren

Die ersten DL-Systeme verwendeten sog. *strukturelle Verfahren* zur Entscheidung von Konzeptsubsumption. Zur Entscheidung, ob $C \sqsubseteq D$ gilt, werden C und D zunächst in eine gewisse Normalform (*Structural Subsumption Normal Form*) gebracht. Die beiden Normalformen werden dann miteinander verglichen, um Subsumption zu entscheiden. Oftmals werden graphbasierte Verfahren verwendet, um z.B. einen speziellen Graph-Homomorphismus nachzuweisen (das DL-System CLASSIC verwendet solche graphbasierten Algorithmen, [Borgida u. Patel-Schneider 1994]).

Strukturelle Verfahren sind typischerweise polynomiell, die Performanz ist dementsprechend hoch. Strukturelle Verfahren sind korrekt, oftmals jedoch nur für sehr einfache Logiken vollständig. Insbesondere versagen sie für DLs mit voller Negation ($\mathcal{C}$), Disjunktion ($\mathcal{U}$), oder qualifizierten existentiellen Rollenrestriktionen ($\mathcal{E}$); strukturelle Verfahren sind jedoch für DLs wie $\mathcal{ALN}$ (und Teilsprachen hiervon) korrekt und vollständig.

Heutzutage werden unvollständige strukturelle Verfahren als *Wächter (Guards)* für vollständige Verfahren zu Optimierungszwecken verwendet (den positiven Antworten dieser Verfahren kann ja vertraut werden; liefert das unvollständige Verfahren eine negative Antwort, muss hingegen das vollständige Verfahren verwendet bzw. „nachgeschaltet“ werden).

2.2.7.2. Tableauverfahren

Moderne DL-Systeme verwenden sog. *Tableau-Verfahren*. Hierbei handelt es sich um „regelbasierte“ Kalküle, die auch für ausdrucksstarke DLs vollständig

sind. Tableau-Verfahren zeichnen sich durch eine hohe Modularität, Flexibilität und Erweiterbarkeit aus – durch Hinzufügen neuer sog. *Tableau-Regeln* können sie ein breites Spektrum unterschiedlicher DLs abdecken (im Unterschied zu strukturellen Verfahren).

Das erste Tableau-Verfahren zur Entscheidung von Konzepterfüllbarkeit wurde in [Schmidt-Schauß u. Smolka 1991] für die DL $\mathcal{ALC}$ angegeben. Dieses Verfahren eignet sich (mit minimalen Modifikationen) auch zur Entscheidung von ABox-Erfüllbarkeit. In der Folge wurde deutlich, dass das Verfahren eine Neuentdeckung des bereits für die Modallogik K_n bekannten Tableau-Verfahrens war. Die alsdann bekannten Korrespondenzen zur Modallogik wurden von nun an genutzt, um Tableau-Verfahren für ausdrucksstärkere DLs als $\mathcal{ALC}$ zu entwickeln.

Da in dieser Arbeit in Kap. 6 ein Rahmenwerk zur Erstellung von Aufgaben- und DL-spezifischen Beweisern vorgestellt wird, und Tableau-Verfahren für viele dieser Beweiser die theoretische Basis darstellen, müssen wir – für unsere Zwecke ausreichend detailliert – die Theorie der Tableau-Verfahren vorstellen. Wir tun dies am Beispiel eines Tableau-Verfahrens zur Entscheidung des $\mathcal{ALC}$ -Konzepterfüllbarkeitsproblems nach [Baader u. Sattler 2000], das eine Reformulierung des in [Schmidt-Schauß u. Smolka 1991] angegebenen Verfahrens ist.

2.2.7.3. Das Basis-Tableau-Verfahren zur Entscheidung von Konzepterfüllbarkeit in $\mathcal{ALC}$

Ein Tableau-Verfahren zur Entscheidung der Erfüllbarkeit eines beliebigen $\mathcal{ALC}$ -Konzeptes C (ohne TBox!) weist die Erfüllbarkeit von C durch Konstruktion einer *endlichen Repräsentation eines Modells für C* nach, bzw. scheitert bei der Konstruktion einer solchen Modell-Repräsentation, wenn C unerfüllbar ist. Dabei dürfen natürlich keine Modell-Konstruktionsmöglichkeiten übersehen werden; andererseits wäre der Kalkül unvollständig (nur den positiven Antworten könnte dann getraut werden).

Im Falle von $\mathcal{ALC}$ ist diese endliche Repräsentation des Modells tatsächlich in gewisser Weise isomorph zum Modell selbst. Dies gilt jedoch nicht für ausdrucksstärkere DLs (insbesondere nicht für solche, die keine Endliche-Modell-Eigenschaft, s. S. 67, haben). Diese endliche Modell-Repräsentation wird als *Vervollständigung* oder als *abgeschlossenes Tableau* bezeichnet.

Ein *Tableau* ist gewissermaßen die Arbeitsdatenstruktur des Kalküls (oft auch als *Constraintsystem* bezeichnet) und somit namensgebend für diese Kalküle. Prinzipiell kann ein Tableau als eine ABox (bzw. ein knoten- und kantenbeschrifteter Graph) betrachtet werden. Im Vergleich zu einer ABox kann diese Datenstruktur jedoch noch weitere Arten von Assertionen

2.2 Beschreibungslogiken

enthalten (z.B. Ungleichheits-Assertionen). Für den hier diskutierten $\mathcal{ALC}$ -Konzepterfüllbarkeitstester kann ein Tableau jedoch als eine ABox begriffen werden.

Ein Tableau-Kalkül ist ein regelbasierter Kalkül. Die Erfüllbarkeit eines Konzeptes C wird durch Anwendung von *Tableau-Regeln* auf das für C konstruierte *initiale Tableau* $\{i : C\}$ entschieden (wir notieren dieses Tableau als ABox). Die Regeln werden nun auf das initiale Tableau erschöpfend angewendet, bis entweder eine Vervollständigung gefunden wurde (in diesem Falle sind keine Regeln mehr anwendbar), oder aber jede (anhand der *Regelanwendungsstrategie* mögliche) Regelanwendungsreihenfolge einen *elementaren Widerspruch* im Tableau erzeugt (C ist dann unerfüllbar). Das Tableau ist dann *widersprüchlich* bzw. *kontradiktorisch*.

Eine widersprüchliche ABox bzw. ein widersprüchliches Tableau ist offensichtlich unerfüllbar:

Definition 46 (Widersprüchliche ABox, Elementarer Widerspruch) Eine ABox $\mathcal{A}$ heißt *widersprüchlich*, wenn sie einen sog. *elementaren Widerspruch* enthält: $\{x : C, x : \neg C\} \subseteq \mathcal{A}$ für ein beliebiges x und C .

Eine ABox ist eine *Vervollständigung*, wenn sie keinen Widerspruch enthält, und keine der Regeln aus Abb. 2.5 anwendbar sind. ■

Die Tableau-Regeln konstruieren jedoch nicht beliebige Vervollständigungen, sondern lediglich solche einer *bestimmten Klasse*. Wie erläutert, ist es für $\mathcal{ALC}$ ausreichend, lediglich *endliche Baummodelle* zu betrachten. Dementsprechend reflektieren auch die Vervollständigungen die Baumstruktur (tatsächlich kann die Baummodell-Eigenschaft von $\mathcal{ALC}$ auch als eine Konsequenz der Vollständigkeit des Tableau-Verfahrens begriffen werden).

Der Kalkül zur Entscheidung der Erfüllbarkeit eines $\mathcal{ALC}$ -Konzeptes arbeitet auf einer endlichen Menge von ABoxen $\{\mathcal{A}_1, \dots, \mathcal{A}_k\}$. Intuitiv ist diese Menge von ABoxen erfüllbar gdw. *eine* der $\mathcal{A}_i$ erfüllbar ist; die Menge repräsentiert daher eine Disjunktion. Die Menge repräsentiert die verschiedenen Vervollständigungsmöglichkeiten.

Definition 47 (Erfüllbarkeit einer Menge von ABoxen) Eine Menge von ABoxen $S = \{\mathcal{A}_1, \dots, \mathcal{A}_k\}$ ist *erfüllbar* gdw. eine der ABoxen $\mathcal{A}_i$ erfüllbar ist. ■

Der Kalkül startet nun mit der initialen Menge $S_0 =_{def} \{\{x : \text{NNF}(C)\}\}$. Aus Gründen der Einfachheit muss das Konzept hier in die sog. Negationsnormalform (NNF) mittels $\text{NNF}(C)$ gebracht werden. Die NNF von C kann durch folgende erschöpfende Regelanwendungen hergestellt werden: $\neg(C_1 \sqcap C_2) \rightarrow \neg C_1 \sqcup \neg C_2$,

$\sqcap$ -Regel:
wenn

1. $x : C_1 \sqcap C_2 \in \mathcal{A}$
2. $\{x : C_1, x : C_2\} \not\subseteq \mathcal{A}$

dann

$$\mathcal{A}' := \mathcal{A} \cup \{x : C_1, x : C_2\}$$

 $\sqcup$ -Regel:
wenn

1. $x : C_1 \sqcup C_2 \in \mathcal{A}$
2. $\{x : C_1, x : C_2\} \cap \mathcal{A} = \emptyset$

dann

$$\mathcal{A}' := \mathcal{A} \cup \{x : C_1\}$$

$$\mathcal{A}'' := \mathcal{A} \cup \{x : C_2\}$$

 $\exists$ -Regel:
wenn

1. $x : \exists R.C_1 \in \mathcal{A}$
2. es gibt kein y , sodass $\{y : C_1, (x, y) : R\} \subseteq \mathcal{A}$

dann

$$\mathcal{A}' := \mathcal{A} \cup \{y : C_1, (x, y) : R\}$$

 $\forall$ -Regel:
wenn

1. $\{x : \forall R.C_1, (x, y) : R\} \subseteq \mathcal{A}$
2. $y : C_1 \notin \mathcal{A}$

dann

$$\mathcal{A}' := \mathcal{A} \cup \{y : C_1\}$$

 Abbildung 2.5.: Tableau-Regeln für $\mathcal{ALC}$ aus [Baader u. Sattler 2000]

$\neg(C_1 \sqcup C_2) \rightarrow \neg C_1 \sqcap \neg C_2$, $\neg \forall R.C_1 \rightarrow \exists R.\neg C_1$, und $\neg \exists R.C_1 \rightarrow \forall R.\neg C_1$. Per struktureller Induktion kann leicht nachgewiesen werden: $\mathcal{I} \models C$ gdw. $\mathcal{I} \models \text{NNF}(C)$. Der Kalkül wendet auf $\mathcal{S}_0$ nun so lange die Regeln aus Abb. 2.5 an, bis entweder eine Vervollständigung konstruiert wurde, oder aber alle ABoxen in dieser Menge widersprüchlich sind. Durch Anwendung einer der Regeln aus Abb. 2.5 wird eine ABox $\mathcal{A}_i$ aus $\mathcal{S}_i = \{\mathcal{A}_1, \dots, \mathcal{A}_k\}$ entfernt und gegen eine ABox $\mathcal{A}'_i$ oder zwei ABoxen $\mathcal{A}'_i$ und $\mathcal{A}''_i$ ersetzt: $\mathcal{S}_{i+1} = (\mathcal{S}_i \setminus \{\mathcal{A}_i\}) \cup \{\mathcal{A}'_i\}$ bzw. $\mathcal{S}_{i+1} = (\mathcal{S}_i \setminus \{\mathcal{A}_i\}) \cup \{\mathcal{A}'_i, \mathcal{A}''_i\}$. Wir schreiben dann $\mathcal{S}_i \rightarrow \mathcal{S}_{i+1}$. Die transitive Hülle von $\rightarrow$ wird mit $\stackrel{+}{\rightarrow}$ notiert.

Für den folgenden Algorithmus gilt nun $\text{concept\_satisfiable?}(C) = \text{TRUE}$ gdw. C erfüllbar ist:

Definition 48 ($\mathcal{ALC}$ -Erfüllbarkeitstester, $\text{concept\_satisfiable?}$) Sei C ein $\mathcal{ALC}$ -Konzept. Der Erfüllbarkeitstester $\text{concept\_satisfiable?}(C)$ ist dann wie folgt definiert:

$i := 0$

$\mathcal{S}_0 := \{\{x : \text{NNF}(C)\}\}$

while $\mathcal{S}_i$ enthält keine Vervollständigung, und es gibt eine nichtwidersprüchliche ABox in $\mathcal{S}_i$, auf die eine Regel anwendbar ist:

Wähle ein $\mathcal{A}_i \in \mathcal{S}_i$ und eine auf $\mathcal{A}_i$ anwendbare Regel R (Abb. 2.5).

Ist $\text{apply}(R, \mathcal{A}_i) = \{\mathcal{A}'_i\}$ das Resultat der Regelanwendung, dann setze $\mathcal{S}_{i+1} := (\mathcal{S}_i \setminus \{\mathcal{A}_i\}) \cup \{\mathcal{A}'_i\}$.

Ist $\text{apply}(R, \mathcal{A}_i) = \{\mathcal{A}'_i, \mathcal{A}''_i\}$ das Resultat der Regelanwendung, dann setze $\mathcal{S}_{i+1} := (\mathcal{S}_i \setminus \{\mathcal{A}_i\}) \cup \{\mathcal{A}'_i, \mathcal{A}''_i\}$.

2.2 Beschreibungslogiken

end while

return:

FALSE, wenn alle ABoxen in $\mathcal{S}_i$ widersprüchlich sind,

TRUE sonst. ■

Bevor wir Terminierung, Korrektheit und Vollständigkeit *des Verfahrens* nachweisen, weisen wir eine sehr wichtige Eigenschaft *der Tableau-Regeln* nach, die sog. *lokale Korrektheit*. Diese ist wesentlich zum Nachweis der Korrektheit und Vollständigkeit *des Verfahrens*.

Lemma 2 (Lokale Korrektheit der $\mathcal{ALC}$ -Regeln) Gilt $\mathcal{S} \rightarrow \mathcal{S}'$ durch Regelanwendung, dann ist $\mathcal{S}$ (un)erfüllbar gdw. $\mathcal{S}'$ (un)erfüllbar ist. ■

Lokale Korrektheit besagt also, dass die Erfüllbarkeit von $\mathcal{S}$ durch Regelanwendungen nicht beeinträchtigt wird (man beachte, dass $\mathcal{S}$ lt. Def. 47 erfüllbar ist gdw. eine der ABoxen in $\mathcal{S}$ erfüllbar ist). Dies wird nun für jede Regel „lokal“ nachgewiesen. Für die $\sqcup$ -Regel argumentiert man wie folgt: Sei $x : C_1 \sqcup C_2 \in \mathcal{A}_i$ für $\mathcal{A}_i \in \mathcal{S}$. Gilt nun $\mathcal{I} \models \mathcal{S}$, so ist $x^{\mathcal{I}} \in (C_1 \sqcup C_2)^{\mathcal{I}}$, d.h., $x^{\mathcal{I}} \in C_1^{\mathcal{I}}$ oder $x^{\mathcal{I}} \in C_2^{\mathcal{I}}$. Also gilt entweder $\mathcal{I} \models \mathcal{A}_i \cup \{x : C_1\}$ oder $\mathcal{I} \models \mathcal{A}_i \cup \{x : C_2\}$. Das heißt aber, dass $\mathcal{I} \models \mathcal{S}'$ gilt. Natürlich gilt diese Argumentation auch umgekehrt. Gilt hingegen $\mathcal{I} \not\models \mathcal{S}$, so erst recht $\mathcal{I} \not\models \mathcal{S}'$. Also erhält die $\sqcup$ -Regel die (Un)Erfüllbarkeit. Für die anderen Regeln argumentiert man analog.

Das folgende Lemma sagt nun aus, dass das Verfahren korrekt und vollständig ist, und immer terminiert (und somit ein Entscheidungsverfahren ist):

Lemma 3 Sei C ein beliebiges $\mathcal{ALC}$ -Konzept. Dann gilt:

1. $\text{concept\_satisfiable?}(C)$ *terminiert*. Es gibt keine unendliche Sequenz von Regelanwendungen $\mathcal{S}_0 \rightarrow \mathcal{S}_1 \rightarrow \mathcal{S}_2 \rightarrow \dots$.

Dies ist eine unmittelbare Konsequenz daraus, dass durch Regeln ausschließlich Subkonzepte des initialen Konzeptes $\text{NNF}(C)$ zu den ABoxen in $\mathcal{S}_i$ hinzugefügt werden, und daraus, dass die Anzahl der Rollen-Assertionen durch die maximale Schachtelungstiefe der $\exists R.C$ -Konzepte in $\text{subconcepts}(\text{NNF}(C))$ und somit durch $|\text{subconcepts}(\text{NNF}(C))|$ nach oben beschränkt ist. Außerdem werden niemals ABox-Assertionen durch Regeln entfernt.

2. $\text{concept\_satisfiable?}(C)$ ist *vollständig*.

Die Definition der *Vollständigkeit* besagt: Ist C erfüllbar, so muss der Algorithmus $\text{concept\_satisfiable?}(C) = \text{TRUE}$ liefern. Durch Kontraposition erhalten wir: Gilt $\text{concept\_satisfiable?}(C) = \text{FALSE}$, so muss C unerfüllbar sein (FALSE-Antworten eines vollständigen Verfahrens kann vertraut werden).

Offensichtlich liefert der Algorithmus nach Def. 48 genau dann $\text{concept\_satisfiable?}(C) = \text{FALSE}$, wenn alle ABoxen in $\mathcal{S}_i$ widersprüchlich sind. Eine widersprüchliche ABox ist trivialerweise unerfüllbar; daher ist auch $\mathcal{S}_i$ unerfüllbar (lt. Definition 47). Da $\mathcal{S}_i$ durch Regelanwendung aus $\mathcal{S}_0 =_{\text{def}} \{\{x : \text{NNF}(C)\}\}$ hervorgegangen ist, $\mathcal{S}_0 \xrightarrow{+} \mathcal{S}_i$, und die Regeln (wie soeben argumentiert) *lokal korrekt* sind, muss bereits $\mathcal{S}_0$ und somit $\text{NNF}(C)$ und daher auch C unerfüllbar sein. Hiermit ist Vollständigkeit bewiesen.

Da C beliebig ist, ist $\text{concept\_satisfiable?}$ vollständig für $\mathcal{ALC}$.

3. $\text{concept\_satisfiable?}(C)$ ist *korrekt*.

Die Definition der *Korrektheit* besagt: Liefert der Algorithmus $\text{concept\_satisfiable?}(C) = \text{TRUE}$, so muss C erfüllbar sein (TRUE-Antworten eines korrekten Algorithmus' kann vertraut werden).

Der Algorithmus liefert lt. Definition genau dann TRUE, wenn $\mathcal{S}_i$ mit $\mathcal{S}_0 \xrightarrow{+} \mathcal{S}_i$ eine ABox $\mathcal{A}$ enthält, $\mathcal{A} \in \mathcal{S}_i$, die eine Vervollständigung ist. Die ABox $\mathcal{A}$ enthält also keinen Widerspruch, und keine der Regeln ist auf $\mathcal{A}$ anwendbar. Zum Nachweis der Erfüllbarkeit von $\mathcal{S}_i$ zeigen wir, dass $\mathcal{A}$ erfüllbar ist, indem wir ein Modell $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ für $\mathcal{A}$ wie folgt konstruieren:

- $\Delta^{\mathcal{I}} =_{\text{def}} \text{inds}(\mathcal{A})$
- $C^{\mathcal{I}} =_{\text{def}} \{x \mid x : C \in \mathcal{A}\}$, für $C \in \mathcal{N}_{\mathcal{C}}$
- $R^{\mathcal{I}} =_{\text{def}} \{(x, y) \mid (x, y) : R \in \mathcal{A}\}$, für $R \in \mathcal{N}_{\mathcal{R}}$.

Der Nachweis von $\mathcal{I} \models \mathcal{A}$ ist trivial. Das konstruierte Modell wird auch *kanonisches Modell* genannt. Offensichtlich hat es die Form eines Baums und ist endlich und erfüllt somit die Baummodell- als auch die Endliche-Modell-Eigenschaft.

Aus der nachgewiesenen Erfüllbarkeit von $\mathcal{A}$ folgt nun die Erfüllbarkeit von $\mathcal{S}_i$ und aufgrund der lokalen Korrektheit der Regeln die Erfüllbarkeit von $\mathcal{S}_0$ und somit von $\text{NNF}(C)$ und C . Hiermit ist die Korrektheit bewiesen.

Da C beliebig ist, ist $\text{concept\_satisfiable?}$ korrekt für $\mathcal{ALC}$. ■

Man beachte, dass die Konstruktion eines Modells zum Nachweis der Korrektheit nur deswegen gelingt, weil die Regeln „ausreichend viele Assertionen“ zu $\mathcal{A}$ bzw. hinreichend viele ABoxen zu $\mathcal{S}_0$ im Laufe der Expansion hinzugefügt haben. Insbesondere ist z.B. sichergestellt, dass für Assertionen wie $i : \exists R.D \in \mathcal{A}$ ein R -Nachfolger j mit $\{(i, j) : R, j : D\} \subseteq \mathcal{A}$ existiert, dass Disjunktionen aufgelöst wurden (aus $i : C \sqcup D \in \mathcal{A}$ folgt $i : C \in \mathcal{A}$ oder $i : D \in \mathcal{A}$), etc. Würden die Regeln die ABoxen nicht ausreichend „ansättigen“, würde die Konstruktion eines Modells anhand der vorliegenden syntaktischen Information in Form von ABox-Assertionen nicht möglich sein (die syntaktisch vorliegende Information wäre nicht ausreichend, um ein Modell zu konstruieren).

2.2 Beschreibungslogiken

Das obige Tableau-Verfahren kann auch zur Entscheidung von $\mathcal{ALC}$ -ABox-Erfüllbarkeit (mit leerer TBox bzw. bzgl. einfacher TBoxen nach vorheriger Auf-faltung) verwendet werden; hier sind nur minimale Modifikationen notwendig (alle Konzept-Assertionen müssen in NNF sein).

Das $\mathcal{ALC}$ -Tableau-Verfahren kann für ausdrucksstärkere DLs angepasst und erweitert werden. Hierfür werden für die zusätzlichen Konstruktoren entsprechende Regeln definiert, welche die Semantik der entspr. Konstruktoren reflektieren. Zudem können noch weitere Arten von elementaren Widersprüchen (Clashes) auftreten, die erkannt werden müssen; Def. 46 muss dann also verfeinert werden.

Zum Beweis der Terminierung, Vollständigkeit und Korrektheit werden prinzi-piell die gleichen „Beweismuster“ wie oben angewendet:

- Lokale Korrektheit der zusätzlichen Tableau-Regeln ist i.d.R. einfach zu beweisen.
- Im Unterschied zum $\mathcal{ALC}$ -Verfahren werden jedoch für komplexere DLs sog. *Regelanwendungsstrategien* benötigt, damit die Terminierung des Verfahrens sichergestellt ist.

Terminierung muss auch für DLs mit transitiven Rollen oder allgemeinen TBoxen durch besondere Techniken, sog. *Blockierungsverfahren*, sicher-gestellt werden.

Das klassische Beispiel ist das $\mathcal{ALC}_{\mathcal{R}^+}$ -Konzept $(\exists R.C) \sqcap (\forall R.\exists R.C)$, wel-ches für $R \in \mathcal{N}_{\mathcal{R}^+}$ (R ist also eine transitive Rolle) ohne entsprechende *Blockierungsbedingung* zu einer nichtterminierenden Regelanwendungskette führt [Sattler 1996b].

So führt die Expansion von

$$\mathcal{A}_0 = \{i : (\exists R.C) \sqcap (\forall R.\exists R.C)\}$$

zunächst zu

$$\mathcal{A}_1 = \mathcal{A}_0 \cup \{j : C, j : \exists R.C, (i, j) : R, j : \forall R.\exists R.C\},$$

da für $\mathcal{ALC}_{\mathcal{R}^+}$ eine spezialisierte $\forall$ -Regel zur Berücksichtigung der transitiv abgeschlossenen Rollen benötigt wird, die für $j : \forall R.\exists R.C \in \mathcal{A}_1$ verant-wortlich ist. Die weitere Expansion führt dann zu

$$\mathcal{A}_2 = \mathcal{A}_1 \cup \{k : C, k : \exists R.C, (j, k) : R, k : \forall R.\exists R.C\}.$$

Offensichtlich stoppt der Expansionsprozess nun nicht, wenn die Anwen-dung der $\exists$ -Regel nicht *blockiert* wird, da für $\mathcal{A}_2$ bzgl. k die gleichen Asser-tionen wie bereits zuvor für $\mathcal{A}_1$ bzgl. j erzeugt wurden.

- Vollständigkeit ist für einen *Erfüllbarkeitstester* einfach nachzuweisen; der obige Beweis kann ohne Änderungen übernommen werden.

- Korrektheit ist für einen *Erfüllbarkeitstester* i.d.R. schwieriger zu beweisen. Es muss gezeigt werden, wie aus einer Vervollständigung ein (kanonisches) Modell für $\mathcal{A}$ konstruiert werden kann. Dies wird insbesondere für DLs ohne Endliches-Modell-Eigenschaft erschwert, da hier eine Konstruktionsvorschrift angegeben werden muss, die aus einer endlichen Vervollständigung ein unendliches Modell konstruiert. Zur Konstruktion werden Techniken aus der Modallogik verwendet, das sog. *Abwickeln* (*Unraveling*). Die auch für die Terminierung benötigten Blockierungsbedingungen stellen letztlich sicher, dass das Abwickeln erfüllbarkeitserhaltend funktioniert.

Schließlich wollen wir noch darauf hinweisen, dass sich die Begriffe Korrektheit und Vollständigkeit (für entscheidbare DLs) in gewisser Weise dual zueinander verhalten: Betrachten wir nun statt einem Erfüllbarkeitstester `concept_satisfiable?` einen Unerfüllbarkeitstester `concept_unsatisfiable?`, so wird Korrektheit zu Vollständigkeit, und andersrum.

Korrektheit bedeutet dann: Ist `concept_unsatisfiable?(C) = TRUE`, so ist C unerfüllbar. Durch Kontraposition erhalten wir: Ist C erfüllbar, so liefert `concept_unsatisfiable?(C) = FALSE`. Dann liefert jedoch `concept_satisfiable?(C) = TRUE`. Somit kann der Vollständigkeitsbeweis für `concept_satisfiable?` als Korrektheitsbeweis für `concept_unsatisfiable?` gedeutet werden, und analog der Korrektheitsbeweis für `concept_satisfiable?` als Vollständigkeitsbeweis für `concept_unsatisfiable?`. Die verwendeten *Beweismuster* sind also identisch – der Korrektheitsbeweis wird zum Vollständigkeitsbeweis, und der Vollständigkeitsbeweis zum Korrektheitsbeweis.

2.2.8. Beschreibungslogiksysteme

Ein DL-System ist mehr als eine Implementierung eines Beweisers zur Entscheidung von Konzept-, ABox- oder Wissensbasiserfüllbarkeit. Ein DL-System hat viele Dimensionen bzw. unterschiedliche Facetten, die man zudem aus *unterschiedlichen Perspektiven* betrachten kann.

Verschiedene DL-Systeme unterscheiden sich zum einen bzgl. der unterstützten DL. Zum anderen gibt es Systeme mit und ohne ABox-Komponente. Die Art der unterstützten TBoxen kann variieren, die eventuell verfügbaren konkreten Domänen können variieren (prinzipiell sind auch benutzerdefinierbare konkrete Domänen denkbar), etc.

Die von den Systemen angebotenen *Dienste* variieren sowohl bzgl. der Quantität und Art der Dienste¹⁹ (funktionale Eigenschaften), als auch bzgl. der Qua-

¹⁹Beispielsweise wird ein DL-System ohne ABox-Komponente auch keine ABox-Dienste anbieten.

2.2 Beschreibungslogiken

lität der Dienste (Quality of Service); letztere Diensteigenschaften werden auch nichtfunktionale Eigenschaften genannt.²⁰

Die Dienste müssen sowohl von Menschen als auch von Programmen in Anspruch genommen werden können. Für beide Arten von Nutzern sind entsprechende *Schnittstellen* vorzusehen:

- Für den menschlichen Nutzer sind *grafische Nutzungsoberflächen (GUIs)* zum DL-System wünschenswert. Abb. 2.6 zeigt ein (hauptsächlich vom Autor entwickeltes) GUI für das RACERPRO DL-System namens RACERPORTER [Wessel u. Möller 2007a]. Ein solches GUI sollte zum einen Funktionalität zur Inspektion und Visualisierung des Zustands des DL-Systems bieten (z.B. die Taxonomie visualisieren können). Hier ist insbesondere auch semantische und syntaktische (*Ad Hoc*) *Anfragefunktionalität* wünschenswert, mit deren Hilfe die Wissensbasis befragt werden kann. Damit die erhaltenen Informationen bzw. Anfrageergebnisse als Eingabe für weitere Verarbeitungsschritte im GUI verwendet werden können, sind zudem Kommunikationskanäle bzw. entspr. gemeinsame Arbeitsdatenstrukturen notwendig. Diese müssen durch entspr. Metaphern dem Nutzer verständlich gemacht werden. Das in Abb. 2.6 dargestellte RACERPORTER-GUI verwendet für diesen Zweck eine *Blackboard-* oder *Clipboard-Metapher* [Wessel u. Möller 2007a]. Mit Hilfe des Clipboards können komplexe Wissensbasisinspektionsprozesse aus elementaren GUI-Interaktionen und semantischen und syntaktischen Anfragen komponiert, kohärent visualisiert und aufeinander bezogen werden. Der erste Aufgabenbereich eines solchen GUIs kann etwas verallgemeinernd als *Funktionalität zum Stöbern und Inspizieren (Browsing and Inspection)* bezeichnet werden.

Zum anderen ist auch Unterstützung bei der (grafischen und/oder textuellen) Erstellung von Wissensbasen, also Editier- bzw. Authoring-Funktionalität wünschenswert. RACERPORTER bietet hier sowohl eine Shell als auch einen Editor mit Evaluationsmöglichkeit.

Zudem ist auch eine *Erklärungsfunktionalität* wünschenswert. Diese muss natürlich zunächst vom DL-System realisiert werden (da dieses ja die Inferenzen zieht). Die Erklärungen sollten jedoch durch das GUI entspr. visualisiert und somit transparent gemacht werden. In den heutigen Systemen (mit Ausnahme des CLASSIC-Systems, s. [McGuinness 1996]) ist Erklärungsfunktionalität höchstens ansatzweise realisiert; die gelieferten Erklärungen sind vom menschlichen Nutzer oftmals nicht nachvollziehbar. Hier ist noch erhöhter Forschungsbedarf zu verzeichnen (sowohl zur Generierung als auch zur Visualisierung von Erklärungen).

- Die Nutzung des DL-Systems durch ein Programm (bzw. durch einen Cli-

²⁰Beispielsweise Performanz, Verfügbarkeit, ...

ent im *Client-Server-Modell*) setzt entsprechende *Dienst-Schnittstellen* bzw. ein *API (Application Program Interface)* voraus. Ist das DL-System als *Server* ausgelegt, so sind diese API-Funktionen aus Sicht des Clients letztlich entfernte Funktionen/Prozeduren/Methoden, die durch RPC-artige Aufrufe, WebServices o.ä. in Anspruch genommen werden müssen. Natürlich sind auch andere *lokale* Kopplungen denkbar (DL-System als lokale Komponente, Client-Kode in der selben Laufzeitumgebung wie das DL-System, etc.).

Für die Wissensbasen wird oftmals die durch den *KRSS-Standard (Knowledge Representation System Standard)* normierte Syntax verwendet [Patel-Schneider u. Swartout 1993]. Ein anderes populäres Wissensbasis-Standardformat ist *KIF (Knowledge Interchange Format)* [Genesereth u. Fikes 1992]. KRSS verwendet eine LISP-Syntax; heutzutage werden jedoch von einigen Anwendern XML-basierte Formate bevorzugt. KRSS definiert auch die Dienst-Schnittstelle bzw. das API; hier finden sich *Tell- und Ask-Funktionen*, die sich auf Konzepte, TBoxen, und ABoxen beziehen. Beispielsweise wird mittels (*concept-subsumes? (and C (some R D)) (or C E)*) die Entscheidung von $C \sqcup E \sqsubseteq C \sqcap \exists D.D$ vom DL-System angefordert.

Die alleinige Verwendung der LISP-Syntax wird heutzutage oftmals als nicht ausreichend betrachtet, da XML-basierte Formate eine vorherrschende Stellung erreicht haben (für die entspr. Rahmenwerke zur Verarbeitung mittels der Programmiersprache Java etc. existieren). Zudem ist die Interoperabilität mit anderen Systemen und Standards durch die Verwendung von XML besser gewährleistet. Eine moderne, XML/HTTP-basierte Form von KRSS ist durch das *DIG-Protokoll* [Bechhofer u. a. 2003] gegeben. DIG steht für *Description Logic Implementors Group*. Das DIG-Protokoll normiert ebenfalls eine Tell- und Ask-Schnittstelle. Voraussetzung ist, dass das DL-System als Server im Netzwerk in Anspruch genommen werden kann. Die aktuelle DIG-Version 1.1 leidet an einer unzureichenden Abdeckung der von aktuellen DL-Systemen angebotenen Dienste. Entsprechende Erweiterungsvorschläge (DIG 2.0) sind jedoch in Arbeit, doch noch nicht verabschiedet und implementiert [Turhan u. a. 2006].

Die neuste Generation der DL-Systeme unterstützt zudem die für das SEMANTIC WEB relevanten Sprachen OWL und RDFS, wobei es sich um W3C-Standards handelt; wir kommen in Kap. 3 hierauf zurück. Dementsprechend ist zu erwarten, dass das Interoperabilitätsproblem zukünftig stark entschärft wird, und Dienste mehr und mehr über DIG z.B. über Webservice-Schnittstellen in Anspruch genommen werden.

Die bisher diskutierten Aspekte beleuchten ein DL-System aus *externer Perspektive*. Natürlich können wir auch eine *interne Perspektive* einnehmen und Dimensionen wie verwendete Implementationssprache, Art des implementierten Beweisverfahrens etc. werden dann relevant.

2.2 Beschreibungslogiken

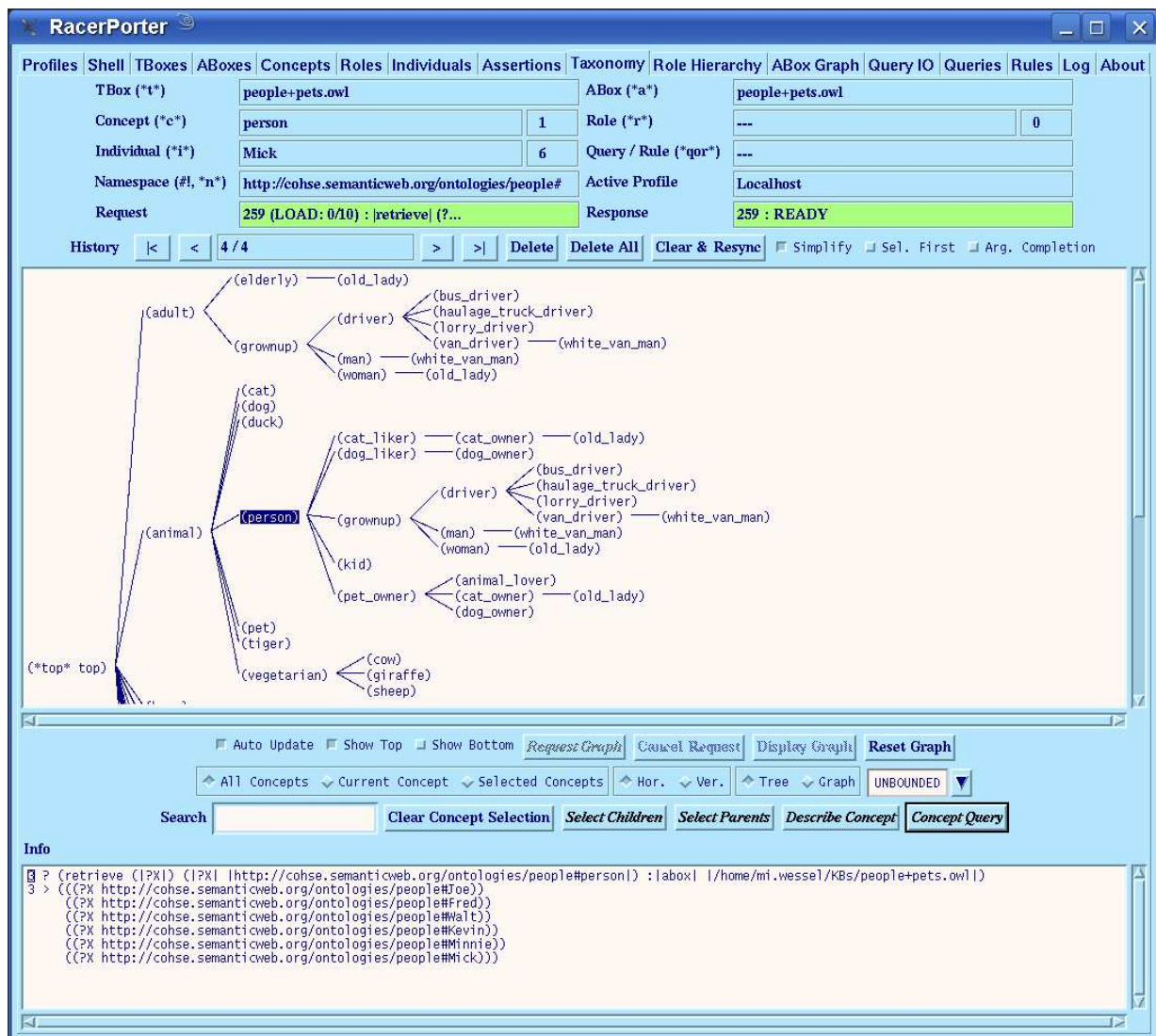


Abbildung 2.6.: RacerPorter – Ein (hauptsächlich vom Autor entwickeltes) GUI für RACERPRO.

Gemäß [Baader u. Sattler 2000] lässt sich die Geschichte der DL-Forschung und somit auch die Geschichte der DL-System-Entwicklung bis zum heutigen Tage in 4 Phasen einteilen; s. auch [Möller u. Haarslev 2003] und [Woods u. Schmolze 1992]:

- Phase 1 – Die Phase der ersten Implementationen. Die DL-Systeme in dieser Phase verwendeten strukturelle Verfahren. Die verwendeten Algorithmen waren zwar effizient (polynomiell), aber unvollständig. Systeme wie KL-ONE [Brachman u. Schmolze 1985] und LOOM [Brill 1994, MacGregor 1991] sind hier als einflussreich und prominent zu nennen.

- Phase 2 – In dieser Phase wurden erste Komplexitäts- und Unentscheidbarkeitsresultate erzielt. Es wurde erkannt, dass die Inferenzprobleme bereits für recht einfache DLs (im „worst case“) intraktable sind. Subsumption in KL-ONE wurde als unentscheidbar bewiesen [Schmidt-Schauß 1989]. Damit war klar, dass die in KL-ONE implementierten polynomiellen Beweisverfahren unvollständig sein mussten.

Die Reaktion der Entwickler des CLASSIC-Systems („Classification of Individuals and Concepts“) [Brachman u. a. 1989] war daher, lediglich eine sehr einfache DL zu implementieren, $\mathcal{ALIN\mathcal{F}h}(\mathcal{O})$, sodass Traktabilität garantiert schien. Der CLASSIC-Subsumptionsentscheider arbeitet ebenfalls strukturell, ist polynomiell und *nahezu* vollständig. Die in CLASSIC verwendete Approximation für Nominale ist jedoch unvollständig (daher wird $\mathcal{O}$ in Klammern gesetzt). Ohne Nominale ist CLASSIC jedoch vollständig. Aufgrund der Beschränkungen der Ausdrucksmächtigkeit von $\mathcal{ALIN\mathcal{F}h}(\mathcal{O})$ waren Anwendungen jedoch nur recht schwierig mit CLASSIC zu realisieren (der Autor vermisste in [Wessel 1996] insbesondere die Disjunktion).

- Phase 3 – Die Phase der Tableau-Verfahren für ausdrucksstarke DLs und gründlichen Komplexitätsanalysen. Insbesondere wurden nun ausdrucksstarke DLs mit voller Negation und Disjunktion untersucht. Das *ALC*-Papier [Schmidt-Schauß u. Smolka 1991] ist dieser Phase zuzuordnen. Das KRIS-System („Knowledge Representation and Inference System“) [Baader u. Hollunder 1991] war das erste System, welches eine ausdrucksstarke DL mit einem vollständigen Tableau-Verfahren implementierte. Obwohl Systeme wie KRIS als erfolgreich einzustufen sind, waren die unoptimierten Tableau-Verfahren jedoch zu ineffizient für größere Wissensbasen.

Die Korrespondenzen zur Modallogik wurden erkannt und ermöglichten den Transfer einer ganze Reihe von Komplexitätsergebnissen und Techniken aus der Modallogik. Beispielsweise kann die aus der Modallogik bekannte Technik des *Abwickelns (Unravelling)* aus der Perspektive des Autors als *inverses Blockieren* verstanden werden.

- Phase 4 – Optimierte Entscheidungsalgorithmen für *sehr* ausdrucksstarke DLs: Das FACT-System („Fast Classification of Terminologies“) demonstrierte zum ersten Mal, dass DL-Systeme für sehr ausdrucksstarke Sprachen wie $\mathcal{ALCHfN}_{\mathcal{R}+}$ mit sehr großen Terminologien (TBoxen) umgehen können. Diese Performanz wurde letztlich durch die Kombination von unvollständigen Verfahren mit Tableau-Verfahren ermöglicht, und durch die Integration von Optimierungsverfahren, die für aussagenlogische Erfüllungstester entwickelt worden waren. Ein weiteres einflussreiches System war DLP [Patel-Schneider 1998].

2.2 Beschreibungslogiken

Weder FACT (noch DLP) boten ABoxen (auch die Nachfolger FACT++ und IFACT nicht); diese Lücke füllte das RACE-System („Reasoner for ABox and Concept Expressions“) [Haarslev u. Möller 2000]. RACE wurde später in RACER umbenannt („Renamed ABox and Concept Expression Reasoner“). RACER [Haarslev u. Möller 2001c] kann als das einflussreichste ABox-DL-System aus Phase 4 bezeichnet werden. Insbesondere war RACER das erste System, welches OWL DL (partiell) unterstützte. Weitere Systeme wie PELLET [Sirin u. Parsia 2006] und KAON2 [Hustadt u. a. 2004] folgten.

Heutzutage wird eine kommerzielle Reimplementierung und Weiterentwicklung namens RACERPRO von der Firma Racor Systems vertrieben; die unterstützte DL ist $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$. Die (eingeschränkten, s. $\mathcal{D}^-$ in Namen) konkreten Domänen sind jedoch fix und nicht benutzerdefinierbar. RACERPRO bietet folgende konkrete Domänen:

- Die Domäne der natürlichen Zahlen mit Ordnung:
 $\mathcal{D}_{\mathbb{Z}} = (\mathbb{Z}, \{<, >, \geq, \leq\})$.
- Lineare Gleichungen und Ungleichungen zwischen Polynomen mit beliebig vielen Variablen über $\mathbb{R}$ (oder $\mathbb{N}$) und Ordnung (Prädikat $<$) – die bereits diskutierte Tarski-Domäne.
- Die Domäne der Zeichenketten mit Gleichheit und Ungleichheit:
 $\mathcal{D}_{\mathbb{S}} = (\mathbb{S}, \{=, \neq\})$.

Das RACERPRO-System zeichnet sich insbesondere auch durch die im Rahmen dieser Arbeit entstandene ausdrucksstarke Anfragesprache NRQL aus.

Das RACERPRO-System kann sehr große TBoxen (mehrere 100.000 Axiome, [Haarslev u. Möller 2001a]) verarbeiten. Die Verarbeitung sehr großer ABoxen bzw. die Anfragebeantwortung auf diesen ist jedoch noch Gegenstand aktueller Forschung [Möller u. a. 2006a;b], und zum Teil auch dieser Arbeit. Zudem ist RACERPRO momentan ein rein Hauptspeicherbasiertes System (Stand: Sommer 2006). Als grobe Richtlinie kann man zum Zeitpunkt des Schreibens dieser Arbeit (April 2006) davon ausgehen, dass das RACERPRO-System ABoxen mit einigen tausend oder zehntausend Individuen verarbeiten kann. Im Laufe der Jahre wurden die durch RACERPRO verarbeitbaren ABoxen jedoch immer größer, wozu letztlich auch die Arbeiten des Autors beigetragen haben. Dies betrifft insbesondere auch die ABox-Anfragebeantwortung mittels NRQL.

2.2.8.1. Die Beispiel-Wissensbasis als RACERPRO-Wissensbasis

In Kap. 2.2.5 hatten wir eine Beispiel-Familienwissensbasis diskutiert. Wir geben nun diese Wissensbasis in der konkreten Syntax des RACERPRO-Systems

an. Die konkrete Syntax von RACERPRO ist eine stark erweiterte KRSS-Syntax. Wir führen so anschaulich in die Nutzung eines konkreten DL-Systems (RACERPRO) ein und verdeutlichen die KRSS-Syntax. Die TBox sieht in konkreter Syntax wie folgt aus:

```
(in-tbox family)

(define-primitive-role has-descendant :inverse has-ancestor
                                :transitive t)

(define-primitive-role has-parent
                                :domain person
                                :range parent
                                :inverse has-child
                                :parent has-ancestor)

(define-concrete-domain-attribute age :type real)

(define-concrete-domain-attribute name :type string)

(define-primitive-role has-son :parent has-child
                                :domain person :range male-person)

(define-primitive-role has-daughter :parent has-child
                                :domain person :range female-person)

(implies top (and mammal (or male female)))

(equivalent male (not female))

(implies person (and (an age)
                    (exactly 2 has-parent person)
                    (exactly 1 has-parent male-person)
                    (exactly 1 has-parent female-person)))

(equivalent male-person (and person male))

(equivalent female-person (and person female))

(equivalent adult (and person (>= age 18.0)))

(equivalent man (and male-person adult))

(equivalent woman (and female-person adult))
```

2.2 Beschreibungslogiken

```
(equivalent parent (and person (some has-child top)))  
  
(equivalent mother (and parent female-person))  
  
(equivalent father (and parent male-person))  
  
(equivalent mother-of-son (and mother (some has-son top)))  
  
(equivalent mother-of-daughter (and mother (some has-daughter top)))  
  
(equivalent mother-having-only-daughters  
  (and mother  
    (all has-child female-person)  
    (some has-daughter top)))  
  
(equivalent grandmother  
  (and woman (some has-child parent)))  
  
(equivalent special-mother  
  (and mother-having-only-daughters mother-of-son))
```

Die ABox sieht in konkreter RACERPRO-Syntax wie folgt aus:

```
(in-abox family family)  
  
(instance margrit woman)  
(instance rudolf man)  
(instance birgit person)  
(instance birgit (exactly 2 has-child))  
  
(related margrit birgit has-daughter)  
(related margrit michael has-son)  
(related rudolf birgit has-daughter)  
(related rudolf michael has-son)  
(related birgit mara has-daughter)  
(related birgit alina has-daughter)  
  
(constrained michael michael-age age)  
(constrained michael michael-name name)  
(constrained birgit birgit-age age)  
  
(constraints (= michael-age 35.0))  
(constraints (> birgit-age michael-age))  
(constraints (string= michael-name "Michael Wessel"))
```

Zur Illustration der Dienst-Schnittstelle lösen wir die am Beispiel diskutierten Standard-Inferenzen aus:

```
> (concept-satisfiable? special-mother)
NIL

> (concept-subsumes? mother grandmother)
T

> (concept-equivalent? top mammal)
T

> (concept-synonyms mammal)
(MAMMAL *TOP* TOP)

> (concept-disjoint? woman man)
T

> (tbox-coherent?)
NIL

> (taxonomy)
S. Abb. 2.3

> (abox-consistent?)
T

> (individual-instance? margrit grandmother)
T

> (individuals-related? rudolf alina has-descendant)
T

> (concept-instances woman)
(MARGRIT BIRGIT)

> (individual-fillers margrit has-descendant)
(MICHAEL MARA ALINA BIRGIT)

> (related-individuals has-parent)
((MARA BIRGIT)
 (ALINA BIRGIT)
 (BIRGIT MARGRIT)
 (BIRGIT RUDOLF)
 (MICHAEL MARGRIT)
 (MICHAEL RUDOLF))
```

2.2.9. Beschreibungslogiken und Datenbanken

Die in dieser Arbeit erstellte Rahmenarchitektur und prototypischen ontologiebasierten Informationssysteme auf Basis von DL-Technologie ermöglichen im Vergleich zu auf konventioneller Datenbanktechnologie basierenden Informationssystemen einige Mehrwertdienste. Wir wollen daher zunächst das klassische relationale Datenmodell als prominentestes und wichtigstes Datenbankmodell logisch analysieren bzw. rekonstruieren, und dann mit dem DL-„Datenmodell“ vergleichen. *Datenbank* verwenden wir im Folgenden als synonym zu *relationale Datenbank*; oftmals ist hiermit auch das Datenbanksystem gemeint, doch wird aus dem Kontext deutlich werden, was gemeint ist.

DLs eröffnen aufgrund ihrer Deduktivität auch viele Nutzungsmöglichkeiten auf der *intensionalen Ebene*, also der Ebene des konzeptuellen oder logischen Schemas einer Datenbank bzw. eines Informationssystems. Aus logischer Perspektive sind durch ein Schema gewisse Bedingungen (Constraints) gegeben, die in allen Datenbanksausprägungen notwendigerweise erfüllt sein müssen. DLs können z.B. zur Konsistenzprüfung des Schemas verwendet werden (vergleichbar einer TBox-Konsistenzprüfung) und könnten daher bereits beim Datenbankentwurf zur Qualitätssicherung genutzt werden. Es existieren auch Werkzeuge, die ER- oder UML-Diagramme (welche oftmals zum konzeptuellen Entwurf verwendet werden) auf DLs abbilden und dann Konsistenzprüfungen durchführen [Franconi u. Ng 2000]. In dieser Arbeit beschränken wir uns jedoch vorwiegend auf die extensionale Perspektive.

2.2.9.1. Charakterisierung des relationalen Modells

Ein *relationales Datenbankschema* $\mathcal{R}$ ist eine Menge von Relationenschemata R_i : $\mathcal{R} =_{def} \{R_1, \dots, R_n\}$ [Abiteboul u. a. 1995]. Ein *Relationenschema* R_i kann als ein n -stelliger typisierter Relationsname bzw. n -stelliger Prädikatsname verstanden werden. Jede Spalte bzw. Position j der n Positionen des Relationsnamens ist durch ein Attribut A_j eines entsprechenden *Typs* benannt. Wir schreiben daher $R_i : A_1 \times \dots \times A_n$, wobei A_j die Attribute sind (nicht zu verwechseln mit den DL-Attributen, also den Elementen in $\mathcal{N}_A$). Ein Attribut A_i hat einen Typ, der letztlich eine konkrete Grundmenge bezeichnet, z.B. $\mathbb{N}$. Wir bezeichnen die Domäne für das Attribut A_i mit $\mathcal{D}_{A_i}$.

Eine *Datenbankinstanz* bzw. *Ausprägung* des Schemas R_i kann als Interpretation des Namens R_i angesehen werden: $R_i^{\mathcal{I}}$ ist somit eine Menge von n -stelligen Tupeln mit $R_i^{\mathcal{I}} \subseteq \mathcal{D}_{A_1} \times \dots \times \mathcal{D}_{A_n}$. Eine Interpretation ist hierbei wie in der Prädikatenlogik als Paar $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ definiert, doch bildet $\cdot^{\mathcal{I}}$ lediglich Relationsnamen auf Relationen ab; das Universum $\Delta^{\mathcal{I}}$ ist zudem partitioniert durch die Domänen der Attribute. Die Attribute A_j spezifizieren letztlich Zugriffsfunktio-

nen $A_j : \Delta^{\mathcal{I}^n} \mapsto \mathcal{D}_{A_j}$: für $(i_1, \dots, i_n) \in R_i^{\mathcal{I}}$ ist $A_j((i_1, \dots, i_n)) =_{\text{def}} i_j$ und $i_j \in \mathcal{D}_{A_j}$.

Eine Datenbankinstanz von R_i kann auf zwei Arten angegeben werden, vgl. [Abiteboul u. a. 1995]:

1. Die *beweistheoretische Sichtweise* bzw. *Sichtweise aus der Logikprogrammierung* ist, dass eine Relation durch eine endliche Menge von *Fakten* extensional spezifiziert wird, z.B. durch $\{R(a, b), R(c, d)\}$. Offensichtlich können wir diese Fakten, solange es sich um ein- und zweistellige Prädikatsnamen handelt, auch als (atomare) Konzept- und (atomare) Rollen-Assertionen notieren: $a : C$ bzw. $(a, b) : R$. Allerdings werden für eine solche ABox bei der Anfragebeantwortung andere Annahmen gemacht, s.u.

Eine solche Formelmenge ist offensichtlich immer erfüllbar, und es gibt ein *kleinstes, eindeutiges Herbrand-Modell*. Dieses ist auch eine relationale Struktur (über der Herbrand-Grundmenge). Natürlich gibt es auch noch weitere Modelle. Bei der Anfragebeantwortung werden im relationalen Modell jedoch i.d.R. nur die kleinsten Herbrand-Modelle betrachtet.

In einem kleinsten Herbrand-Modell gilt $R^{\mathcal{I}} = \{(a, b), (c, d)\}$, weil $a^{\mathcal{I}} = a$, und somit kann das Herbrand-Modell als „isomorph“ zur Faktenbasis angesehen werden. Zudem wird für die Konstanten natürlicherweise die UNA gemacht.

2. Die *modelltheoretische Sichtweise* ist, dass $R_i^{\mathcal{I}}$ durch eine endliche relationale Struktur bzw. ein endliches relationales Herbrand-Modell $\mathcal{I}$ vorgegeben ist. Nun ist $R_i^{\mathcal{I}}$ also eine vorgegebene Datenbankinstanz: $R^{\mathcal{I}} = \{(a, b), (c, d)\}$.

Anhand von $\mathcal{I}$ können wir dann die endliche Theorie

$$\text{Th}(\mathcal{I}) = \{ F \mid F \in \mathcal{G}_{\mathcal{R}}, \mathcal{I} \models F \}$$

bilden, die sog. *relationale Theorie* des Modells. Hierbei ist $\mathcal{G}_{\mathcal{R}}$ die Menge aller Fakten, die durch Verwendung der Relationsnamen $\mathcal{R}$ und der Konstanten in der Herbrand-Domäne gebildet werden können. Wir erhalten dann für jedes R_i eine Faktenbasis, für R erhalten wir $\{R(a, b), R(c, d)\}$.

Letztlich handelt es sich unter beiden Sichtweisen um extensionale Repräsentationen endlicher Mengen $R_i^{\mathcal{I}}$. Dies ist in Datenbanken möglich, da nur endliche Grundmengen betrachtet werden.

Nach Reiter [Reiter 1984] sind beide Sichtweisen äquivalent. Betrachten wir die möglichen Modelle der relationalen Theorie $\text{Th}(\mathcal{I})$, so können diese zum originalen Herbrand-Modell $\mathcal{I}$ isomorph gemacht werden, wenn die folgenden zusätzlichen Axiome (Sätze) zu $\text{Th}(\mathcal{I})$ hinzugefügt werden. Die Modelle der so

2.2 Beschreibungslogiken

gewonnenen *erweiterten* relationalen Theorie sind dann isomorph zu $\mathcal{I}$. Folgende Axiome werden hinzugefügt:

- $\forall \vec{x}. (R_i(\vec{x}) \leftrightarrow (\vec{x} = \vec{a}_1 \vee \dots \vee \vec{x} = \vec{a}_m))$, wobei $\vec{a}_1, \dots, \vec{a}_m$ eine Auflistung aller Tupel in $R_i^{\mathcal{I}}$ ist.

Hierdurch werden die Extensionen der Relationen *abgeschlossen*. Somit werden Modelle ausgeschlossen, die *mehr Tupel* für $R_i^{\mathcal{I}}$ zulassen als im originalen Herbrandmodell.

- $\neg(k = k')$ für jedes Paar von Konstanten aus der Herbrand-Grundmenge $\Delta^{\mathcal{I}}$.

Hierdurch wird die UNA für Konstanten sichergestellt.

- $\forall x. (x = c_1 \vee \dots \vee x = c_n)$, wobei $c_1, \dots, c_n$ eine Auflistung aller Konstanten in der Herbrand-Grundmenge $\Delta^{\mathcal{I}}$ ist.

Hierdurch wird die Grundmenge *abgeschlossen* (Domain Closure Axiom).

Natürlich gehört zum relationalen Modell noch eine Anfragesprache, z.B. relationale Algebra, bzw. der relationale Tupel- oder Domänenkalkül. SQL kann als Umsetzung des relationalen Tupelkalküls begriffen werden. Relationale Algebra steht der modelltheoretischen Sichtweise näher. Die Relationale Algebra beinhaltet die Operatoren Projektion (π), Selektion (σ), Kreuzprodukt ($\times$), Umbenennung (R), Vereinigung ($\cup$) und Komplement ($\setminus$) aus. Andere „vollständige Basen“ von Operatoren sind bekannt (z.B. kann der natürliche Verbund $\bowtie$ statt des Kreuzproduktes verwendet werden). Es ist auch bekannt, dass relationale Algebra Defizite hat, z.B. kann die transitive Hülle einer Relation nicht mittels relationaler Algebra berechnet werden (hierfür kann kein endlicher Ausdruck angegeben werden). Aus der Perspektive der Logikprogrammierung sind Sprachen wie Datalog entwickelt worden; Datalog mit Negation (ohne Rekursion und Disjunktion) hat die gleiche Ausdruckskraft wie relationale Algebra. Sprachen wie Datalog stehen der beweistheoretischen bzw. Sichtweise der Logikprogrammierung näher. Hier spricht man auch von *deduktiven Datenbanken*. Für Details s. [Abiteboul u. a. 1995].

2.2.9.2. Unterschieden zwischen DLs und dem relationalen Modell

Die Unterschiede zwischen einer DL-ABox und einer relationalen Datenbankinstanz können wir (für unsere Zwecke ausreichend genau) wie folgt charakterisieren, vgl. [Borgida u. a. 2003, Borgida 1995, Calvanese u. a. 1998b] und [Borgida u. a. 2003]:

- Der erste offensichtliche Unterschied betrifft die Stelligkeit der Prädikate: Datenbanken bieten n -stellige Prädikatsnamen bzw. Relationen, während Standard-DLs (wie die hier vorgestellten) lediglich ein- und zweistellige

Prädikatsnamen (atomare Konzepte und Rollen) bieten. Tatsächlich sind jedoch auch DLs mit n -stelligen Konzepten und Rollen untersucht worden (z.B. DLR, [Calvanese u. a. 1997]).

- Aufgrund der größeren Ausdrucksmächtigkeit können mit DLs (sofern man sich auf ein- und zweistellige Prädikate beschränkt) prinzipiell reichhaltigere Domänenmodelle bzw. Konzeptualisierungen erstellt und formalisiert werden.
- Traditionell sind die Anfragesprachen von Datenbanksystemen (z.B. SQL) „reifer“ als entsprechende Anfragesprachen für DL-Systeme. Tatsächlich existieren selbst heutzutage nur wenige DL-Systeme, die überhaupt die notwendige Voraussetzung für eine extensionale Anfragesprache bieten (ABoxen müssen verfügbar sein). Der Entwurf und die Implementierung erweiterter DL-Anfragesprachen ist ein wesentliches Anliegen dieser Arbeit. Insbesondere werden für viele Anwendungen tupelwertige Anfrageergebnisse benötigt, wie sie von SQL-Anfragen bekannt sind; d.h., die Implementierung der Standard-Retrievalprobleme aus Kap. 2.2.2.5 ist nur ein erster Schritt.
- Als semantische Integritätsbedingungen sind in Datenbanken lediglich attributsbezogene Typisierungseinschränkungen sowie funktionale Abhängigkeiten (Schlüsseleigenschaften) formulierbar. Zwar können komplexere semantische Einschränkungen (etwa, dass das Gehalt eines Managers nicht niedriger als das Gehalt eines beliebigen Angestellten sein darf) durch sog. Trigger-Regeln z.B. nach Datenbankupdates (Einfügen eines neuen Tupels in einer Tabelle etc.) in Sprachen wie SQL überwacht werden, doch entzieht sich deren prozedurale Semantik dem relationalen Modell.

Integritätsbedingungen können prinzipiell als zusätzliche $\mathcal{L}_{\mathcal{FOPL}}$ -Sätze (mit Signatur $\mathcal{R}$, also ohne Funktionssymbole) in der erweiterten relationalen Theorie verstanden werden. Durch sie werden dann lediglich solche Datenbankinstanzen zugelassen, die Modelle dieser Sätze sind.

In einer DL-TBox (oder „RBox“) können bestimmte Typen solcher Axiome bzw. Sätze bereitgestellt werden. Beispielsweise kann eine Relation (Rolle) als transitiv abgeschlossen, oder zwei Relationen als zueinander invers deklariert werden. Dafür sind funktionale Abhängigkeiten in DLs nicht unmittelbar verfügb- bzw. formulierbar.

- Die betrachteten relationalen Modelle einer Schemas bzw. die Datenbankausprägungen sind stets endlich. Hingegen haben ausdrucksstarke DLs i.d.R. nicht die Endliches-Modell-Eigenschaft. Daher können auch ABoxen lediglich unendliche Modelle haben.
- Datenbanken verwenden die *Closed Domain Assumption (CDA)*, während

2.2 Beschreibungslogiken

DLs die *Open Domain Assumption (ODA)* für die Anfragebeantwortung bzw. „Inferenz“ verwenden [Brachman u. Levesque 2004, Kap. 11.2.5]. Wie erläutert, kann in Datenbanken die Grundmenge $\Delta^{\mathcal{I}}$ als die Menge der Konstanten, die in den Datenbankinstanzen verwendet werden, betrachtet werden. Somit hat jedes Element aus $\Delta^{\mathcal{I}}$ einen Namen.

In DLs hingegen können Konzepte die Existenz weiterer Individuen in den Modellen erzwingen, die keinen Namen haben bzw. für die es keine Konstanten bzw. keine ABox-Individuen gibt. Beispielsweise müssen alle Modelle der ABox $\{i : \exists R.C\}$ ein Individuum $x \in C^{\mathcal{I}}$ mit $(i^{\mathcal{I}}, x) \in R^{\mathcal{I}}$ enthalten. Für x ist kein ABox-Individuum vorhanden. In den Modellen können also implizite Individuen vorhanden sein, d.h., die Domäne ist nicht abgeschlossen.

- Datenbanken können keine *indefiniten Informationen* (in semantisch wohlfundierter Weise) darstellen. ABoxen können hingegen in ausdrucksstarken DLs Disjunktionen enthalten. So hat $\{i : C \sqcup D\}$ sowohl ein Modell mit $i^{\mathcal{I}} \in C^{\mathcal{I}}$ als auch ein Modell mit $i^{\mathcal{I}} \in D^{\mathcal{I}}$. Datenbanken bieten zwar *Nullwerte*, diese haben jedoch eine zweifelhafte Semantik und führen zu Komplikationen in den Anfragesprachen.
- Datenbanken verwenden die *Closed World Assumption (CWA)*, während DLs die *Open World Assumption (OWA)* für die Anfragebeantwortung bzw. „Inferenz“ verwenden [Brachman u. Levesque 2004, Kap. 11.2.1], [Baader u. a. 2003a, Kap. 2.2.4.4], [Genesereth u. Nilsson 1987, Kap. 6.1].

Unter der CWA wird aus der Abwesenheit eines Faktums (z.B. der Abwesenheit eines Tupels in einer bestimmten Tabelle bzw. Relation) geschlossen, dass das Faktum falsch sein muss. Unter der OWA wird dieser Schluss nicht gemacht; nach wie vor werden sowohl Modelle, in denen das Faktum erfüllt, als auch solche, in denen das Faktum nicht erfüllt ist als möglich erachtet. Unter der CWA werden hingegen nur noch letztere Modelle als möglich erachtet. Da die betrachteten Modelle unter der CWA also eingeschränkt werden, ist das negierte Faktum dann eine logische Konsequenz (es folgt). Unter der CWA wird daher angenommen, dass die Domäne vollständig abgeschlossen modelliert und erfasst wurde („Was ich nicht weiß, ist falsch“).

Zur Illustration der Unterschiede zwischen CWA und OWA sei die Faktenbasis bzw. die einfache ABox

$$\{ \text{has\_child}(\text{birgit}, \text{alina}), \text{has\_child}(\text{birgit}, \text{mara}), \\ \text{female\_person}(\text{alina}), \text{female\_person}(\text{mara}) \}$$

betrachtet. Lediglich unter der CWA ist nun folgerbar, dass *birgit genau zwei Kinder hat, und dass alle ihre Kinder weiblich sind*. Unter der OWA ist lediglich ableitbar, dass *birgit mindestens zwei weibliche Kinder hat*.

Unter der CWA wird die Anfrage, ob $has\_child(birgit, tom)$ gilt, mit FALSE beantwortet, und FALSE wäre als „Falsch“ bzw. „Nein“ zu interpretieren, in dem Sinne, dass hiermit $\neg has\_child(birgit, tom)$ gemeint ist. Unter der OWA würde diese Anfrage ebenfalls mit FALSE beantwortet werden, allerdings muss FALSE nun also „Unbekannt“ interpretiert werden, bzw. als $has\_child(birgit, tom)$ *kann nicht bewiesen werden*. Die CWA hat jedoch die Eigenschaft der *Nicht-Monotonie*. Wird nämlich zur Faktenbasis nun $has\_child(birgit, tom)$ hinzugefügt, so kann die Inferenz $\neg has\_child(birgit, tom)$ nicht mehr gezogen werden. Dies verletzt also die Monotonie-Eigenschaft der Folgerungsrelation „ $\models$ “ von $\mathcal{L}_{FOPL}$. Die Monotonie-Eigenschaft besagt, dass für beliebige $\mathcal{F} \subseteq \mathcal{F}_{FOPL}$, $\mathcal{F}' \subseteq \mathcal{L}_{FOPL}$ mit $\mathcal{F} \subseteq \mathcal{F}'$ und $F \in \mathcal{L}_{FOPL}$ aus $\mathcal{F} \models F$ auch $\mathcal{F}' \models F$ folgt; diese Eigenschaft gilt daher natürlich auch für „ $\vdash$ “, also die Ableitbarkeitsrelation.

Die CWA ist auch problematisch, sobald Disjunktionen erlaubt sind. So sind z.B. aus der ABox $\{i : C \sqcup D\}$ sowohl $i : \neg C$ als auch $i : \neg D$ per CWA „folgerbar“. Die so entstandene „Theorie“ ist aber bereits inkonsistent, denn $\{i : C \sqcup D, i : \neg C, i : \neg D\}$ ist unerfüllbar. Letztlich ist die CWA eine Konsequenz daraus, dass eine Datenbankinstanz als eindeutiges *minimales* Herbrandmodell betrachtet wird (bzw. ist eine Konsequenz aus den zusätzlichen Axiomen in der erweiterten relationalen Theorie). Ein solches eindeutiges minimales Modell gibt es nicht mehr, sobald Disjunktionen vorhanden sind.

Die CWA wird in Sprachen wie Prolog als *Negation as Failure (NAF)* realisiert.

- Aufgrund der oben genannten Punkte (unvollständige bzw. indefinite Information, OWA, ODA, reichhaltigere Domänen-Modellierungen) ist *Anfragebeantwortung* bzw. die Berechnung der Menge der *sicheren Antworten einer Anfrage* (s. Kap. 2.1.1.5) in DLs wesentlich schwieriger als in Datenbanken. Während die Anfragebeantwortung im relationalen Modell als (wiederholtes) Model Checking-Problem einer Formel aus $\mathcal{L}_{FOPL}$ (der Anfrage) formalisiert werden kann, ist für DL-Anfragebeantwortung Deduktion bzw. Theorembeweisen erforderlich.
- Beschreibungslogiken ermöglichen *ontologiebasierte Anfragebeantwortung*. In ontologiebasierten Anfragen wird das domänenspezifische Vokabular aus einer Ontologie verwendet. Naturgemäß sind solche Anfragen vom Nutzer einfacher zu formulieren, da er das Vokabular der Domäne bzw. die zugrundeliegende Konzeptualisierung der IS-Domäne i.d.R. gut kennt. Hingegen erfordert Anfrageformulierung mittels SQL tieferes Wissen bzgl. des Datenmodells. Ontologiebasierte Anfragebeantwortung ist wesentliches Thema dieser Arbeit.
- DLs sind gut geeignet zur Repräsentation *semistrukturierter Daten*

2.2 Beschreibungslogiken

([Buneman 1997, Calvanese u. a. 1998a]). Ein relationales Schema ist starr, die Daten müssen uniform den Definitionen der R_i -Schemata folgen: $R_i : A_1 \times \dots \times A_n$. Abweichungen hiervon können lediglich durch Nullwerte mit zweifelhafter Semantik repräsentiert werden.

In ABoxen können Daten jedoch *selbstbeschreibend* und bei Bedarf sogar schemalos (also ohne TBox) repräsentiert werden. Beliebige Graphstrukturen lassen sich aufspannen. Oftmals werden beschriftete Graphen als Datenmodelle für semistrukturierte Daten verwendet [Calvanese u. a. 1998a]. Solche Datenmodelle lassen sich naturgemäß gut auf ABoxen abbilden.

2.3. Zusammenfassung

In diesem Kapitel haben wir die zum Verständnis der Arbeit notwendigen Grundlagen beleuchtet. Dies war zum einen erforderlich, um die Arbeit in sich abgeschlossen zu machen, und zum anderen, um die vom Autor verwendete Terminologie explizit zu machen.

Wir haben die theoretischen Grundlagen aus der Welt der logikbasierten Wissensrepräsentation und der wissensbasierten Systeme dargestellt. Prädikatenlogik und Aussagenlogik wurden in einem für diese Arbeit ausreichendem Detaillierungsgrad eingeführt, und wichtige Begriffe (Syntax, Semantik, Modell, Repräsentation, Datum, Information, Informationssystem, Thesaurus, Konzeptualisierung, Ontologie) wurden eingeführt und präzisiert. Die logisch-theoretischen Grundlagen der DLs wurden erläutert, deren Syntax und Semantik, sowie die wichtigen sog. Standard-Inferenzprobleme definiert. Stellvertretend für die wichtigste Klasse von Kalkülen zur Entscheidung des Erfüllbarkeitsproblems für Konzepte und ABoxen, die sog. Tableau-Kalküle, haben wir einen einfachen $\mathcal{ALC}$ -Tableau-Kalkül diskutiert, und seine Eigenschaften (Terminierung, Vollständigkeit, Korrektheit) untersucht. Schließlich haben wir die typischen Leistungen von DL-Systemen beschrieben und durch Verwendung des RACERPRO-Systems anhand einer Beispiel-Wissensbasis konkretisiert und illustriert. Diese Beispiel-Wissensbasis wird im weiteren Verlauf der Arbeit noch öfter verwendet werden. Da relationale Datenbanken eine wichtige Rolle in dieser Arbeit spielen, haben wir DLs mit dem relationalen Datenbankmodell verglichen, und die wichtigsten Unterschiede explizit gemacht. Dabei wurden die durch DL-Systemen ermöglichten Mehrwertdienste diskutiert; insbesondere wurde der Begriff der ontologiebasierten Anfragebeantwortung eingeführt.

3. Ontologiebasierte Informationssysteme – Zwei Leitdomänen

Inhaltsangabe

3.1	Leitdomäne 1: Geographische Informationssysteme	133
3.1.1	Datenmodelle für GIS und Aspekte von Geo-Daten	134
3.1.2	Die DISK, ein konkreter Geo-Datenbestand	143
3.1.3	Ein DL-inspirierter Beschreibungsrahmen	144
3.1.4	Die Mehrwertdienste eines ontologiebasierten GIS	148
3.1.5	Beschreibungslogiken mit RCC-Relationen	150
3.1.6	Nutzungspotentiale und -defizite von DLs & DL-Systemen	158
3.1.7	Fazit	162
3.2	Leitdomäne 2: Information Retrieval im SEMANTIC WEB	163
3.2.1	Die Vision des SEMANTIC WEB	163
3.2.2	Der SEMANTIC WEB-Stack	165
3.2.3	Anfragesprachen für das SEMANTIC WEB	177
3.2.4	Nutzungspotentiale und -defizite von DLs & DL-Systemen	179
3.2.5	Fazit	185
3.3	Zusammenfassung	187

Dieses Kapitel stellt *zwei Leitdomänen* für ontologiebasierte Informationssysteme (ISe) vor:

1. die Domäne der geographischen Informationssysteme,
2. die Vision des SEMANTIC WEB.

Wir *beschreiben und analysieren beide Domänen* zur Beleuchtung der *Mehrwertdienste*, die durch Nutzung von Ontologien und logischer Deduktion in diesen Domänen ermöglicht werden, bzw. in der 2. Domäne bereits immanent sind. Dieses Kapitel dient also ganz wesentlich der Vorstellung der IS-Domänen, und der *Analyse und Anforderungsermittlung* für die im weiteren Verlauf der Arbeit zu entwickelnde Software zur Realisierung ontologiebasierter ISe. Dabei richten wir unser Hauptaugenmerk auf die *ontologiebasierter Anfragebeantwortung*.

Wir wählen bewusst zwei IS-Domänen, die aus Forschungsperspektive noch

„Potential bieten“ und somit noch nicht bereits erschöpfend durch konventionellere IS-Technologie abgewickelt worden sind (anders als z.B. betriebliche Personalinformationssysteme mittels relationaler Datenbanken, etc.). Beide Domänen können als *Nicht-Standard-Domänen* für ISe und somit in gewisser Weise als *innovative* und zudem auch *datenintensive* IS-Domänen bezeichnet werden.

Wir analysieren und identifizieren die Einsatzmöglichkeiten von DLs und existierenden DL-Systemen zur *Realisierung* von ontologiebasierten ISen in diesen Domänen.

Warum sind DLs und DL-Systeme interessant für diese IS-Domänen? Die Relevanz von DLs bzw. DL-Systemen in diesen IS-Domänen, *die dann zur DL- bzw. DL-System-Nutzungs-Szenarien werden*, ergibt sich aus folgenden Gründen:

1. DLs verfügen über eine *formale Semantik*. Die durch DL-Systeme bereitgestellten Dienste können als Implementierungen von mathematisch rigoros definierten Problemlösern angesehen werden; hierbei ist stets Entscheidbarkeit bzw. Terminierung sicherzustellen. Dieses ist für ISe z.B. unter dem Gesichtspunkt *Sicherheit von Anfragen* relevant. Zudem werden Mehrwertdienste durch Inferenzen bzw. logikbasierte Deduktion ermöglicht (s.u.).
2. DLs bzw. DL-Systeme liefern eine *klare Referenzarchitektur*. Die Trennung in eine *intensionale* (TBox) und eine *extensionale* (ABox) Komponente kann als prototypisch für ontologiebasierte ISe bezeichnet werden und findet sich in ähnlicher Form auch in konventionellen ISen (Trennung in Schema + Daten, etc.).

Wir verwenden im Folgenden einen DL-inspirierten Beschreibungsrahmen für ontologiebasierte ISe. Insbesondere übernehmen wir die in DL-Systemen vorgenommene klare Aufteilung in eine intensionale und eine extensionale Komponente. Aus Gründen der größeren Flexibilität nehmen wir jedoch an, dass Prädikatenlogik erster Ordnung zur Semantikbeschreibung verwendet wird. Dies stellt sicher, dass schließlich auch DL-Systeme (deren Expressivität i.d.R. nicht über Prädikatenlogik erster Ordnung hinausgeht) als Instanzen dieses Beschreibungsrahmens gedeutet werden können.

3. DLs bilden eine ganze *Familie* von Sprachen mit wohluntersuchter Komplexität. Die für die Domäne benötigte Ausdrucksmächtigkeit kann *flexibel variiert* werden, wodurch insbesondere Skalierungs- und Performanzprobleme adressiert werden können. Dies setzt jedoch eine sehr generische und flexible Architektur voraus (s.u.).
4. Es existieren hochoptimierte DL-Systeme wie RACERPRO, deren Nutzung im Rahmen eines „Bottom Up“-Ansatzes als Basis-Maschinen zur Realisie-

rung ontologiebasierter ISe wünschenswert erscheint (heutzutage bilden relationale Datenbanksysteme das Fundament konventioneller ISe). Es ist daher aus einer pragmatischen Perspektive wichtig, die *Praxistauglichkeit* existierender DL-Systeme als *Basis-Technologie* für solche ISe zu evaluieren, um Defizite zu erkennen. Das *Experiment* ist eine wissenschaftliche Methode zum Erkenntnisgewinn. Die Nutzung von DL-Systemen zur Realisierung deduktiver ontologiebasierter ISe ist heutzutage immer noch innovativ. Die Untersuchungen und Nutzungsexperimente des Autors haben letztlich auch dazu beigetragen, dass das RACERPRO-System von den Autoren immer weiter verbessert wurde; zumal wurden *neue Anforderungen* formuliert und umgesetzt.

5. In der SEMANTIC WEB-Domäne ergibt sich die Relevanz von DLs und DL-Systemen unmittelbar, da SEMANTIC WEB-Sprachen wie OWL DL-inspiriert sind, und existierende DL-Systeme wie RACERPRO OWL unterstützen.
6. Die vorliegende Arbeit ist nicht frei von *Bottom Up*-Einflüssen. Die Arbeit des Autors wurde letztlich durch Forschungsprojekte mit klarem DL- und auch RACERPRO-Kontext finanziert.

So ergab sich die Wahl der SEMANTIC WEB-Domäne mehr oder weniger durch den Wunsch, eine effiziente und ausdrucksstarke ABox-Anfragesprache für das RACERPRO-System zur Verfügung zu stellen. Da RACERPRO OWL unterstützt, wurde deutlich, dass durch OWL auch an die ABox-Anfragesprache neue Anforderungen gestellt werden. Existierende Vorschläge für DL-ABox- und SEMANTIC WEB-Anfragesprachen schieden aus mehreren Gründen in diesem Kontext aus, sodass der Weg gewählt wurde, eine eigene Anfragesprache zu entwerfen und zu implementieren. Vorweggreifend soll erwähnt werden, dass diese Anfragesprache einige Alleinstellungsmerkmale unter den SEMANTIC WEB-Anfragesprachen hat. Diese können anhand der in diesem Kapitel vorgenommenen Analyse und Anforderungsdefinition begründet werden.

Wie erwähnt, liegt unser Hauptaugenmerk insbesondere auf der *ontologiebasierten Anfragebeantwortung in ontologiebasierten ISen*. Einige Autoren verwenden den Begriff *ontologiebasiertes Information Retrieval (IR)*. Wir verwenden die folgende, sehr allgemeine Definition von Information Retrieval [Wikipedia 2006a]:

Information Retrieval (IR) bzw. Informationswiedergewinnung, gelegentlich Informationsbeschaffung, ist ein Fachgebiet, das sich mit computergestütztem inhaltsorientiertem Suchen beschäftigt. Es ist ein Teilgebiet sowohl der Informationswissenschaft als auch der Informatik.

IR unterscheidet sich von klassischen Datenbank-Abfragen durch zwei wesentliche Charakteristika vgl. [Wikipedia 2006a], [Rechenberg u. Pomberger 2006, E5], [Ferber 2003; 2006, Baeza-Yates 1999]:

1. **Vagheit:** Der Nutzer kann sein Informationsbedürfnis i.d.R. nicht präzise und formal formulieren; idealerweise werden Anfragen daher in natürlicher Sprache formuliert. Formale Anfragesprachen (wie SQL) scheiden daher oftmals aus; statt dessen wird Volltextsuche, Suche nach Schlüsselwörtern, Suche mit webbasierten Formularen oder Schlagwortsystemen, Ähnlichkeitsbasierte Suche etc. verwendet.
2. **Unsicherheit:** Die geforderten Informationen stehen im System u.U. gar nicht explizit bereit, sondern müssen durch Auswertungsprogramme erst „gewonnen“ bzw. im Sinne obiger Definition „beschafft“ werden. Typischerweise wird IR auf Text- oder Multimedia-Dokumenten verwendet, die bestenfalls semistrukturiert (wenn nicht unstrukturiert) sind. Dementsprechend ist Maschinenauswertbarkeit der Inhalte solcher Ressourcen nicht ohne weiteres gegeben. Die Entscheidung, ob eine Ressource (ein Dokument) als *Treffer* einer Anfrage zurückgegeben und somit als relevant betrachtet werden sollte oder nicht, ist daher naturgemäß keine binäre (JA/NEIN-) Entscheidung, sondern oftmals lediglich graduell zu beantworten (zu 60% relevant). Dementsprechend spielen Relevanz-Maße oder Fuzzy-Retrieval-Modelle hier eine wichtige Rolle. Es werden jedoch auch „scharfe“, z.B. auf boolescher Logik basierende IR-Retrieval-Modelle betrachtet.

Die im Rahmen dieser Arbeit vorgestellten Verfahren zur DL-gestützten, ontologiebasierten Anfragebeantwortung sind letztlich „scharf“ und somit nicht „vage“, und die betrachteten Ressourcen sind auch nicht unstrukturiert (sondern strukturiert oder semistrukturiert). Die Nutzung von in einer Ontologie definiertem Vokabular ermöglicht dem Nutzer jedoch, *sein Informationsbedürfnis zu formulieren* (s. 1.), und durch tiefe logische ontologiebasierte Modellierung von Inhalten ist es möglich, die impliziten, *nicht explizit vorhandenen geforderten Informationen* (s. 2.) mittels logischer Deduktion explizit zu machen. Wir werden im Rahmen dieser Arbeit jedoch keine Relevanz-Maße betrachten.

Zudem eröffnet die Definition von IR als *inhaltsorientierter Suche* auch die Deutungsmöglichkeit von IR als *ontologiebasierter Anfragebeantwortung*, die wir in dieser Arbeit verfolgen. Insbesondere im SEMANTIC WEB-Konzept spricht man oft von SEMANTIC WEB Information Retrieval, und Ontologien spielen hier eine wesentliche Rolle.

Ontologiebasierte Anfragebeantwortung verspricht insbesondere auch ein günstiges Verhältnis von *Genauigkeit und Vollständigkeit (Precision und Recall)*, die im IR-Kontext wie folgt definiert sind [Baeza-Yates 1999, Kap. 3.2.1]:

Definition 49 (Genauigkeit (Precision) und Vollständigkeit (Recall)) Sei q eine Retrieval-Anfrage, und $\mathcal{R}$ eine Menge von Ressourcen (Dokumenten). Sei $\mathcal{R}_q \subseteq \mathcal{R}$ die Menge der bzgl. q relevanten Ressourcen, und $q^\mathcal{E}$ die *Extension* der Anfrage, also die Menge der anhand von q gelieferten Ressourcen (Treffer).

Da nicht notwendigerweise $q^\mathcal{E} \subseteq \mathcal{R}_q$ gelten muss, unterscheiden wir die Menge $q_+^\mathcal{E} =_{\text{def}} q^\mathcal{E} \cap \mathcal{R}_q$ der *relevanten Extension* von der Menge $q_-^\mathcal{E} =_{\text{def}} \mathcal{R} \setminus q_+^\mathcal{E}$ der *irrelevanten Extension*.

Dann ist die *Vollständigkeit (Recall)* der Anfrage definiert als Quotient aus der relevanten Extension der Anfrage und allen relevanten Ressourcen:

$$\text{recall}(q) =_{\text{def}} \frac{q_+^\mathcal{E}}{\mathcal{R}_q},$$

und die *Genauigkeit (Precision)* der Anfrage definiert als Quotient aus der relevanten Extension der Anfrage und der Extension der Anfrage:

$$\text{precision}(q) =_{\text{def}} \frac{q_+^\mathcal{E}}{q^\mathcal{E}}.$$

■

Liefert eine Anfrage q viele irrelevante Treffer, dann ist die Genauigkeit schlecht (sehr viel kleiner als 1). Liefert eine Anfrage hingegen eine große Teilmenge (oder sogar Obermenge) der relevanten Ressourcen, dann ist die Vollständigkeit gut (nahe bei 1). Offensichtlich kann recall auch maximiert werden, indem stets *alle* Dokumente zurückgeliefert werden – dann ist die Genauigkeit jedoch sehr schlecht. Für typische IR-Anwendungen sind diese Maße daher gegenläufig (vgl. [Rechenberg u. Pomberger 2006, Kap. E5.5.1]).

Da insbesondere $\mathcal{R}_q$ in unserer Informationsgesellschaft exponentiell wächst, besteht die Gefahr, dass recall fällt, wenn nicht $q_+^\mathcal{E}$ mit innovativen IR-Verfahren maximiert wird. Dies ist möglich, indem z.B. für die Schlagwörter in einer Retrievalanfrage auch deren in einem Thesaurus definierten *Synonyme* zum Retrieval herangezogen werden. Wir hatten bereits dargestellt, dass ein Thesaurus eine sehr einfache Art von Ontologie ist. Gleichzeitig kann durch Verwendung von Ontologien eine gute Genauigkeit erzielt werden, für die eine *tiefe Modellierung von Inhalten* erforderlich ist. Beispielsweise können *Homonyme* als falsche Treffer erkannt werden, was nicht ohne weiteres möglich ist bei reinen Volltext-Retrievalverfahren.

Wir werden jedoch auch Defizite von DLs und DL-Systemen in diesen IS-Domänen bzw. Nutzungs-Szenarien identifizieren. Die identifizierten Probleme und Defizite werden Anlass zur Definition eines eigenen Datenmodells in Kap. 4 geben, welches zwar DL-inspiriert ist, jedoch bestimmte Aspekte der Leitdomänen

besser unterstützen kann und somit die identifizierten Defizite von DLs bzw. DL-Systemen in diesen Szenarien lindert bzw. umgeht. Bei der Identifikation eines DL-Nutzungsproblems werden wir i.d.R. gleich eine entsprechende Lösungsmöglichkeit anbieten und diskutieren. Insofern weichen wir in dieser Darstellung zugunsten der Leserfreundlichkeit bewusst von einem starren Analyse-Entwurf-Schema ab.

Da die Informationen mittels des Datenmodells nicht nur repräsentiert, sondern auch ontologiebasiert abgefragt werden sollen, entwerfen wir später auch passende Anfragesprachen für diese Domänen. Sowohl Datenmodell als auch Anfragesprachen werden dann später softwaretechnisch umgesetzt. Das realisierte Rahmenwerk stellt die Basis-Architektur für die in dieser Arbeit implementierten prototypischen ontologiebasierten ISe bereit.

3.1. Leitdomäne 1: Geographische Informationssysteme

Geographische Informationssysteme (GIS) sind ISe zur Erfassung, Speicherung, Pflege, Analyse und Darstellung raumbezogener Sachverhalte [Bartelme 1995, Bill u. Fritsch 1991, Star u. Estes 1990]. Die raumbezogenen Sachverhalte werden durch elektronische Karten dargestellt bzw. räumlich indiziert. Die primäre Abstraktion in einem GIS ist die eines *Geo-Objektes* (auch *Feature* genannt) - hierbei handelt es sich um eine GIS-geeignete Repräsentation eines „Phänomens“ (nicht notwendigerweise eines „Objektes“) in der realen Welt. Geo-Objekte haben verschiedene *Aspekte*, die wir weiter unten diskutieren. Beispiele für Geo-Objekte sind: ein Teich, ein Land, ein Fluss, ein Haus, die Gebiete in Deutschland, in denen an mehr als 100 Tagen die Sonne schien, etc. Wir werden in dieser Arbeit eine spezielle GIS-Domäne betrachten, die *Domäne der digitalen Stadtkarten (s.u.)*. Dementsprechend nehmen wir die vereinfachende Sichtweise ein, dass die Geo-Objekte *Karten-Objekte* sind.

Die Präsentationskomponente eines GIS wird i.d.R. eine Kartendarstellung für Geo-Objekte verwenden; Karten sind auch das primäre Analyse-Werkzeug des GIS-Nutzers. Die Stärken eines GIS liegen insbesondere in der Fähigkeit, problemspezifische Karten dynamisch und bei Bedarf erzeugen zu können, anhand derer der Nutzer entsprechende räumliche Zusammenhänge und Sachverhalte schnell erkennen oder analysieren kann.

GIS werden oft zur *Entscheidungsunterstützung (Decision Support)*, z.B. zur Planung von Bauvorhaben bei der Stadtplanung, genutzt. Heutzutage werden jedoch auch Routenplaner und mit interaktiven Stadtplänen gekoppelte Internet-Suchmaschinen von einer breiten Nutzergemeinde täglich verwendet; elektronische Karten spielen hier eine wesentliche Rolle. Georeferenzierte Ressourcen werden zunehmend wichtiger, insbesondere auch zur Selektion von ortsbasierten bzw. -gebundenen Diensten („Location Based Services“) für mobile Geräte (PDAs, Mobiltelefone, etc.). Die Anwendung und Entwicklung von Ontologien ist hier ein sehr aktives Forschungsfeld.

Zunächst beschreiben wir die wesentlichen Aspekte von Geo-Daten sowie GIS-Datenmodellen. Für konkrete Experimente stand ein konkreter Geo-Datenbestand zur Verfügung, die sog. „Digitale Stadtkarte“, DISK, die wir alsdann beschreiben. Danach stellen wir einen DL-inspirierten Beschreibungsrahmen für die einzelnen Komponenten bzw. „Schichten“ eines *ontologiebasierten GIS* vor und beschreiben die Mehrwertdienste eines solchen GIS. Anschließend beleuchten wir die Frage, welche Art von DL für ein solches ontologiebasiertes GIS wünschenswert bzw. nutzbar wäre. Schließlich analysieren wir die Probleme und Potentiale bzw. Einsatzmöglichkeiten von Standard-DLs und Standard-DL-Systemen zur Realisierung eines ontologiebasierten GIS. Wie erläutert folgt

3.1 Leitdomäne 1: Geographische Informationssysteme

auf eine Problemdarstellung auch gleich eine (pragmatische) Problemlösung. Die Analyse motiviert weitere Arbeitspunkte. Wir schließen das Unterkapitel mit einem Fazit ab.

3.1.1. Datenmodelle für GIS und Aspekte von Geo-Daten

Ein GIS-Datenmodell muss den verschiedenen Facetten der Geo-Objekte Rechnung tragen: Man unterscheidet zwischen räumlichen, strukturellen, und thematischen Aspekten von Geo-Objekten.

Prinzipiell unterscheidet man zwischen vektor- und rasterbasierten Datenmodellen. Die beiden Arten von Datenmodellen werden oftmals als zueinander komplementär dargestellt. Wir beschränken uns hier auf die Vektor-Datenmodelle, da nur sie eine unmittelbare Objektabstraktion bzw. Geo-Objekte mit Identität bieten. Statt individueller Objekte werden im Rastermodell sog. *Ebenen (Layer, Coverages)* bzw. *Tesselationen* verwendet. Die Ebenen werden anhand von thematischen Gesichtspunkten angelegt bzw. organisiert (z.B. eine Ebene für Wasserflächen, eine Ebene für Grünflächen, ...). Eine Ebene ist i.d.R. ein einfaches Byte-Raster; die „Farbe“ eines Rasterelementes (Pixels) gibt Aufschluss über die Thematik bzw. Bedeutung des georeferenzierten „Punktes“ auf der Erdoberfläche (gilt die Eigenschaft oder nicht). Neue Ebenen können nun durch *Verschneidung* existierender Ebenen gebildet werden, bzw. allgemeiner, durch die Anwendung spezieller (z.B. boolescher) Kombinations-Operatoren auf Ebenen. Man spricht dann auch von einer *Karten-Algebra (Map Algebra)*. Bei der Verschneidung bzw. Überlagerung von Ebenen handelt es sich um das wichtigste konventionelle Analysewerkzeug. So können wir durch Verschneidung der Ebenen „Wasserflächen“ und „Grünflächen“ sehen, welche Wasserflächen in Grünflächen liegen, etc. Da Geo-Objekte anhand von thematischen Gesichtspunkten auf Ebenen verteilt werden, spricht man von einer *Top Down*-Modellierung, im Gegensatz zu der objektbasierten (und somit flexibleren) *Bottom Up*-Modellierung des Vektormodells.

Das verwendete Datenmodell wird durch die Domäne des GIS bestimmt. So mag die Räumlichkeit der Geo-Objekte für eine GIS-Domäne z.B. durch topologische Information bereits hinreichend detailliert gegeben sein (z.B. für ein Netzwerk-IS).

3.1.1.1. Räumliche Aspekte von Geo-Objekten

Die *räumlichen Aspekte* eines Geo-Objektes werden durch die Geometrie des Objektes repräsentiert. Die Geometrie eines Geo-Objektes ist wiederum durch eine Instanz bzw. ein Datum eines vom GIS angebotenen *räumlichen Daten-*

types (Spatial Data Type, SDTs) gegeben. Solche räumlichen Datentypen sind typischerweise Punkte, Linien, Polygone und Ketten, sowie beliebige Aggregate dieser, etc. Für ein Geo-Objekt, das einen Teich repräsentiert, könnte z.B. ein Polygon verwendet werden. Üblicherweise wird eine sog. *räumliche Datenbank* oder eine objektrelationale Datenbank mit SDTs zur Speicherung der Geometrie verwendet. Insbesondere sind räumliche Indexstrukturen (z.B. R-Bäume) zur Organisation der räumlichen Daten und zur räumlichen Anfragebeantwortung in solchen Datenbanken unverzichtbar.

In konventionellen GIS werden Anfragen meist durch ein räumliches (erweitertes) SQL gestellt. Oftmals sind auch nur sog. ebenen- bzw. layerbasierten Anfragen möglich („Zeige mir die Ebene Infrastruktur!“).

Die geometrische Repräsentation kann als *reichhaltig* bezeichnet werden: Durch die konkrete Geometrie des Polygons sind nicht nur Form und Lage des Objektes repräsentiert, sondern (mit gewisser Genauigkeit) auch räumliche Aspekte wie Fläche und Umfang.

Diese explizite Geometrie ermöglicht hier also bei Bedarf eine Auswertung weiterer räumlicher Aspekte, die nicht explizit (und somit „implizit“) repräsentiert sind. Man spricht hier auch von einer *intrinsischen Repräsentation räumlicher Aspekte* – die Eigenschaften des Repräsentationsmediums, hier der zweidimensionalen Ebene, erlauben die Rekonstruktion räumlicher Zusammenhänge und Eigenschaften durch „Ablesen“ (hierfür sind natürlich u.U. aufwendige Berechnungsmethoden – z.B. zur Berechnung der Fläche eines Polygons – notwendig). Wir können auch sagen, dass es sich um *abgeleitete* Aspekte handelt.

Abgeleitete Aspekte können nicht nur abgeleiteten Eigenschaften, sondern auch abgeleitete Relationen sein. Hierbei sind insbesondere *qualitative* räumliche Relationen zu nennen, die die relative Lage-Beziehungen zwischen zweidimensionalen Objekten zueinander beschreiben (s.u.).

Wir verwenden im Folgenden den Begriff der *Inspektionsmethode*. Eine Inspektionsmethode nutzt die Reichhaltigkeit der Repräsentation, um bei Bedarf abgeleitete Eigenschaften als auch abgeleitete Relationen zu errechnen. Eine *räumliche* Inspektionsmethode kann daher sowohl

- räumliche Relationen zwischen Geo-Objekten errechnen, denn das *räumliche Zueinander der Objekte* ist intrinsisch durch die Geometrie der Objekte repräsentiert, als auch
- räumliche Eigenschaften wie Fläche, Länge, und Umfang errechnen, die ebenfalls intrinsisch repräsentiert sind.

Ein GIS wird ein umfangreiches Arsenal an räumlichen Analyse- und Berechnungsfunktionen bereitstellen, die sich für Inspektionsmethoden nutzen lassen.

3.1 Leitdomäne 1: Geographische Informationssysteme

Prinzipiell können die Bildbereiche solcher Inspektionsfunktionen bzw. -methoden *quantitativ oder qualitativ* sein. Qualitative Beschreibungen verwenden symbolische Beschreibungen bzw. linguistische Kategorien oder sog. *Ordinalskalen*, z.B. „groß“ statt 123.45 m^2 für das Attribut „Fläche“. Naturgemäß sind qualitative Beschreibungen abstrakter und somit „ungenauer“ bzw. „unbestimmter“ als quantitative Beschreibungen. Ein Beispiel für eine quantitative räumliche Relationen wäre eine Entfernungsangabe bzw. Distanz zwischen zwei Geo-Objekten; qualitative räumliche Relationen sind z.B. durch räumliche Präpositionen („in, über, auf, . . .“) in der natürlichen Sprache gegeben. Derartiges qualitatives räumliches Vokabular ist insbesondere für die *Formulierung räumlicher Anfragen* unerlässlich. Offensichtlich lassen sich z.B. räumliche Lagebeziehungen mit einer eindimensionalen Sprache nur qualitativ-approximativ prägnant beschreiben (andernfalls wäre eine visuelle Sprache notwendig). Die Reichhaltigkeit bzw. Granularität der Repräsentation bestimmt letztlich die Granularität der durch die Inspektionsmethoden ableitbaren Aspekte und Relationen. Letztlich ist jede Art von Repräsentation bzw. Modellierung mit Abstraktion verbunden, wodurch die aus der Repräsentation extrahierbare Information im Voraus determiniert wird.

Eine populäre Menge räumlicher Relationen zur Beschreibung der qualitativen relativen Lage zweier (topologischer) *Regionen* zueinander sind durch den sog. *Region Connection Calculus (RCC)* gegeben [Randell u. a. 1992, Cohn u. a. 1997]. Diese Relationen können als „topologisch“ charakterisiert werden (sie können topologisch definiert werden). Die RCC-Relationen sind aus mehreren Gründen in dieser Arbeit interessant:

1. Sie stellen kognitiv adäquates Relationsvokabular zur Beschreibung qualitativer Lagebeziehungen zwischen Geo-Objekten bereit.
2. Sie können daher sowohl als Vokabular in *räumlichen Anfragen* als auch zur *qualitativen Repräsentation* räumlicher Gegebenheiten (z.B. in einer Karte) für das GIS genutzt werden. Eine solche Repräsentation ist ausreichend, wenn das GIS lediglich räumliche Anfragen mit RCC-Relationen beantworten soll. Natürlich wird die Geometrie der Geo-Objekte in einem GIS meistens repräsentiert sein. Ein zusätzliches RCC-Netz (s.u.) kann jedoch auch als *Index* zur Anfragebeantwortung (für Anfragen mit RCC-Relationen) dienen.
3. Sie können auch zum *qualitativen räumlichen Schließen* verwendet werden, wenn die räumlichen Gegebenheiten lediglich *unterbestimmt* oder gar nicht bekannt sind (also keine Karte bzw. konkrete geometrische Szene vorliegt). Diesen Fall haben wir insbesondere vorliegen, wenn *räumliche Konzepte* mit sprachlichen Mitteln intensional beschrieben werden sollen. Beispielsweise wird das räumliche Konzept „Park mit Teich“ in der Vorstellung des Lesers wahrscheinlich zwei prototypische imaginierte Geo-

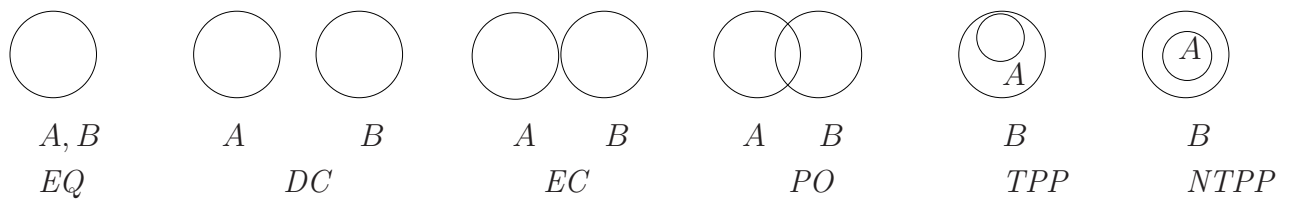


Abbildung 3.1.: RCC8-Basis-Relationen: EQ = Equal, DC = Disconnected, EC = Externally Connected, PO = Partial Overlap, TPP = Tangential Proper Part, NTPP = Non-Tangential Proper Part. Alle Basis-Relationen sind bis auf TPP und NTPP symmetrisch; deren Inverse werden mit TPPI und NTPPI bezeichnet.

Objekte „Teich“ und „Park“ erzeugen, die in einer Enthaltensein-Beziehung zueinander stehen. Die konkrete Form der Geo-Objekte ist aber nicht bekannt. Räumliche Relationen haben gewisse Eigenschaften; z.B. ist die Enthaltensein-Relation transitiv. Für die RCC-Relationen sind entsprechend formalisierte Eigenschaften und Inferenzalgorithmen bekannt, die räumliches Schließen ermöglichen.

Aus diesem Grunde folgt nun ein Abschnitt über RCC und es werden entsprechende Techniken zum qualitativen räumlichen Schließen eingeführt.

3.1.1.1.1. Die Relationen des „Region Connection Calculus (RCC)“ Die sog. *RCC-Basis-Relationen* stellen das elementare Relationsvokabular bereit. Wir stellen hier nicht die vollständige RCC-Theorie und ihre topologischen Grundlagen dar, sondern definieren und illustrieren hier lediglich das in dieser Arbeit benötigte und durch RCC bereitgestellte relationale Vokabular und dessen relationale Eigenschaften. Zudem stellen wir Begriffe und Verfahren aus dem qualitativen räumlichen Schließen mit RCC-Netzen (für unsere Zwecke ausreichend genau) dar.

Definition 50 (RCC-Basis-Relationen) Wir unterscheiden 2 Mengen von RCC-Basis-Relationen, die RCC8- und die (gröberen) RCC5-Relationen:

- $RCC8 = \{DC, EC, PO, EC, TPP, TPPI, NTPP, NTPPI\}$,
- $RCC5 = \{DR, PO, EQ, PP, PPI\}$.

Die RCC8-Basis-Relationen sind in Abb. 3.1 illustriert (hier finden sich auch Erklärungen für die Namen).

RCC bezeichne im Folgenden (je nach Kontext) entweder RCC8 oder RCC5. ■

Eine konkrete räumliche Konstellation zweidimensionaler Regionen kann qualitativ mittels eines *RCC-Netzes* beschrieben werden. Dies ist in Abb. 3.2 illus-

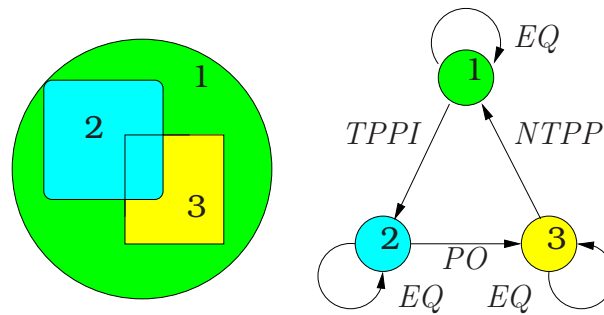


Abbildung 3.2.: Eine räumliche Konstellation und ihr RCC8-Netz

triert. Ein solches anhand einer konkreten geometrischen Konstellation gewonnenes Netz wird nur RCC-Basis-Relationen enthalten. Zudem hat ein solches Netz die Form eines kantenbeschrifteten vollständig vermaschten Graphen, eines sog. K_n -Graphen (n ist die Anzahl der Objekte in der Konstellation). Zudem wird deutlich, dass die Basis-Relationen die sog. *JEPD (Jointly Exhaustive and Pairwise Disjoint)-Eigenschaft* haben: Zwischen je zwei Regionen gilt genau eine RCC-Basis-Relation; d.h., die Basis-Relationen sind erschöpfend und schließen sich gegenseitig aus. Im Allgemeinen lassen wir jedoch beliebige Teilmengen von RCC als RCC-Relationen und somit in RCC-Netzen zu:

Definition 51 (RCC-Relation) Eine RCC-Relation ist eine Teilmenge von RCC. ■

Eine RCC-Relation wird als Disjunktion der verwendeten RCC-Basis-Relationen interpretiert. Durch die Verwendung von Disjunktionen kann größere räumliche Unbestimmtheit ausgedrückt werden (die Relation ist nicht genauer bekannt). Vollständige Unkenntnis bzgl. der vorliegenden Relationen zwischen zwei Regionen kann durch die Disjunktion aller RCC-Basis-Relation (also die Relation RCC) ausgedrückt werden. So sind z.B. die RCC5-Basis-Relation durch RCC8-Relationen wie folgt definiert:

- $DR =_{def} \{DC, EC\}$,
- $PP =_{def} \{TPP, NTPP\}$,
- $PPI =_{def} \{TPPI, NTPPI\}$.

RCC5 ist daher eine *Vergrößerung* von RCC8.

Ein RCC-Netz ist ein mit RCC-Relationen kantenbeschrifteter Graph:

Definition 52 (RCC-Netz) Ein RCC-Netz ist ein kantenbeschrifteter Graph (V, E, L_E) , wobei V die Menge der Knoten ist (welche räumliche Objekte repräsentieren), $E \subseteq V \times V$ die Menge der Kanten, und $L_E : E \rightarrow 2^{RCC}$ eine Kanten-Beschriftungsfunktion, die jeder Kante eine RCC-Relation zuweist. ■

Wird ein RCC-Netz anhand der konkreten Geometrie einer Konstellation abgeleitet (die Polygone sind dann die Regionen), so enthält dieses RCC-Netz offensichtlich nur mit Basis-Relationen beschriftete Kanten. Zudem ist das Netz automatisch *konsistent*. RCC-Netze können aufgrund der JEPD-Eigenschaft inkonsistent werden. Beispielsweise ist folgendes durch eine prädikatenlogische Formelmengende notiertes RCC8-Netz $\mathcal{N}$ intuitiv inkonsistent:

$$\mathcal{N} = \{DC(a, b), NTPPI(b, c), PO(a, c)\}.$$

Dies kann wie folgt gesehen werden: Da a und b disjunkt zueinander sind, b nun aber c enthält, folgt daraus, dass a und c disjunkt sein müssen. Offensichtlich folgt dies aus den spezifischen Eigenschaften der Ebene bzw. der Standardtopologie des metrischen Raumes $(\mathbb{R}^2, \Delta)$, wobei Δ die Standardmetrik ist; die RCC-Relationen haben also bestimmte Eigenschaften, die über die JEPD-Eigenschaft hinausgehen. Der Folgerung $DC(a, c)$ wird jedoch mittels $PO(a, c)$ widersprochen. Aufgrund der JEPD-Eigenschaft kann nun nicht $DC(a, c)$ und $PO(a, c)$ gelten. Das Netz ist daher inkonsistent.

In der sog. *Kompositionstabelle* werden diese weiteren topologischen Eigenschaften der RCC-Relationen festgehalten. Die RCC5-Kompositionstabelle ist in Abb. 3.3 gezeigt; entsprechende Kompositionstabellen existieren auch für andere RCC-Systeme (z.B. RCC8). *Qualitatives räumliches Schließen* wird mit der Kompositionstabelle wie folgt unterstützt: Gegeben drei Knoten a, b, c in einem RCC-Netz mit Kanten $R(a, b)$ und $S(b, c)$, so listet die Kompositionstabelle als Eintrag für $R \times S$ die strengstmögliche RCC-Relation $T = \{T_1, \dots, T_n\}$ auf, die zwischen a und c gelten kann: $R(a, b) \wedge S(b, c) \rightarrow T_1(a, c) \vee \dots \vee T_n(a, c)$. Offensichtlich kann dies als eine Art Relationskomposition verstanden werden (daher der Name). Ein Eintrag in der Kompositionstabelle für die Relationen R und S repräsentiert daher eine Formel der Art

$$\forall x, y, z. R(x, y) \wedge S(y, z) \rightarrow T_1(x, z) \vee \dots \vee T_n(x, z).$$

¹

Die JEPD-Eigenschaft kann prädikatenlogisch durch die beiden Sätze

$$\forall x, y. \bigvee_{R \in \text{RCC}} R(x, y)$$

und

¹Anmerkung: Dieses Axiom wird im Gegensatz zur Variante mit Biimplikation “ $\leftrightarrow$ ” meist als *schwaches* Kompositionsaxiom bezeichnet. Dementsprechend spricht man auch von *schwacher Komposition*.

3.1 Leitdomäne 1: Geographische Informationssysteme

◦	DR(a,b)	PO(a,b)	EQ(a,b)	PPI(a,b)	PP(a,b)
DR(b,c)	*	DR PO PPI	DR	DR PO PPI	DR
PO(b,c)	DR PO PP	*	PO	PO PPI	DR PO PP
EQ(b,c)	DR	PO	EQ	PPI	PP
PP(b,c)	DR PO PP	PO PP	PP	PO EQ PP PPI	PP
PPI(b,c)	DR	DR PO PPI	PPI	PPI	*

Abbildung 3.3.: Die RCC5-Kompositionstabelle

$$\forall x, y. \bigwedge_{R, S \in \text{RCC}, R \neq S} R(x, y) \rightarrow \neg S(x, y)$$

axiomatisiert werden.

Es sind mehrere *Konsistenzbegriffe* für RCC-Netze bekannt. Wir verwenden hier den Begriff der *relationalen Konsistenz*:

Definition 53 (Relationale Konsistenz) Sei $\mathcal{N}$ ein prädikatenlogisch notiertes RCC-Netz, und bezeichne $\text{comp}(R, S)$ den Eintrag in der entsprechenden Kompositionstabelle für $R \times S$. Dann nennen wir $\mathcal{N}$ *relational konsistent* gdw. die Formelmenge

$$\begin{aligned} \mathcal{N} \cup \{ \forall x, y, z. R(x, y) \wedge S(y, z) \rightarrow T_1(x, z) \vee \dots \vee T_n(x, z) \mid \text{comp}(R, S) = \{T_1, \dots, T_n\} \} \cup \\ \{ \forall x, y. \bigvee_{R \in \text{RCC}} R(x, y) \} \cup \\ \{ \forall x, y. \bigwedge_{R, S \in \text{RCC}, R \neq S} R(x, y) \rightarrow \neg S(x, y) \} \cup \\ \{ \forall x. EQ(x, x) \} \end{aligned}$$

erfüllbar ist. ■

Die Technik der *Constraintpropagierung* wird nun verwendet, um die relationale Konsistenz eines RCC-Netzes zu entscheiden [Görz u. a. 2003, Kap. 8, Kap. 10.3.2.2]. Wir wollen den Algorithmus nur informell beschreiben. Der Algorithmus verwendet die Kompositionstabelle zur iterativen, sukzessiven Einschränkung von Kanten-Beschriftungen. Gelte $L_E((x, y)) = \mathcal{R}$ und $L_E((y, z)) = \mathcal{S}$, so wird durch den Algorithmus $L_E((x, z)) := L_E((x, z)) \cap (\bigcup_{R \in \mathcal{R}, S \in \mathcal{S}} \text{comp}(R, S))$ gesetzt. Hierbei sei $\text{comp}(R, S)$ der Eintrag in der Kompositionstabelle für $R \times S$.

Die Kantenbeschriftung $L_E((x, z))$ wird also eingeschränkt. So werden sukzessive alle Kanten bearbeitet, indem eine Schleife so lange durchlaufen wird, bis entweder keine Kanten-Beschriftung weiter eingeschränkt werden kann (ein Fixpunkt wurde berechnet), oder aber eine Kante mit der leeren Menge beschriftet wurde – in diesem Falle ist das Netz inkonsistent. Im ersten Fall gibt der Algorithmus TRUE, im zweiten Fall FALSE zurück.

Gelte – wie im obigen Beispiel – $L_E((a, b)) = \{DC\}$, $L_E((b, c)) = \{NTPPI\}$ und $L_E((a, c)) = \{PO\}$ im RCC-Netz, so setzt der Algorithmus bereits in der ersten Iteration $L_E((x, z)) := L_E((x, z)) \cap \{DC\} = \{PO\} \cap \{DC\} = \emptyset$. Das Netz wird also als inkonsistent erkannt (der Algorithmus gibt FALSE zurück).

Allerdings ist dieses Verfahren nur vollständig für Netze, die ausschließlich Basis-Relationen verwenden. Für allgemeine RCC-Netze muss daher zunächst nichtdeterministisch eine Basis-Konfiguration des Netzes „geraten“ werden, bzw. die möglichen Basis-Konfigurationen müssen mittels eines Suchverfahrens enumeriert werden. Die so aufgezählten Basis-Konfigurationen werden sukzessive auf relationale Konsistenz geprüft. Eine Basis-Konfiguration eines RCC-Netzes $\mathcal{N}$ entsteht wie folgt:

Definition 54 (Basis-Konfiguration eines RCC-Netzes) Sei $\mathcal{N}$ ein RCC-Netz. Eine *Basis-Konfiguration* entsteht aus $\mathcal{N}$, indem für jede Kanten-Beschriftung $L_E((x, y))$ mit $|L_E((x, y))| > 1$ (nichtdeterministisch) eine Basis-Relation $R \in L_E((x, y))$ ausgewählt wird: $L_E((x, y)) := \{R\}$. ■

Es gilt dann:

Faktum 7 Ein RCC-Netz $\mathcal{N}$ ist konsistent gdw. eine Basis-Konfiguration von $\mathcal{N}$ existiert, die relational konsistent ist. ■

Offensichtlich kann das unvollständige Verfahren zunächst verwendet werden, um die Relationen bzw. Kanten-Beschriftungen im Netz maximal einzuschränken, und somit die Anzahl der zu untersuchenden Basis-Konfigurationen eines Netzes möglichst klein zu halten.

Ein weitergehenderer Konsistenzbegriff fragt nach der *räumlichen Realisierbarkeit* eines RCC-Netzes [Grigni u. a. 1995]. Hier wird eine Abbildung von Knoten auf Regionen eines topologischen Raumes bzw. $\mathbb{R}^2$ gefordert, sodass die im Netz notierten Relationen in dieser Struktur gelten. Offensichtlich ist für Netze, die anhand einer konkreten geometrischen Szene bzw. einer Karte errechnet werden, Realisierbarkeit in $\mathbb{R}^2$ automatisch sichergestellt, da die Szene bzw. Karte als Realisierung bzw. geometrisches Modell des Netzes angesehen werden kann.

3.1 Leitdomäne 1: Geographische Informationssysteme

3.1.1.2. Strukturelle Aspekte von Geo-Objekten

Mit *strukturellen Aspekten* meinen wir hier die kompositionale Struktur von Geo-Objekten. Offensichtlich kann ein Polygon als Aggregat von Liniensegmenten aufgefasst werden, welches wiederum zwei Punkte aggregiert, etc. Dementsprechend werden auch objektrelationale Datenbanken genutzt.

3.1.1.3. Thematische Aspekte von Geo-Objekten

Die *thematischen Aspekte* eines Geo-Objektes geben Aufschluss über dessen inhaltliche Bedeutung. Die Thematik ist also primär der Semantik eines Geo-Objektes zuzurechnen und wird als *zweites Standbein der Geoinformatik* bezeichnet.

Zum einen wird mittels der Thematik die *Kategorie (Klasse, Konzept)* des Geo-Objektes festgelegt. Hier sind geographische Kategorien je nach GIS-Domäne zu finden (z.B. „Teich“, „Haus“, „Park“, ...). Oftmals sind diese thematischen Kategorien jedoch weder taxonomisch noch in einem Thesaurus organisiert. Meist werden numerische sog. *Objektschlüssel* verwendet (nicht zu verwechseln mit Schlüssel im relationalen Datenmodell). So kann beispielsweise der Objektschlüssel „5264“ die Bedeutung von „See, nicht schiffbar“ haben. Die Thematik eines Geo-Objektes wird daher oft als numerischer Attributwert repräsentiert. Die möglichen Objektschlüssel werden dann in einer (numerischen) Begriffsliste gehalten, dem sog. *Objektschlüsselkatalog*.

Zum anderen können den Geo-Objekten auch *beliebige Sachdaten* zugeordnet werden, z.B. der Wasserstand eines Sees oder der Name eines Parks. Zur Repräsentation der thematischen Aspekte und zusätzlicher Sachdaten wird oftmals eine relationale bzw. objektrelationale Datenbank verwendet.

Alle thematischen Informationen und Sachdaten für ein Geo-Objekt bilden einen *thematischen Datensatz* für dieses Objekt. Oftmals werden getrennte Datenbanken für die räumlichen und thematischen Datensätze verwendet (z.B. im populären Arc/Info-System). Die Zuordnung eines thematischen Datensatzes zu dem entsprechenden geometrischen Datum in der räumlichen Datenbank ist durch eine 1-zu-1-Relation gegeben und wird durch eindeutige (datenbankübergreifende) Schlüsselattribute (IDs) sichergestellt. Die Expressivität der verwendeten Daten-Beschreibungssprachen ist in konventionellen GIS also beschränkt (es greift hier die gleiche Kritik wie bereits für das relationale Datenbankmodell in Kap. 2.2.9 geäußert).

Das Konzept der Ebenen (Layer) ist primär für das Rastermodell als Datenorganisationsmittel relevant. Im Vektormodell werden jedoch die thematischen Kategorien bzw. Objektschlüssel oftmals nach Themen organisiert bzw. zu „Ebenen“ zusammengefasst (statt einer taxonomischen Organisation). Beispielswei-

se können alle Objektschlüssel zum Thema „Infrastruktur“ („Straße“, „Weg“, „UBahn“, ...) in einer Liste zusammengefasst werden. Eine themenfokussierte Darstellung im GIS wird dann ermöglicht, indem lediglich die Geo-Objekte angezeigt werden, deren Objektschlüssel in der Liste „Infrastruktur“ enthalten sind.

3.1.2. Ein konkreter Geo-Datenbestand – Die digitale Stadtkarte (DISK)

Es stand uns ein konkreter Geo-Datenbestand im SQD-Dateiformat zur Verfügung. Hierbei handelt es sich um eine digitale Stadtkarte, die einen Ausschnitt vom Hamburger Stadtteil Öjendorf darstellt. Die sog. DISK (Digitale Stadtkarte [Freie und Hansestadt Hamburg 1992]) wird vom „Landesbetrieb Geoinformation und Vermessung Hamburg“ kommerziell vertrieben. Die beiden vorliegenden DISK-Karten sind im Anhang A.3 und A.4 zu sehen.

Bei den Daten handelt es sich um generalisierte Vektordaten, deren Thematik durch einfache numerische Objektschlüssel beschrieben ist. Die Beschreibung der Objektschlüssel (und Zuordnung zu Ebenen von thematisch zusammengehörigen Objektschlüsseln) finden sich im „Hamburger Objektschlüsselkatalog“ [Freie und Hansestadt Hamburg 1996]. Die Daten werden normalerweise zur Erstellung von gedruckten Stadtkarten verwendet, daher sind sie *kartografisch generalisiert*; sie haben somit nicht die Genauigkeit von Katasterdaten. Für unsere Zwecke sind die Daten jedoch ideal, aufgrund der *thematischen Reichhaltigkeit*. Der Hamburger Objektschlüsselkatalog ist reich an interessanten Objektschlüsseln, wie „Wohnfläche“, „Öffentliches Gebäude“, „Acker“, „U-Bahn-Station“, die sämtlich numerisch kodiert sind. Teilweise sind auch hochspezifische Begriffe zu finden, z.B. „Friedhof für Nichtchristen“ oder „Teich, nicht schiffbar“.

Im Objektschlüsselkatalog ist keine explizite Begriffshierarchie bzw. Taxonomie vorhanden. Einige taxonomische Beziehungen sind jedoch implizit anhand der (intendierten) Semantik der Konzepte erkennbar, denn sicherlich sollte „See, nicht schiffbar“ ein Unterkonzept von „See“ sein, insbesondere auch unter der Perspektive, dass unser Hauptanliegen ontologiebasierte Anfragebeantwortung ist. Wir wollen daher nicht nur Instanzen von „Teich“ als Treffer einer Anfrage „Welche Geo-Objekte mit Thematik Teich gibt es?“ zurückliefern, sondern auch Instanzen von „Teich, nicht schiffbar“.

Im Folgenden betrachten wir ausschließlich die DISK und nehmen diese Domäne als prototypisch für die Domäne „(vektorbasierte) Geographische Informationssysteme“ an.

3.1.3. Terminologie – Ein DL-inspirierter Beschreibungsrahmen für ontologiebasierte GIS

Wie bereits erwähnt, übernehmen wir die in DL-Systemen zu findende Aufteilung in eine *intensionale*, eine *extensionale*, und eine *Anfrage-Komponente*. Wir stellen hier den Beschreibungsrahmen im DISK-Kontext vor. Prinzipiell ist er jedoch auch für andere Domänen nutzbar.

3.1.3.1. Die intensionale Komponente

Die *intensionale Komponente* kann als Analogon einer TBox eines DL-Systemes verstanden werden. Hier werden die Konzepte bzw. das Vokabular des ISs intensional beschrieben. Wir nehmen die (generalisierende) Sichtweise ein, dass diese Konzept-Beschreibungssprachen durch Teilmengen von $\mathcal{L}_{FOP\mathcal{L}}$ formalisiert werden können. Wir behalten die Sichtweise aus der DL bei, dass Konzepte einstellige, und Rollen zweistellige Prädikate sind. Mit der Konzept-Beschreibungssprache sollten idealerweise sowohl räumliche als auch thematische Aspekte von Geo-Objekten rein intensional beschrieben werden können.

Auch die Basis-Dienste in der intensionalen Komponente sind DL-inspiriert: Erfüllbarkeit von und Subsumption zwischen Konzepten soll entschieden werden, und die Taxonomie automatisch berechnet werden können.

Kommen sowohl räumliche als auch thematische Anteile in einem Konzept vor, so sprechen wir von einem *räumlich-thematischen Konzept*. Andernfalls von einem *thematischen* bzw. *räumlichen* Konzept.

Ein Konzept soll üblicherweise eine *Menge* von Individuen bzw. Objekten intensional beschreiben bzw. definieren (= Extension des Konzeptes). Für die räumlichen Konzepte ergibt sich dann zwingend, dass nicht alle räumlichen Aspekte durch die Definition festgelegt werden dürfen, da andererseits keine Menge bzw. Klasse, sondern lediglich eine Instanz beschrieben wird (Generalisierungsproblem). In der Regel ist es daher nicht sinnvoll, z.B. die Geometrie der intensional repräsentierten Objekte durch die Definition festzulegen. Das Generalisierungsproblem kann durch die Verwendung qualitativer Beschreibungen aufgrund der *symbolischen Abstraktheit* qualitativer Beschreibungen umgangen werden. Dies ist jedoch nur für ausgewählte räumliche Aspekte möglich (z.B. lässt sich der räumliche Aspekt „Form“ qualitativ schlecht beschreiben).

Eine Inspektion des Objektschlüsselkatalogs zeigt, dass insbesondere qualitative räumliche Relationen bzw. Definition mit räumlichen Präpositionen für die DISK relevant sind. Ein sinnvolles Konzept für die DISK-Ontologie ist z.B. „Teich im Park“. Wir können es als „Teich, der in einem Park enthalten ist“ definieren.

Wir hatten in Kap. 3.1.1.1.1 Techniken des *qualitativen räumlichen Schließens* anhand des RCC-Kalküls erläutert. Offensichtlich entspricht die Präposition „enthalten in“ der RCC8-Relation $\{NTPPI, TPPI\}$ bzw. der RCC5-Basis-Relation PPI . Wie erläutert, haben RCC-Relationen gewisse Eigenschaften, die beim Schließen berücksichtigt werden müssen.

Unterbestimmtheit bzgl. der thematischen Aspekte ist in den Beschreibungen bzw. Definitionen hingegen einfach z.B. durch Disjunktionen formulierbar.

3.1.3.2. Die extensionale Komponente

Die *extensionale Komponente* kann als Analogon einer ABox eines DL-Systems verstanden werden. Hier werden die Individuen bzw. Geo-Objekte repräsentiert und verwaltet. Dementsprechend wird eine Instanzen-Beschreibungssprache zur Konstruktion von Individuen und zur Etablierung von Beziehungen zwischen diesen benötigt, ähnlich den ABox-Assertionen. Instanzen verstehen wir als prädikatenlogische Konstanten; wir nehmen an, dass diese Instanzen ebenfalls prädikatenlogisch beschrieben werden können. Die UNA ist optional.

Die Instanzen-Beschreibungssprache soll sowohl den räumlichen als auch den thematischen Aspekten der Geo-Daten gerecht werden. Ein Bezug zur intensionalen Komponente ist durch gemeinsam verwendetes (räumliches und/oder thematisches) Vokabular bzw. Konzepte gegeben.

Bezüglich der räumlichen Aspekte sind sowohl qualitative als auch quantitative, z.B. geometrische oder metrische Beschreibungen denkbar. Wir gehen davon aus, dass die konkrete Geometrie der Objekte gegeben ist, denn ein DISK-Datenbestand ist letztlich eine „konkrete Karte“. Dies ist ein fundamentaler Unterschied zu den Konzept-Beschreibungen in der intensionalen Komponente. Dementsprechend sprechen wir (etwas vereinfachend) auch von einem „Datenbestand“. Disjunktionen oder Unterbestimmtheit bzgl. räumlicher Aspekte ist hier also zunächst nicht gegeben, allerdings ist dies für thematisches Aspekte möglich (tatsächlich gibt es „unklassifizierte“ Geo-Objekte in diesem Datenbestand, also solche ohne Thematik!).

Die Geometrie der Objekte kann durch prädikatenlogische Formeln beschrieben werden (s. Details in Kap. 4).

Es ist jedoch für die ontologiebasierte Anfragebeantwortung u.U. nicht notwendig, die Geometrie zu beschreiben bzw. zu repräsentieren. Sie kann z.B. verworfen werden, wenn lediglich thematische und qualitative räumliche Eigenschaften und Relationen in den Anfragen und in der intensionalen Komponente verwendet werden. Dann ist auch eine rein symbolische Repräsentation ausreichend. Offensichtlich wird die Geometrie der Objekte jedoch benötigt, wenn z.B. Anfrageergebnisse grafisch (in einer Karte) dargestellt werden sollen.

3.1 Leitdomäne 1: Geographische Informationssysteme

Da durch die Geometrie eine *reichhaltigen Repräsentation* gegeben ist, drängt sich die Idee auf, diese doch zunächst beizubehalten (nicht zuletzt auch unter dem Gesichtspunkt der Ergebnispräsentation), und stattdessen *Inspektionsmethoden* zu nutzen, die die jeweils zur Lösung einer Aufgabe benötigten *ausgewählten räumlichen Aspekte* aus dieser reichhaltigen Repräsentation heraus projizieren.

3.1.3.3. Die Anfrage-Komponente

Die Anfrage-Komponente stellt eine *Anfragesprache* bereit und präsentiert die Anfrageergebnisse in grafischer (möglichst interaktiver) Form (in diesem Falle durch Karten o.ä.).

In unserer logikbasierten Betrachtungsweise sind die einfachsten Anfragen die *Instanzen-Retrieval-Anfragen*. In diesen werden räumlich-thematische *Anfragekonzepte* verwendet. Es werden dann alle Geo-Objekte zurückgeliefert, die die im räumlich-thematischen Anfragekonzept C definierten Bedingungen erfüllen, also Treffer sind, wie in Kap. 2 definiert:

$$\text{concept\_instances}(C) = \{ i \mid i \in \text{inds}(\mathcal{E}), (\mathcal{I}, \mathcal{E}) \models C(i) \}$$

(hier bezeichnet $\mathcal{E}$ die extensionale, und $\mathcal{I}$ die intensionale Komponente).

Stammt ein solches Anfragekonzept oder eines seiner Unterkonzepte aus der Ontologie, so wird *ontologiebasierte Anfragebeantwortung* ermöglicht. Die Geo-Objekte in der extensionalen Komponente sollen daraufhin geprüft werden, ob sie das Anfragekonzept bzw. die Anfrage erfüllen. Da das Anfragekonzept Vokabular aus der Ontologie enthalten kann, ist es Aufgabe der Anfragebeantwortungsmaschine, *intensionale Konzept-Beschreibungen* und *extensionale Instanzen-Beschreibungen* miteinander zu verrechnen, um einen Treffer festzustellen. Dies wird insbesondere aufgrund des potentiellen Vorhandenseins von *unvollständiger Information* (z.B. in Form von Disjunktionen) in der extensionalen Komponente erschwert. Zudem könnten auch im Anfragekonzept bzw. der referenzierten Ontologie Disjunktionen auftreten. Anfragebeantwortung erfordert also Deduktion.

Die räumlichen Aspekte einer konkreten Karte sind jedoch vollständig bekannt – wir hatten diese daher als räumliche „Daten“ charakterisiert. Räumliche Unterbestimmtheit liegt also nicht vor. Dementsprechend kann eine Karte auch als eine konkrete (mathematische) Struktur gedeutet werden. Anfragebeantwortung bzgl. räumlichen Aspekte kann dann als *Model Checking* in dieser Struktur formalisiert werden (in diesem Fall ist $(\mathcal{I}, \mathcal{E}) \models C(i)$ also einfacher zu entscheiden, sofern C nur räumliche Aspekte referenziert).

Für die DISK definieren wir eine Reihe von *Referenz-Anfragen*. Diese Anfragen können als typische ontologiebasierte Anfragen an eine solche Stadtkarten-

Domäne (wie die DISK) verstanden werden. Sie verwenden reichhaltiges thematisches und räumliches Vokabular. Die Anfragen referenzieren unterschiedliche räumliche und thematische Aspekte; beispielsweise qualitative räumliche Relationen, metrische Aspekte, Entfernungsbeziehungen, etc.:

Q1 „Zeige mir alle Wohngebiete!“

Diese Anfrage adressiert ausschließlich thematische Aspekte. *Wohngebiet* ist als thematisches Konzept in der Ontologie definiert.

Q2 „Zeige mir alle Wohngebiete, die an einer Autobahn liegen!“

Die Anfrage referenziert räumliche und thematische Aspekte. Wir können „liegt an“ in *EC* oder $\{EC, PO\}$ übersetzen. Diese Information kann ebenfalls in der Ontologie abgelegt werden.

Q3 „Zeige mir alle Wohngebiete, die an einer Grünfläche liegen, welche einen Teich oder See enthält! Das Wohngebiet soll zudem an einem Gewerbegebiet liegen (zum Einkaufen).“

Die Präposition „enthält“ können wir in *PPI* übersetzen. Hier wird das Vorliegen einer komplexen räumlichen Objektkonfiguration der Art $EC(x, y) \wedge PPI(y, z) \wedge EC(x, w)$ gefordert.

Q4 „Zeige mir alle Wohngebiete, die mindestens 8 Gebäude enthalten, und deren Fläche größer als 30000 qm ist!“

Hier wird eine metrische Bedingung verwendet, nämlich dass die Fläche des Objektes größer als 30000 qm sein soll. Zudem wird gefordert, dass *mindestens 8* Gebäude enthalten sein sollen.

Q5 „Zeige mir alle Wohngebiete, die NICHT an Industrie- oder Gewerbegebiete angrenzen!“

Hier wird offensichtlich verlangt, dass eine bestimmte räumlich-thematische Gegebenheit *nicht* vorliegt.

Es stellt sich die Frage nach der Semantik des „nicht“ in dieser Anfrage – Soll „nicht“ hier *beweisbar nicht an ein Industrie- oder Gewerbegebiet angrenzend* bedeuten, oder aber *anhand der Karte nicht ableitbar, dass das Wohngebiet an ein Industrie- oder Gewerbegebiet angrenzt*?

Beide Interpretation sind je nach Anfrageintention sinnvoll. Der ersten Interpretation liegt eine OWA-, und der zweiten eine CWA-Semantik (bzw. eine „Negation as Failure“-Semantik) zugrunde.

Q6 „Welche Gewässer in Naturschutzgebieten werden von Bächen gespeist, die durch oder an Gewerbegebieten mit Fabriken vorbei fließen?“

Solche Gewässer sollten vielleicht gut überwacht werden (Umweltmonitoring). „Gespeist“ bzw. „fließt in“ übersetzen wir in $\{PO, EC\}$ (ebenso wie

3.1 Leitdomäne 1: Geographische Informationssysteme

„aus“, „durch“ oder „an“; es ist klar, dass von der Flussrichtung dann abstrahiert wird).

Q7 „Welche Wohngebiete enthalten in einer Umgebung von 750 m Radius eine UBahn-Station?“

Hier wird eine *qualitativ-metrische Relation* verwendet („im Radius von 750 m enthalten“).

Offensichtlich ist man in den meisten Fällen im Anfrageergebnis nicht nur an einem primären Karten-Objekt interessiert, sondern an der gesamten in der Anfrage geforderten erfüllenden *Konstellation* von Karten-Objekten. Im Anfrageergebnis von **Q7** sollte nicht nur das Wohngebiet, sondern auch die UBahn-Station ersichtlich sein. Wir benötigen dann Anfragesprachen, die tupelwertige Ergebnisse liefern können, anders als z.B. *concept_instances*, welches nur Mengen von Karten-Objekten liefern kann. Wir wollen dies als ein wesentliches Defizit der einfachen Anfragekonzepte (mit nur einer freien Variablen) bzw. von *concept_instances* festhalten.

3.1.4. Die Mehrwertdienste eines ontologiebasierten GIS

In einem deduktiven ontologiebasiertem GIS sind folgende Mehrwertdienste denkbar. Die Frage der Realisierbarkeit bzw. Umsetzbarkeit dieser Dienste beleuchten wir weiter unten. Wir werden dabei auch Probleme feststellen.

Die folgenden Dienste sind nicht GIS-spezifisch, sondern gelten auch für ontologiebasierte ISe in anderen Domänen:

- *Ontologiebasierte Anfragebeantwortung bietet eine nutzer- und domänen-adäquate Sicht auf die Datenbestände.* Es kann Vokabular aus der Ontologie verwendet werden. Im Idealfall benötigt der Nutzer keine Kenntnisse des vom IS verwendeten Datenmodells, sofern er nur ausreichend über die Domäne informiert ist. Ontologiebasierte Anfragen sind also nutzerfreundlich. Wir verdeutlichen dies für die DISK:

Im Objektschlüsselkatalog der DISK wird sehr spezielles domänenspezifisches Vokabular verwendet, z.B. „Friedhof für Nichtchristen“ und „Teich, nicht schiffbar“. Dieses Vokabular ist umgangssprachlich nicht gebräuchlich. Dementsprechend schwierig würde sich die Anfrageformulierung für Endnutzer eines DISK-ISs gestalten – die Anfrage „Welche Friedhöfe gibt es?“ wird nicht vollständig beantwortet, da der Nutzer nicht auch nach „Friedhöfen für Nichtchristen“ gefragt hat (und auf diese Idee käme er wohl auch nicht). Die Ontologie kann jedoch zusätzliches allgemeingebräuchliches Vokabular hinzufügen (z.B. sowohl durch generalisierende Oberkonzepte, als auch durch Synonyme). Durch eine tiefere Modellie-

rung als im Objektschlüsselkatalog werden daher taxonomische Beziehung explizit gemacht bzw. vom System inferiert. Ontologiebasierte Anfragebeantwortung wird daher *vollständiger* sein (höher recall-Werte liefern).

- *Die Ontologie kann auf Kohärenz und Erfüllbarkeit geprüft werden (s. Def. 35, S. 69; dort sind diese Begriffe für TBoxen definiert).* Diese Dienste können auch als *Schemakonsistenzprüfung* verstanden werden. Modellierungsfehler und -defizite im Schema des ISs als auch implizierte Subsumptionsbeziehungen können erkannt werden. Dies wird wesentlich zur *Qualitätssicherung* des ISs beitragen.

In der DISK-Domäne könnte das Konzept „Naturschutzgebiet“ beispielsweise implizieren, dass keine Häuser in diesem Gebiet gebaut werden dürfen, bzw. alle Objekte, die in einem solchen Gebiet enthalten sind, dürfen keine Häuser sein. Ein Konzept wie „Villa im Naturschutzgebiet“ sollte dann automatisch als inkohärent bzw. inkonsistent erkannt werden.

- *Der „Datenbestand“ in der extensionalen Komponente kann auf Konsistenz (Erfüllbarkeit) geprüft werden (s. Def. 41, S. 74; dort ist der Begriff für ABoxen definiert).* Dieser Dienst kann auch *Datenkonsistenzprüfung* genannt werden.

Ist ein DISK-Objekt beispielsweise als Instanz des Konzeptes „Naturschutzgebiet“ bekannt, steht jedoch mit einer Instanz des Konzeptes „Villa“ in einer räumlichen Enthaltensein-Beziehung, so ist der Datenbestand inkonsistent bzgl. der intensionalen Komponente bzw. Ontologie. Die *Qualität* des Datenbestands kann somit automatisch gesichert werden.

- *Anfragen können auf Erfüllbarkeit geprüft werden.* Intuitiv ist eine ontologiebasierte Anfrage erfüllbar gdw. es einen Datenbestand gibt, sodass die Anfrage auf diesem eine nichtleere Antwort liefert. *Unerfüllbare Anfragen* können somit niemals Ergebnisse liefern und verschwenden daher CPU-Zeit.

Beispielsweise kann bzgl. der DISK-Ontologie erkannt werden, dass die Anfrage „Naturschutzgebiet mit Villa“ unerfüllbar ist.

- *Anfragen können auf Subsumption (Query Subsumption oder Query Containment) geprüft werden.* Eine Subsumptionsbeziehung zwischen zwei Anfragen q_1 und q_2 liegt genau dann vor, in Zeichen $q_1 \sqsubseteq q_2$, wenn q_1 und q_2 die gleiche Ontologie referenzieren, und wenn für alle denkbaren Datenbestände (die die verwendete Ontologie referenzieren) das Anfrageergebnis von q_2 stets eine Obermenge des Ergebnisses von q_1 ist. Die Anfrage q_2 wird die subsumierende, und q_1 die subsumierte Anfrage genannt.

Festgestellte Subsumptionsbeziehungen können bei der Anfragebeantwortung ausgenutzt werden. So können zur Beantwortung der Anfrage

q evtl. bereits zuvor berechnete und (in einem Cache) gespeicherte Ergebnisse einer *subsumierten* Anfrage sofort zurückgegeben werden – die Vereinigung der Anfrageergebnisse aller subsumierten Anfragen stellt eine Teilmenge des Ergebnisses von q dar. Die evtl. gespeicherten Antworten von *subsumierenden* Anfragen können ebenfalls genutzt werden (als sog. „Generatoren“). Speziell die subsumierten Anfragen können also „billige Antworttupel“ bereitstellen, aber zu dem Preis, dass die Anfrageergebnisse (anfrageindiziert) gespeichert werden müssen. Wir kommen hierauf in Kap. 4.2.11 zurück.

3.1.5. Beschreibungslogiken mit RCC-Relationen

Bei der Diskussion der intensionalen Komponente hatten wir dargestellt, dass für eine DISK-Ontologie qualitative räumlich-thematische Konzepte wie „Ein Park mit Teich“ wünschenswert wären. Aus diesem Grund hat der Autor untersucht, wie eine DL zur Modellierung räumlich-thematischer Konzepte verwendet werden kann, wenn die räumlichen Aspekte *ausschließlich* durch RCC-Relationen beschrieben werden. Unter der Annahme, dass Rollen für RCC-Basis-Relationen existieren, könnte dieses Konzept wie folgt definiert werden:

$$park\_mit\_teich \triangleq park \sqcap \exists PPI.teich$$

Dieser Vorschlag basiert auf den Arbeiten [Cohn 1993, Lutz 1998]. Eine generellere, verwandte Sprache namens $\mathcal{ALCRP}(\mathcal{D})$ hatte sich bereits als unentscheidbar herausgestellt [Haarslev u. a. 1998; 1999].

Die besonderen Eigenschaften der RCC-Relationen sind durch die JEPD-Eigenschaft und die Kompositionstabelle(n) gegeben. Wir hatten dargestellt, dass ein Eintrag in der Kompositionstabelle prädikatenlogisch einer Formel der Art

$$\forall x, y, z. S(x, y) \wedge T(y, z) \rightarrow (R_1(x, z) \vee \dots \vee R_n(x, z))$$

entspricht. In DL-Notation können wir diesen Satz mittels eines *komplexen Rolleninklusionsaxioms* der Form

$$S \circ T \sqsubseteq R_1 \sqcup \dots \sqcup R_n$$

notieren (eine Menge solcher Axiome wird in der TBox oder RBox abgelegt). Semantisch wird durch ein Axiom $S \circ T \sqsubseteq R_1 \sqcup \dots \sqcup R_n$ daher

$$S^{\mathcal{I}} \circ T^{\mathcal{I}} \subseteq R_1^{\mathcal{I}} \cup \dots \cup R_n^{\mathcal{I}}$$

in den Modellen $\mathcal{I}$ erzwungen.

Der Autor hat daher zunächst das Erfüllbarkeitsproblem von $\mathcal{ALC}$ in Kombination mit einer Menge von Rollenaxiomen der Art $S \circ T \sqsubseteq R_1 \sqcup \dots \sqcup R_n$ untersucht. Die resultierende Sprache wurde $\mathcal{ALC}_{\mathcal{RA}^\oplus}$ genannt ($\mathcal{RA}$ steht für *Role Axioms*). Die Sprache stellte sich als unentscheidbar heraus [Wessel 2000b]. Um die JEPD-Eigenschaft sicherzustellen, wurde $\mathcal{ALC}_{\mathcal{RA}^\oplus}$ mit *disjunkten Rollen* unter-

sucht. Diese DL wurde $\mathcal{ALC}_{\mathcal{RA}}$ genannt. Auch $\mathcal{ALC}_{\mathcal{RA}}$ stellte sich als unentscheidbar heraus; der $\mathcal{ALC}_{\mathcal{RA}^\oplus}$ -Beweis ist fundamental anders und konnte daher nicht einfach „übernommen“ werden [Wessel 2001b]. Wir hatten in Kap. 2.2.6 angemerkt, dass Rollendisjunktheit nicht modallogisch axiomatisierbar ist. Es wurde jedoch auch eine entscheidbare Klassen von Rollenaxiomen gefunden. Diese Sprache wurde $\mathcal{ALC}_{\mathcal{RASG}}$ genannt [Wessel 2000a]. Leider ist die praktische Eignung von $\mathcal{ALC}_{\mathcal{RASG}}$ für diese Domäne eingeschränkt.

Da die $\mathcal{ALC}_{\mathcal{RA}}$ - und $\mathcal{ALC}_{\mathcal{RA}^\oplus}$ -Unentscheidbarkeitsbeweise sehr umfangreiche und komplexe Mengen von Rolleninklusionsaxiomen benötigen, war nicht klar, ob nicht die Mengen von Rollenaxiomen, die durch Umsetzung der RCC-Kompositionstabellen in Rollenaxiome entstehen, nicht doch zu entscheidbaren DLs führen. Zudem wurden inverse Rollen, also $\mathcal{ALCI}$, als Basis für eine „RCC-Beschreibungslogik“ notwendig, damit die Semantik der RCC-Relationen vollständiger eingefangen werden konnte. Der Autor definierte und untersuchte daher die sog. $\mathcal{ALCI}_{\text{RCC}}$ -Familie von DLs [Wessel 2003a]. Es handelt sich um eine Familie, da die RCC-Systeme austauschbar sind. Das Konzept $\text{park} \sqcap \exists \text{PPI}.\text{teich}$ ist beispielsweise ein $\mathcal{ALCI}_{\text{RCC5}}$ -Konzept. Wir hatten die RCC-Relationssysteme RCC5 und RCC8 bereits diskutiert. Der Autor hat noch weitere Vergrößerungen definiert und untersucht, nämlich RCC3, RCC2 und RCC1 (s. Abb. 3.5) [Wessel 2003a]. In der $\mathcal{ALCI}_{\text{RCC}}$ -Familie wird die Menge der Rollennamen $\mathcal{N}_{\mathcal{R}}$ nun auf die entsprechende Menge der verwendeten RCC-Basis-Relationen des entsprechenden RCC-Systems eingeschränkt (aus Gründen der Einfachheit erlauben wir hier lediglich RCC-Rollen, natürlich sind auch beliebige nicht-RCC-Rollen zusätzlich denkbar):

- $\mathcal{ALCI}_{\text{RCC8}}: \mathcal{N}_{\mathcal{R}} = \{DC, EC, PO, EQ, TPP, TPPI, NTPP, NTPPI\}.$
- $\mathcal{ALCI}_{\text{RCC5}}: \mathcal{N}_{\mathcal{R}} = \{DR, PO, EQ, PP, PPI\}.$
In $\mathcal{ALCI}_{\text{RCC8}}$ sind PP und PPI definierbar als
 $PP =_{\text{def}} \{TPP, NTPP\}$ und $PPI =_{\text{def}} \{TPPI, NTPPI\}.$
- $\mathcal{ALCI}_{\text{RCC3}}: \mathcal{N}_{\mathcal{R}} = \{DR, ONE, EQ\}.$
In $\mathcal{ALC}_{\text{RCC5}}$ ist ONE definierbar als $ONE =_{\text{def}} \{PP, PPI, PO\}.$
- $\mathcal{ALCI}_{\text{RCC2}}: \mathcal{N}_{\mathcal{R}} = \{DR, O\}.$
In $\mathcal{ALCI}_{\text{RCC3}}$ ist O definierbar als $O =_{\text{def}} \{ONE, EQ\}.$
- $\mathcal{ALCI}_{\text{RCC1}}: \mathcal{N}_{\mathcal{R}} = \{SR\}.$
In $\mathcal{ALCI}_{\text{RCC2}}$ ist SR definiert als $SR =_{\text{def}} \{DR, O\}.$

Die semantischen Eigenschaften bzw. Forderungen an die Interpretationen dieser Rollen sind folgende:

- **RCC1:**
 - SR (*spatially related*): reflexiv, symmetrisch, transitiv.

3.1 Leitdomäne 1: Geographische Informationssysteme

RCC1	SR(a,b)	RCC2	DR(a,b)	O(a,b)
SR(b,c)	*	DR(b,c)	*	*
		O(b,c)	*	*

RCC3	DR(a,b)	ONE(a,b)	EQ(a,b)
DR(b,c)	*	DR ONE	DR
ONE(b,c)	DR ONE	*	ONE
EQ(b,c)	DR	ONE	EQ

RCC8	DC(a,b)	EC(a,b)	PO(a,b)	TPP(a,b)	NTPP(a,b)	TPPI(a,b)	NTPPI(a,b)	EQ(a,b)
DC(b,c)	*	DC EC PO TPPI NT- PPI	DC EC PO TPPI NT- PPI	DC	DC	DC EC PO TPPI NT- PPI	DC EC TPPI NT- PPI	DC
EC(b,c)	DC EC PO TPP NTPP	DC EC PO TPP TPPI EQ	DC EC PO TPPI NT- PPI	DC EC	DC	EC PO TPPI NT- PPI	PO TPPI NT- PPI	EC
PO(b,c)	DC EC PO TPP NTPP	DC EC PO TPP NTPP	*	DC EC PO TPP NTPP	DC EC PO TPP NTPP	PO TPPI NT- PPI	PO TPPI NT- PPI	PO
TPP(b,c)	DC EC PO TPP NTPP	EC PO TPP NTPP	PO TPP NTPP	TPP NTPP	NTPP	PO EQ TPP TPPI	PO TPPI NT- PPI	TPP
NTPP(b,c)	DC EC PO TPP NTPP	PO TPP NTPP	PO TPP NTPP	NTPP	NTPP	PO TPP NTPP	PO TPPI TPP NTPP NT- PPI EQ	NTPP
TPPI(b,c)	DC	DC EC	DC EC PO TPPI NT- PPI	DC EC PO TPP TPPI EQ	DC EC PO TPP NTPP	TPPI NT- PPI	NTPPI	TPPI
NTPPI(b,c)	DC	DC	DC EC PO TPPI NT- PPI	DC EC PO TPPI NT- PPI	*	NTPPI	NTPPI	NTPPI
EQ(b,c)	DC	EC	PO	TPP	NTPP	TPPI	NTPPI	EQ

Abbildung 3.4.: Die RCC-Kompositionstabellen – Ein Eintrag * bezeichnet die volle Disjunktion bzw. Menge aller RCC-Relationen.

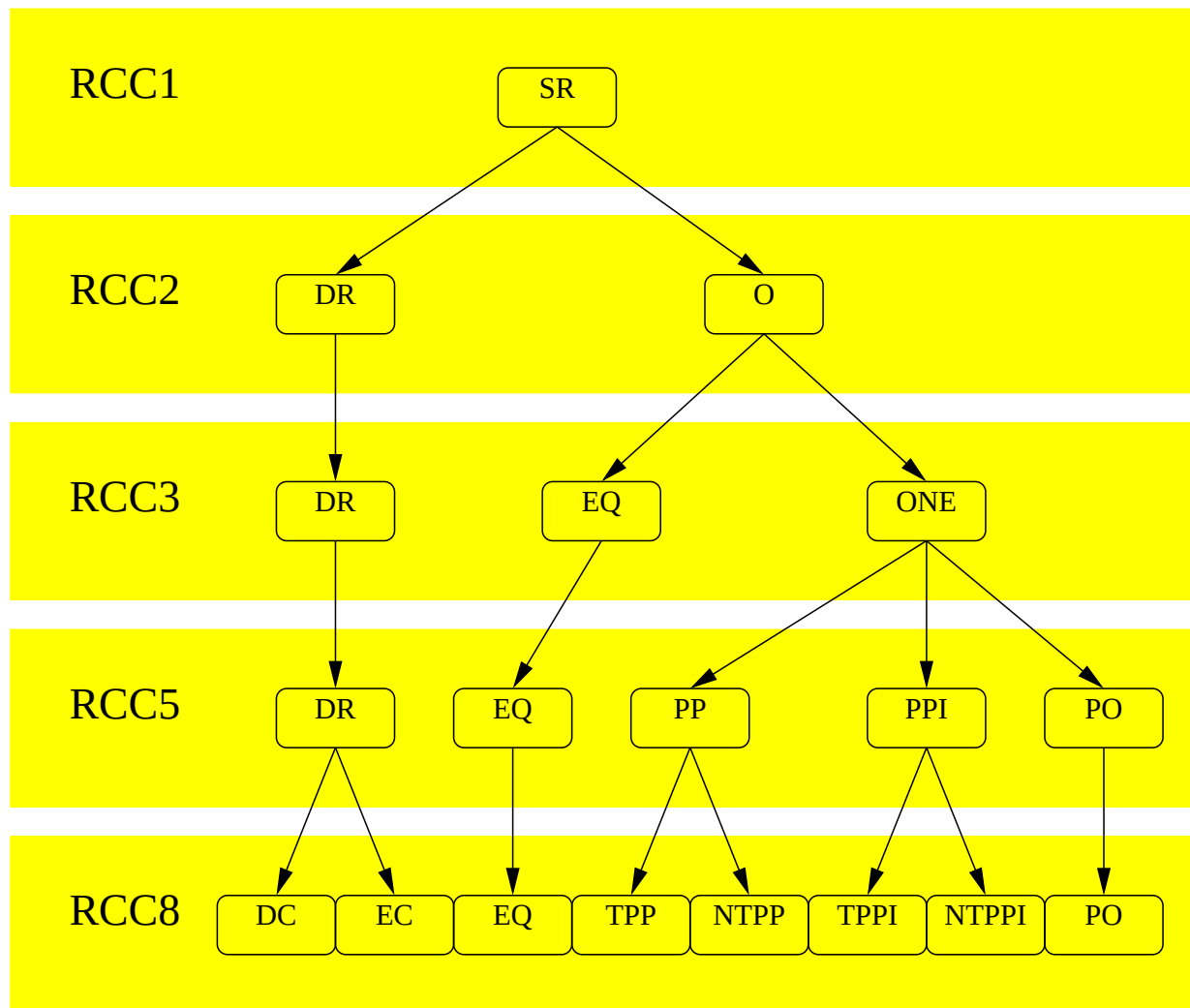


Abbildung 3.5.: Die für die $\mathcal{ALCI}_{RCC}$ -Familie betrachteten RCC-Relationensysteme. Für RCC3 wird die RCC5-Relation $\{PP, PPI, PO\}$ zur Basis-Relation *ONE* („Overlapping but not equal“); die RCC3-Relation $\{ONE, EQ\}$ wird zur RCC2-Basis-Relation *O* („Overlapping“), und schließlich wird $\{O, DR\}$ zu *SR* („Spatially Related“) in RCC1 kollabiert.

- *RCC2*:
 - *O* (*overlapping*): reflexiv, symmetrisch.
 - *DR* (*discrete*): symmetrisch, irreflexiv.
- *RCC3*:
 - *EQ* (*equal*): reflexiv, symmetrisch, transitiv.

3.1 Leitdomäne 1: Geographische Informationssysteme

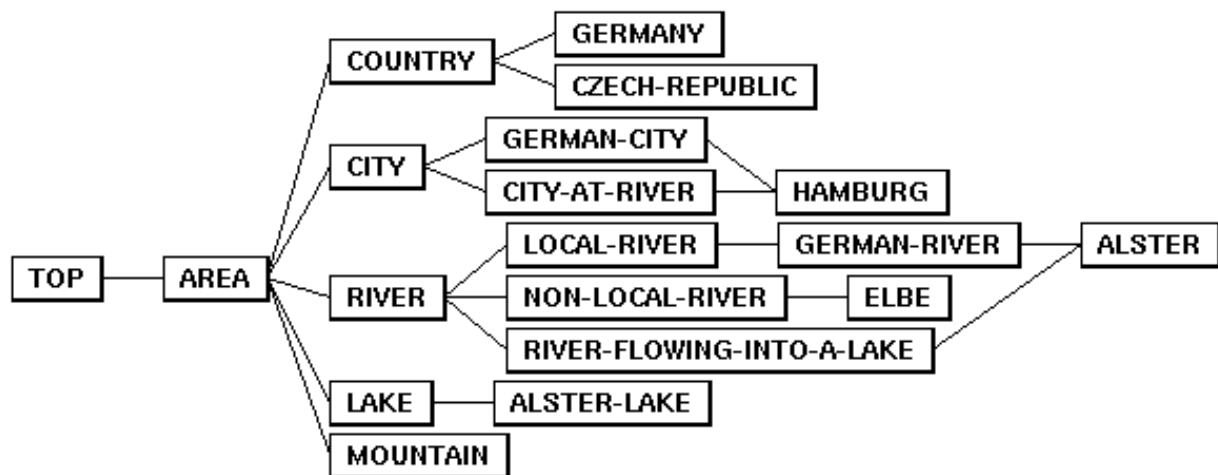


Abbildung 3.6.: Taxonomie der Beispiel-TBox.

- *ONE* (overlapping, but not equal): symmetrisch, irreflexiv.
- *DR* (discrete): symmetrisch, irreflexiv.
- *RCC5*:
 - *EQ* (equal): reflexiv, symmetrisch, transitiv.
 - *PO* (partial overlapping): symmetrisch, irreflexiv.
 - *DR* (discrete): symmetrisch, irreflexiv.
 - *PP* (proper part): asymmetrisch, irreflexiv, transitiv.
 - *PPI* (proper part inverse): asymmetrisch, irreflexiv, transitiv.
- *RCC8* hat 8 Rollen. Alle Rollen außer *EQ* sind irreflexiv. *EQ* ist reflexiv. Alle Rollen außer *TPP*, *TPPI*, *NTPP*, *NTPPI* sind symmetrisch. *TPP* und *TPPI* sind nicht transitiv, aber *NTPP* und *NTPPI*.

Die entspr. Kompositionstabellen sind in Abb. 3.4 angegeben; die *RCC5*-Tabelle wurde bereits in Abb. 3.3 dargestellt.

In [Wessel 2002; 2003a] wird gezeigt, dass $\mathcal{ALCI}_{RCC1}$ bis einschließlich $\mathcal{ALCI}_{RCC3}$ entscheidbar sind. Leider sind die *RCC3*-Basis-Relationen zu abstrakt für eine *DISK*-Ontologie.

$\mathcal{ALCI}_{RCC5}$ ist hingegen eine praxisrelevante Sprache. Zur Illustration geben wir eine $\mathcal{ALCI}_{RCC5}$ -Beispiel-Geo-TBox an und demonstrieren eine nichttriviale Subsumptionsbeziehung:

<i>country</i>	$\sqsubseteq$	<i>area</i>
<i>city</i>	$\sqsubseteq$	<i>area</i>
<i>river</i>	$\sqsubseteq$	<i>area</i>
<i>lake</i>	$\sqsubseteq$	<i>area</i>
<i>mountain</i>	$\sqsubseteq$	<i>area</i>
<i>germany</i>	$\sqsubseteq$	<i>country</i>
<i>czech_republic</i>	$\sqsubseteq$	<i>country</i>
<i>local_river</i>	$\equiv$	$(river \sqcap \neg(\exists PO.country))$
<i>non_local_river</i>	$\equiv$	$(river \sqcap (\exists PO.country))$
<i>river_flow_into_a_lake</i>	$\equiv$	$(river \sqcap (\exists PO.lake))$
<i>german_river</i>	$\equiv$	$(river \sqcap (\exists PP.germany) \sqcap (\forall PO.\neg country))$
<i>german_city</i>	$\equiv$	$(city \sqcap (\forall PP.(country \rightarrow germany)))$
<i>city_at_river</i>	$\equiv$	$(city \sqcap (\exists PO.river))$
<i>elbe</i>	$\sqsubseteq$	$(river \sqcap (\exists PO.czech_republic) \sqcap (\exists PO.germany))$
<i>alster_lake</i>	$\sqsubseteq$	<i>lake</i>
<i>alster</i>	$\sqsubseteq$	$(river \sqcap (\exists PP.germany) \sqcap (\exists PO.alster_lake) \sqcap (\forall PO.\neg country) \sqcap (\forall PP.(country \rightarrow germany)))$
<i>hamburg</i>	$\sqsubseteq$	$(city \sqcap (\exists PO.alster))$

Die Taxonomie dieser TBox ist in Abb. 3.6 dargestellt (und wurde durch das MIDeLoRA-System, s. Kap. 6, berechnet). Interessanterweise wurde hier *hamburg* als Unterkonzept von *german_city* und auch von *city_at_a_river* erkannt. Letzterer Schluss ist nicht überraschend, da in der TBox explizit

$$hamburg \sqsubseteq (city \sqcap (\exists PO.alster))$$

angegeben wurde, und *alster* ein Unterkonzept von *river* ist. Die Inferenz, dass *hamburg* eine *german_city* ist, ist aber bemerkenswert. Die Inferenz kann wie folgt nachvollzogen werden:

- Es gilt *hamburg* $\sqsubseteq$ *german_city* gdw. *hamburg* $\sqcap \neg german\_city$ unerfüllbar ist. Also muss auch

$$(city \sqcap (\exists PO.alster)) \sqcap \neg(city \sqcap (\forall PP.(country \rightarrow germany)))$$

unerfüllbar sein. Dieses Konzept ist äquivalent zu

$$(city \sqcap (\exists PO.alster)) \sqcap (\neg city \sqcup (\exists PP.(country \sqcap \neg germany))).$$

Da *city* $\dots \sqcap \neg city$ ein offensichtlicher Widerspruch ist, bleibt zu zeigen dass

$$(city \sqcap (\exists PO.alster)) \sqcap (\exists PP.(country \sqcap \neg germany))$$

unerfüllbar ist.

- Durch Einsetzen der „Definition“ von *alster* erhalten wir

$$(city \sqcap (\exists PO. (river \sqcap (\exists PP.germany) \sqcap (\exists PO.alster\_lake) \sqcap (\forall PO.\neg country) \sqcap (\forall PP.(country \rightarrow germany)))) \sqcap (\exists PP.(country \sqcap \neg germany))).$$

3.1 Leitdomäne 1: Geographische Informationssysteme

Wir nennen dieses Konzept X . Nehmen wir an, es gibt eine Instanz namens $city\_ind$, die dieses Konzept erfüllt, also $city\_ind \in X^I$.

Da offensichtlich auch $city\_ind \in (\exists PP.country \sqcap \neg germany)^I$ gilt (es handelt sich um ein Konjunkt von X), muss noch ein anderes Individuum – nennen wir es $germany\_ind$ – existieren, sodass $(city\_ind, germany\_ind) \in PP^I$ gilt.

Zudem gilt noch $city\_ind \in (\exists PO.alster)^I$. Daher muss es noch ein weiteres Individuum geben, nennen wir es $alster\_ind$, sodass $(city\_ind, alster\_ind) \in PO^I$ gilt.

Weil nun $(germany\_ind, city\_ind) \in PPI^I$ und $(city\_ind, alster\_ind) \in PO^I$ und $PPI \circ PO \sqsubseteq PPI \sqcup PO$ gilt, muss entweder $(germany\_ind, alster\_ind) \in PPI^I$ oder $(germany\_ind, alster\_ind) \in PO^I$ gelten.

Im ersten Falle gilt auch $(alster\_ind, germany\_ind) \in PP^I$, und es gibt einen direkten Widerspruch durch $alster\_ind \in (\forall PP.(country \rightarrow germany))^I$.

Im zweiten Fall erhalten wir ebenfalls einen Widerspruch, da $alster\_ind \in (\forall PO.\neg country)^I$ gilt (PO ist symmetrisch).

Somit haben wir die Unerfüllbarkeit von $hamburg \sqcap \neg germany\_city$ gezeigt. Daher gilt $hamburg \sqsubseteq germany\_city$.

Zwischenzeitlich wurde in [Lutz u. Wolter 2004; 2006] gezeigt, dass $\mathcal{ALCT}_{RCC5}$ und $\mathcal{ALCT}_{RCC8}$ unentscheidbar sind. Alternativ könnten syntaktische Einschränkungen zur Sicherstellung der Entscheidbarkeit wie in [Haarslev u. a. 1999] eingeführt werden [Wessel 2003a], doch wäre die Beispiel-KB dann nicht mehr zulässig.

Aufgrund dieser Ergebnisse scheint es fraglich, dass ein GIS mit räumlicher Semantik (für nichttriviale RCC-Relationssysteme) in der intensionalen Komponente implementiert werden kann. Wir müssen daher *Unvollständigkeit* bzgl. der räumlichen Aspekte in Kauf nehmen, oder aber auf deren Modellierung in der TBox verzichten.

In [Wessel 2003b, Wessel u. Möller 2006] hatten wir beobachtet, dass ontologiebasierte Anfragebeantwortung auf *speziellen* $\mathcal{ALCT}_{RCC}$ -ABoxen mittels eines $\mathcal{ALCIN}$ -DL-Systems (also beispielsweise RACERPRO) realisiert werden kann, nämlich auf solchen, für die wir die *Annahme der Domänenabgeschlossenheit* (CDA) machen.

Zum „Abschließen der Domäne“ werden unqualifizierten Anzahlrestriktionen (N) benötigt. Eine solche ABox nennen wir eine *RCC-abgeschlossene ABox*. Zunächst wird anhand der Geometrie der Karte das RCC-Netz berechnet, wobei das verwendete RCC-System beliebig ist. Das Netz wird mittels Rollen-Assertionen in der ABox $\mathcal{A}$ repräsentiert. Die Individuen in der ABox entsprechen dann den Knoten im RCC-Netz. Dann *schließen* wir diese ABox bzgl. der

RCC-Rollen ab, indem wir

1. die UNA für die Individuen sicherstellen, und
2. für jedes Individuum i und für jede RCC-Relation R die Anzahl n der R -Nachfolger im RCC-Netz bestimmen: $n = |\{j \mid (i, j) : R \in \mathcal{A}\}|$.

Wir fügen dann die Konzept-Assertion $i : (\leq_n R)$ hinzu. Die Assertion $i : (\geq_n R)$ ist eine Konsequenz aus den n vorhandenen R -Rollen-Assertionen und der UNA. Somit gilt in allen Modellen: $i^{\mathcal{I}} \in (=_n R)^{\mathcal{I}}$.

Die resultierende ABox nennen wir $\mathcal{A}_c$ (c für „closed“). Es gilt für diese ABox nun:

Proposition 1 Sei $\mathcal{A}_c$ eine abgeschlossene ($\mathcal{ALCCIN}_{RCC}$ -erfüllbare) RCC-ABox, und $\mathcal{T}$ eine assoziierte $\mathcal{ALCCIR}_{RCC}$ -TBox.

Dann gilt: $\text{concept\_instances}_{\mathcal{ALCCIN}_{RCC}}(C, \mathcal{A}_c) = \text{concept\_instances}_{\mathcal{ALCCIN}}(C, \mathcal{A}_c)$ (für ein $\mathcal{ALCCIR}_{RCC}$ -Anfragekonzept C). ■

M.a.W., für ontologiebasierte Anfragebeantwortung auf abgeschlossenen $\mathcal{ALCCIR}_{RCC}$ -ABoxen kann ein Standard-DL-System verwendet werden.

Beweis 1 (Skizze) Da $i \in \text{concept\_instances}_{\mathcal{ALCCIN}_{RCC}}(C, \mathcal{A}_c)$ gdw. $\mathcal{A}_c \cup \{i : \neg C\}$ $\mathcal{ALCCIN}_{RCC}$ -unerfüllbar ist, ist zu zeigen, dass $\mathcal{A}_c \cup \{i : \neg C\}$ $\mathcal{ALCCIN}_{RCC}$ -unerfüllbar ist gdw. $\mathcal{A}_c \cup \{i : \neg C\}$ $\mathcal{ALCCIN}$ -unerfüllbar ist. Offensichtlich können wir uns hier auf $\exists R.D$ - und $\forall R.D$ -Konzepte beschränken, mit $R \in RCC$.

Zunächst gilt trivialerweise stets: Ist $\mathcal{A}_c \cup \{i : E\}$ $\mathcal{ALCCIR}_{RCC}$ -erfüllbar, so auch $\mathcal{ALCCIN}$ -erfüllbar (dies kann durch Übersetzung der KBs in prädikatenlogische Formelmengen gesehen werden, denn die $\mathcal{ALCCIR}_{RCC}$ -Übersetzung ist eine Obermenge der $\mathcal{ALCCIN}$ -Übersetzung). Daher folgt aus der $\mathcal{ALCCIN}$ -Unerfüllbarkeit von $\mathcal{A}_c \cup \{i : E\}$ auch die $\mathcal{ALCCIN}_{RCC}$ -Unerfüllbarkeit.

Angenommen, $\mathcal{A}_c \cup \{i : \exists R.D\}$ ist $\mathcal{ALCCIN}$ -erfüllbar, aber $\mathcal{ALCCIN}_{RCC}$ -unerfüllbar. Es gibt daher ein $i^{\mathcal{I}} \in (\exists R.D)^{\mathcal{I}}$ mit $(i^{\mathcal{I}}, a) \in R^{\mathcal{I}}$ und $a \in D^{\mathcal{I}}$. Aufgrund der Abgeschlossenheit muss es ein $j \in \text{inds}(\mathcal{A}_c)$ mit $j^{\mathcal{I}} = a$ geben. Da $\mathcal{A}_c$ $\mathcal{ALCCIN}_{RCC}$ -erfüllbar ist (die originale ABox ist konsistent), $\mathcal{A}_c \cup \{i : \exists R.D\}$ aber $\mathcal{ALCCIN}_{RCC}$ -unerfüllbar ist, muss $i^{\mathcal{I}} \in (\forall R.\neg D)^{\mathcal{I}}$ in der (bzw. jeder) erweiterten Interpretation gelten (andernfalls wäre die erweiterte ABox erfüllbar). Aufgrund der JEPD-Eigenschaft der RCC-Relationen und da die RCC-Rollen-Assertionen anhand einer konkreten Karte berechnet wurden (somit liegt vollständiges Wissen bzgl. der vorliegenden RCC-Relation vor), muss jedoch $(i, j) : R \in \mathcal{A}_c$ gelten (da ja $(i^{\mathcal{I}}, a) \in R^{\mathcal{I}}$ mit $j \in \text{inds}(\mathcal{A}_c)$ und $j^{\mathcal{I}} = a$), und daher $i^{\mathcal{I}} \notin (\forall R.\neg D)^{\mathcal{I}}$ und somit $i^{\mathcal{I}} \in (\exists R.D)^{\mathcal{I}}$ gelten, im direkten Widerspruch zur Annahme, dass $\mathcal{A}_c \cup \{i : \exists R.D\}$ $\mathcal{ALCCIN}_{RCC}$ -unerfüllbar ist. Daher folgt aus der $\mathcal{ALCCIN}$ -Erfüllbarkeit von $\mathcal{A}_c \cup \{i : E\}$ auch die $\mathcal{ALCCIN}_{RCC}$ -Erfüllbarkeit, bzw. per Kontraposition aus der $\mathcal{ALCCIN}_{RCC}$ -Unerfüllbarkeit auch die $\mathcal{ALCCIN}$ -Unerfüllbarkeit.

3.1 Leitdomäne 1: Geographische Informationssysteme

Die Argumentation für $\mathcal{A}_c \cup \{i : \forall R.D\}$ verläuft analog. □

Offensichtlich werden die unqualifizierten Anzahlrestriktionen lediglich zur Erreichung der Domänenabgeschlossenheit benötigt. In einem Tableau-Beweiser könnte man den gleichen Effekt durch eine spezialisierte $\exists$ -Regel erreichen: Anstatt zur Befriedigung von $i : \exists R.D \in \mathcal{A}$ die ABox um $\{j : D, (i, j) : R\}$ für ein neues Individuum j zu erweitern, würde diese modifizierte $\exists$ -Regel versuchen, bereits existierende Individuen aus $\text{inds}(\mathcal{A}_c)$ für diesen Zweck zu verwenden. Die Regel wird dann nichtdeterministisch, da mehrere Kandidatenindividuen existieren können. Dies erfordert jedoch einen *Spezial-Beweiser*. Wir kommen auf diese Idee zurück.

Wir haben also gezeigt, dass mit gewissen Einschränkungen durchaus Standard-DL-Systeme zu Realisierung $\mathcal{ALCI}_{\mathcal{RCC}}$ -artiger ontologiebasierter Anfragebeantwortung verwendet werden können. Allerdings gibt es prinzipielle Probleme mit der Vollständigkeit in der intensionalen Komponente – Konzepte werden u.U. nicht als inkonsistent erkannt, die TBox kann inkonsistent oder inkohärent werden, ohne dass dieses bemerkt wird, etc.

Aufgrund der Probleme mit der Vollständigkeit der intensionalen Komponente können $\mathcal{ALCI}_{\mathcal{RCC}}$ -Anfragekonzepte lediglich unvollständig auf Erfüllbarkeit und Subsumption getestet werden. Wir halten dies zunächst als (weiteres) Defizit fest. Zudem hatten wir als weiteres Defizit bereits festgehalten, dass Anfragekonzepte keine tupelwertigen Ergebnisse liefern können. Prinzipiell ist die Unvollständigkeit dieser Dienste für die Anfragekomponente nicht so gravierend wie die Unvollständigkeit in der TBox.

3.1.6. Nutzungspotentiale und -defizite von DLs und DL-Systemen

Stehen wir vor der Aufgabe, die Komponenten des ontologiebasierten GIS mit Hilfe von DL-Technologie zu realisieren, so ergeben sich eine Fülle von Nutzungsmöglichkeiten, aber auch Defizite. Die Diskussion motiviert die in späteren Kapiteln ausgeführten Arbeiten:

- Für die intensionale Komponente können wir eine TBox verwenden.

Wir hatten die Probleme bzgl. der intensionalen Modellierung der räumlichen Aspekte exemplarisch am Beispiel der $\mathcal{ALCI}_{\mathcal{RCC}}$ -Familie erläutert. Die adäquate Berücksichtigung räumlicher Eigenschaften erfordert Spezial-DLs mit räumlicher Semantik. Entscheidbarkeit für diese ist oftmals schwer zu erreichen, und die Axiomatisierung räumlicher Eigenschaften ist prinzipiell oftmals nur unvollständig möglich. Entweder muss Unvollständigkeit in Kauf genommen werden, oder aber Unentscheidbarkeit. Hingegen ist die Nützlichkeit entscheidbarer räumlicher Logiken in die-

ser Domäne stark eingeschränkt (z.B. von $\mathcal{ALCI}_{RCC3}$). Zudem erfordern Spezial-DLs auch Spezial-Beweiser. Wir werden diesen Weg dennoch in Kap. 6 (mittels MIDELORA) weiterverfolgen.

Standard-DLs und somit Standard-DL-Systeme wie RACERPRO können in diesen Szenarien dennoch genutzt werden. Dies mag wissenschaftlich zunächst nicht interessant erscheinen. Wir verfolgen diesen Weg jedoch nicht zuletzt deswegen weiter, weil wir die praktische Verwendbarkeit von DLs und DL-Systemen als Basis-Technologie für derartige Anwendungen evaluieren möchten. Verwenden wir Standard-DLs und -Beweiser, müssen wir daher Unvollständigkeit bzgl. der räumlichen Aspekte in der intensionalen Komponente in Kauf nehmen.

Rein skalare räumliche Aspekte (wie Länge und Fläche) können prinzipiell in der konkreten Domäne modelliert werden, z.B. *grosser_park* mittels

$$grosser\_park \triangleq park \sqcap \exists(size). \geq_{100000}$$

Prinzipiell können mit Hilfe einer *räumlichen* konkreten Domäne sogar *räumliche Relationen* durch sog. *definierte Rollen* bereitgestellt werden. Diese würden automatisch die erforderlichen Eigenschaften durch die Reichhaltigkeit der konkreten Domäne haben. Leider ist die $\mathcal{ALC}(\mathcal{D})$ -Erweiterung namens $\mathcal{ALCRP}(\mathcal{D})$ [Haarslev u. a. 1999; 1998], die dies ermöglicht, aufgrund einer Entscheidbarkeit garantierenden Syntaxeinschränkung in diesem Szenario nur eingeschränkt nutzbar, wie bereits erläutert. Zudem existieren keine performanten Implementationen von $\mathcal{ALCRP}(\mathcal{D})$ -Beweisern.

Die Expressivität von Standard-DLs ist zwar nicht ausreichend zur Modellierung echter räumlich-thematischer Konzepte. Wie bereits anhand der Objektschlüssel „Friedhof für Nichtchristen“ und „Friedhof“ diskutiert, würde aber bereits ein einfacher thematischer Thesaurus einen großen Mehrwert in diesem Szenario darstellen. Wir remodellieren daher den Objektschlüsselkatalog als eine TBox. Wir achten dabei auf entsprechendes „Common Sense“-Vokabular und stellen eine reichhaltigere thematische Modellierung bereit.

Prinzipiell können wir auch RCC-Rollen in den Konzeptdefinitionen in der TBox zulassen und somit Konzepte wie

$$park\_mit\_teich \triangleq park \sqcap \exists PPI.teich$$

aufnehmen. Die Eigenschaften der Relationen werden dann aber ausschließlich zur ontologiebasierten Anfragebeantwortung berücksichtigt, wie in Kap. 3.1.5 dargestellt.

- Für die *extensionale Komponente* können wir zumindest zur Repräsentation *ausgewählter Aspekte* der Karten- bzw. Geo-Objekte der DISK eine ABox verwenden. Wir werden also pro Karten-Objekt in der DISK ein ABox-

3.1 Leitdomäne 1: Geographische Informationssysteme

Individuum anlegen. Aufgrund der Anzahl der Karten-Objekte muss daher mit tausenden von Individuen gerechnet werden.

Die thematischen Aspekte und weitere zugehörige Sachdaten eines Karten-Objektes lassen sich sicherlich gut in der ABox repräsentieren. Es sollten jedoch nur die Aspekte repräsentiert werden, die relevant sind (entweder für Anfragen oder bzgl. der Ontologie).

Für die räumlichen Aspekte gilt prinzipiell die gleiche Problematik wie bereits für die intensionale Komponente diskutiert. Metrische und skalare räumliche Eigenschaften (Länge, Fläche, ...) lassen sich als „Daten“ in der konkreten Domäne repräsentieren.

Qualitative räumliche Relationen können als Rollen-Assertionen anhand der Geometrie der Karte vorweg berechnet und in der ABox (z.B. in Form eines RCC-Netzes) repräsentiert werden. Da hier ein K_n -Graph gebildet werden muss, werden die ABoxen sehr groß.

Prinzipiell kann die Geometrie der Karten-Objekte (und somit eine „reichhaltige Repräsentation“) in einer ausdrucksstarken (räumlichen) konkreten Domäne realisiert werden (ein konvexes Polygon kann durch ein lineares Ungleichungssystem dargestellt werden). Eine solche Repräsentation erscheint jedoch nicht adäquat, solange die Geometrie nicht für nicht-triviale Inferenzprozesse genutzt wird. Stattdessen ist es hier wichtiger, die Reichhaltigkeit der Geometrie zu nutzen, um abgeleitete räumliche Aspekte (Eigenschaften und Relationen) mittels Inspektionsmethoden zu berechnen. Zudem kann effiziente Beantwortung räumlicher Anfragen nur durch spezialisierte räumliche Indexstrukturen ermöglicht werden. Beides ist in Standard-DLs bzw. -Systemen nicht ohne weiteres realisierbar. Letztlich wird die Geometrie natürlich auch für Präsentationszwecke benötigt.

Ein Ausweg wäre, eine *hybriden Repräsentation* zu verwenden. Ein Karten- bzw. Geo-Objekt wird dann als ein *Paar* von Individuen in zwei verschiedenen Repräsentationsschichten angelegt – ein Individuum in einer ABox, welches mit einer Instanz eines räumlichen Datentyps (Spatial Data Types, SDTs) zur Repräsentation der Geometrie assoziiert ist (über eine injektive Abbildung). Die räumliche Schicht kann sowohl Indexstrukturen als auch Inspektionsmethoden bereitstellen. Diese können die ABox bei Bedarf um entsprechende Assertionen erweitern. Offensichtlich ist solch eine hybride Repräsentation nur dann sinnvoll, wenn die beiden Repräsentationen über eine entsprechende *hybride* Anfragesprache *kombiniert* werden können. Einfache Instanzenretrieval-Anfragen scheiden dann aus.

Eine weitere Idee ist, statt einer hybriden Repräsentation eine *räumliche*

ABox zu verwenden. In einer räumlichen ABox fallen die beiden Ebenen der soeben diskutierten hybriden Repräsentation zusammen. Ein ABox-Individuum ist dann auch eine Instanz eines SDTs. Somit ist eine räumliche ABox eine reichhaltige Repräsentation. Auch die Realisierung dieser Idee erfordert einen Spezial-Beweiser. Wir kommen in Kap. 6.1.5 hierauf zurück und erläutern die Vorteile dieser Repräsentation später.

- Die *Anfrage-Komponente* muss eine entsprechend ausdrucksstarke Anfragesprache zur Verfügung stellen, sodass die Referenz-Anfragen aus Kap. 3.1.3.3 formuliert werden können. Es ist bereits deutlich geworden, dass einige der in den Referenz-Anfragen referenzierten räumlichen Aspekte Spezial-DLs erfordern. Unter Verwendung einer hybriden Repräsentation in der extensionalen Komponente und einer entsprechenden Anfragesprache könnten diese jedoch per Model Checking behandelt werden. Wir halten folgende Defizite von einfachen Instanzenretrieval-Anfragen fest:
 - es sind keine Variablen verfügbar und somit keine tupelwertigen Ergebnisse,
 - unvollständige Inferenz für Anfragen (Anfrage-Erfüllbarkeit und Anfrage-Subsumption) aufgrund von Unentscheidbarkeit oder Unvollständigkeit für „räumliche“ Anfragekonzepte,
 - keine Nutzung räumlicher Indexstrukturen zur räumlichen Anfragebeantwortung (erfordert Spezial-Beweiser),
 - keine Möglichkeit, hybride bzw. „reichhaltige“ Repräsentationen zu befragen,
 - Anfragen müssen durch teure Widerlegungsbeweise beantwortet werden, auch wenn bereits Model Checking ausreichend wäre.

Aus diesen Gründen entscheiden wir uns dafür, eine *hybride konjunktive Anfragesprache* bereitzustellen. Die Basis-Idee einer solchen Anfragesprache ist, dass spezielle Typen von Anfrage-Atomen zur Adressierung spezieller (räumlicher, thematischer, ...) Aspekte bereitgestellt werden, die dann durch entspr. spezialisierte Algorithmen ausgewertet werden können.

Wir werden später auch darstellen, wie Erfüllbarkeit und Subsumption für spezielle hybride räumlich-thematische Anfragen entschieden werden kann.

Die Anfragesprache soll zudem so flexibel sein, dass sie zur Anfragebeantwortung auf verschiedenen Repräsentationen in der extensionalen Komponente (Repräsentation als ABox, hybride Repräsentation, Repräsentation als *räumliche* ABox, ...) genutzt werden kann. Zur Anfragebeantwortung müssen daher u.U. verschiedene *Quellen* (bzw. verschie-

3.1 Leitdomäne 1: Geographische Informationssysteme

dene Repräsentationen) genutzt und die erhaltene Teilergebnisse müssen von der Anfragebeantwortungsmaschine zu einem Gesamtergebnis kombiniert werden.

3.1.7. Fazit

Die Diskussion machte deutlich, dass die Realisierung eines ontologiebasierten GIS zur ontologiebasierten Anfragebeantwortung ein nichttriviales Unterfangen ist.

Zur Evaluierung der Praxistauglichkeit existierender DL-Systeme in diesem Szenario und um die verschiedenen diskutierten Repräsentationsmöglichkeiten in der extensionalen Komponente zu ermöglichen, benötigen wir ein *flexibles Datenmodell*, welches eine Generalisierung einer ABox darstellt. Dieses Datenmodell wird in Kap. 4 entworfen und in Kap. 5 softwaretechnisch realisiert werden.

Das Datenmodell ist ohne (hybride konjunktive) *flexible Anfragesprache* nutzlos. Die Flexibilität ist erforderlich, da die Anfragesprache für verschiedene Repräsentationen in der extensionalen Komponente nutzbar sein soll. Wir entwerfen den Rahmen für die Anfragesprache(n) parallel zum Datenmodell. Da das Datenmodell eine ABox generalisiert und ABoxen in der extensionalen Komponente genutzt werden, muss diese Anfragesprache auch eine Generalisierung einer konjunktiven ABox-Anfragesprache sein.

Schließlich werden wir in Kap. 6 die Idee ausbauen, dass einige Nutzungsmöglichkeiten von DL-Technologie in dieser Domäne nur durch Spezial-Beweiser ermöglicht werden. Wir hatten mehrere Punkte diskutiert, die Spezial-Beweiser erforderlich machen:

1. Spezielle DLs erfordern Spezial-Beweiser ($\mathcal{ALCI}_{RCC}$ -Familie).
2. Spezielle Inferenzprobleme oder spezielle domänenspezifische Optimierungsverfahren erfordern Spezial-Beweiser, da diese nicht von Standard-DL-Systemen implementiert werden. Wir hatten in Kap. 3.1.5 erläutert, wie eine abgeschlossene RCC-ABox durch zusätzliche Konzept-Assertionen mit unqualifizierten Anzahlrestriktionen generiert werden kann. Auf diese zusätzlichen Assertionen kann prinzipiell verzichtet werden, wenn stattdessen eine *Spezial-Implementierung* von `concept_instances` bereitgestellt wird. Dies erfordert aber einen ABox-Beweiser, der eine spezialisierte $\exists$ -Regel verwendet.
3. Zur Realisierung der Idee der *räumlichen ABox* benötigen wir Beweiser, die die *Reichhaltigkeit* der ABox ausnutzen können und die sich somit der Räumlichkeit der Repräsentation „bewusst“ sind.

Diese Punkte motivieren die Schaffung des flexiblen MIDELORA-Systems in Kap. 6.

3.2. Leitdomäne 2: Information Retrieval im SEMANTIC WEB

Das SEMANTIC WEB [Berners-Lee u. a. 2001] ist heutzutage ein sehr aktives und massiv gefördertes Forschungsfeld. DLs bilden ein wichtiges Standbein des SEMANTIC WEB, denn die offizielle W3C SEMANTIC WEB-Sprache OWL (Web Ontology Language) [Bechhofer u. a. 2006b] wurde stark durch Beschreibungslogiken inspiriert.

Für DL-Systeme wie RACER (dem Vorgänger von RACERPRO) ergaben sich daher mit dem Aufkommen von OWL ganz neue Einsatzmöglichkeiten – sie konnten durch Abbildung von OWL-Sprachkonstrukten auf DL-Konstrukte als Inferenzmaschinen für das SEMANTIC WEB fungieren [Haarslev u. Möller 2003]; vgl. auch [Baader u. a. 2002; 2003b].

Wir wollen in Leitdomäne 2 also ISe, die IR im SEMANTIC WEB ermöglichen, betrachten.

Wir beschreiben zunächst die Vision und die „Architektur“ des SEMANTIC WEBS und stellen die SEMANTIC WEB-Sprachen RDF und OWL vor. Ein hervorragendes einführendes Buch ist [Antoniou u. v. Harmelen 2004]; detaillierter auf RDF und RDFS geht [Powers 2003] ein. Ontologiebasierte Anfragebeantwortung bzw. IR im SEMANTIC WEB ist ohne Anfragesprachen natürlich nicht möglich. Wir beschreiben daher existierende „Standard“-Anfragesprachen für das SEMANTIC WEB. Schließlich identifizieren wir Nutzungsmöglichkeiten und Defizite von DLs und DL-Systemen zur Realisierung eines SEMANTIC WEB IR-Dienstes. Die hier identifizierten Potentiale und Defizite motivieren die weiteren Arbeitspunkte in dieser Arbeit. Wir schließen das Unterkapitel mit einem Fazit ab.

3.2.1. Die Vision des SEMANTIC WEB

Das WWW, wie es heutzutage existiert, hat den Nachteil, dass die meisten „Informationen“, die es enthält, nicht für Maschinen zugänglich sind. Die auf Webseiten (mittels HTML) präsentierten Informationen bzw. Inhalte sind lediglich von Menschen interpretierbar; die *maschinelle* Verarbeitung derartiger Seiten beschränkt sich auf die technische Bereitstellung und Präsentation. Die treffende Allegorie „Syntactic Web“ verdeutlicht diesen Sachverhalt.

Heutige Internet-Suchmaschinen (z.B. „Google“) verwenden ausschließlich klassische statistische IR-Methoden und Volltextsuche in HTML-Seiten, Indizierung anhand von Schlüsselwörtern, Relevanzsortierung anhand des Vernetzungsgrads, etc. Es ist wohlbekannt, dass diese Art von IR Defizite hat. So erge-

3.2 Leitdomäne 2: Information Retrieval im SEMANTIC WEB

ben sich u.a. Probleme mit Homonymen und Synonymen (Thesauri können einige dieser Probleme mildern). Es ist eine alltägliche Erfahrung, dass bei der Internet-Recherche mit Suchmaschinen oftmals viele irrelevante Treffer geliefert werden (schlechte *Genauigkeit*), oder aber die wirklich relevanten Ressourcen nicht geliefert werden (schlechte *Vollständigkeit*), s. Def. 49, S. 130.

Eine *explizite inhaltliche Beschreibung von Web-Ressourcen mittels einer maschinenverarbeitbaren Annotationssprache mit formaler Semantik* findet heutzutage (in aller Breite) noch nicht statt. Dies ist jedoch Voraussetzung, um *inhaltsbasierte bzw. semantikbasierte Suchanfragen* oder z.B. Dienste anhand von (funktionalen) Dienstbeschreibungen aufzufinden („SEMANTIC WEB Service Matchmaking“). Die benötigten Annotationen werden oftmals auch *Meta-Daten* genannt, also „Daten über Daten“. Im Rahmen des SEMANTIC WEBS werden für diese Meta-Daten ausschließlich logikbasierte Annotationen verwendet, sodass diese eine wohldefinierte formale Semantik aufweisen, die *maschinelle Inferenz* ermöglicht. So wird mit Hilfe der Annotationen die durch die Ressource repräsentierte Information explizit und maschinen zugänglich und -verarbeitbar [Berners-Lee u. a. 2001].

Zur Identifikation bzw. Referenzierung der zu annotierenden Web-Ressourcen werden sog. *universelle Ressourcen-Identifizierer-Referenzierer* (*Universal Resource Identifier References*, *URIRefs*) verwendet [Berners-Lee u. a. 2004]. Ein URIRef ist ein mit einem Postfix versehener *universeller Ressourcen-Identifizierer* (*Uniform Resource Identifier*, *URI*) [Berners-Lee u. a. 2004, Kapitel 4]. URIs umfassen die bekannten URLs (Uniform Resource Locators) und URNs (Uniform Resource Names). Der Postfix wird *Fragment-Identifizierer* (*Fragment Identifier*) genannt.

Ein URIRef wird als `URI#Ref` notiert. Ein URI kann auf eindeutige Weise Dinge im Internet referenzieren (ein HTML-Dokument, einen Webservice, ...), aber auch Dinge, die nicht im Internet verfügbar sind, z.B. Personen oder Bücher in einer Bibliothek, oder auch abstrakte Konzepte, die physisch nicht existieren, wie z.B. das Konzept eines Autors. Der Fragment-Identifizierer wird verwendet, um ein Element („Fragment“) *innerhalb* der durch den URI-Präfix identifizierten Ressource zu referenzieren. Ein Beispiel für einen URIRef ist `http://www.sts.tu-harburg.de/people#mi.wessel`. Da die Annotationen für die Ressourcen maschineninterpretier- und -auswertbar sein müssen, findet XML als syntaktische Basis für die Sätze der Annotationssprachen Anwendung, da es sich bei XML um ein universelles Standardformat handelt. Ressourcen werden annotiert, indem in der Annotation einfach eine URIRef auf diese Ressource verwendet wird – URIRefs „halten somit das SEMANTIC WEB zusammen“.

Die prominentesten Annotationssprachen sind RDF [Manola u. Miller 2004] und OWL [Bechhofer u. a. 2006a]. RDF steht für *Resource Description Fra-*

network. Der Name macht schon deutlich, dass hiermit Ressourcen *beschrieben* (annotiert) werden sollen. RDF-*Schema* (RDFS) [Brickley u. Guha 2000] kann als eine sehr einfache Ontologie-Sprache bezeichnet werden und dient dazu, das *Vokabular (Schema)* für RDF zu definieren. OWL ist wesentlich ausdrucksstärker. Sowohl RDF / RDFS als auch OWL haben formale Semantiken [Hayes 2004], [Patel-Schneider u. a. 2004].

Natürlich müssen diese Sprachen nicht ausschließlich zur Annotation von Web-Ressourcen genutzt werden; RDF kann auch einfach als graphbasiertes „Datenmodell“ charakterisiert und verwendet werden, ebenso wie OWL-Wissensbasen auch ohne jeglichen Internet-Bezug Anwendung finden können.

Um überhaupt Retrieval von Ressourcen im WWW (und somit „Recall“) zu ermöglichen, muss in den Anfragen *standardisiertes Basis-Vokabular* verwendet werden, welches einer *gemeinsam verwendeten, geteilten Konzeptualisierung* entspringt. Für diesen Zweck werden z.B. von der ISO (und ähnlichen Organisationen) genormte domänenspezifische Thesauri und Begriffssysteme gepflegt, beispielsweise für das Bibliothekswesen. Der Aspekt der gemeinsam verwendeten, geteilten Konzeptualisierung wurde bereits bei der strengeren Definition des Begriffs „Ontologie“ nach Gruber in Kap. 2.1.2.5 erläutert.

Im SEMANTIC WEB werden die benötigten gemeinsam verwendeten Konzeptualisierungen nun öffentlich zugänglich bereitgestellt, denn auch Ontologien sind letztlich Ressourcen im Web und können somit mittels URIs referenziert und erhalten werden. Bei Bedarf können diese Ontologien auch von anderen Nutzern (dezentral und verteilt) erweitert, „personalisiert“ und auch wieder publiziert werden. Eine solche öffentlich zugängliche Ontologie kann z.B. eine „Upper Ontology“ für eine bestimmte Domäne sein, die Vokabular für das Bibliothekswesen bereitstellt. Tatsächlich existieren genormte RDF(S)-Vokabularien wie der *Dublin Core (DC)* für genau diesen Zweck [Kokkelink u. Schwänzl] (auch wenn wir nicht so weit gehen wollen, den DC als eine „echte“ Ontologie zu bezeichnen, dafür ist er zu einfach).

3.2.2. Der SEMANTIC WEB-Stack

Für das SEMANTIC WEB wurde vom *World Wide Web Consortium (W3C)* eine geschichtete Architektur definiert, die als SEMANTIC WEB-Stack bekannt ist, s. Abb. 3.7. Als unterste Ebenen werden bekannte Standards (wie XML, XML-Namespaces und -Schemata, Unicode, URIs etc.) verwendet. Wir setzen diese als bekannt voraus.

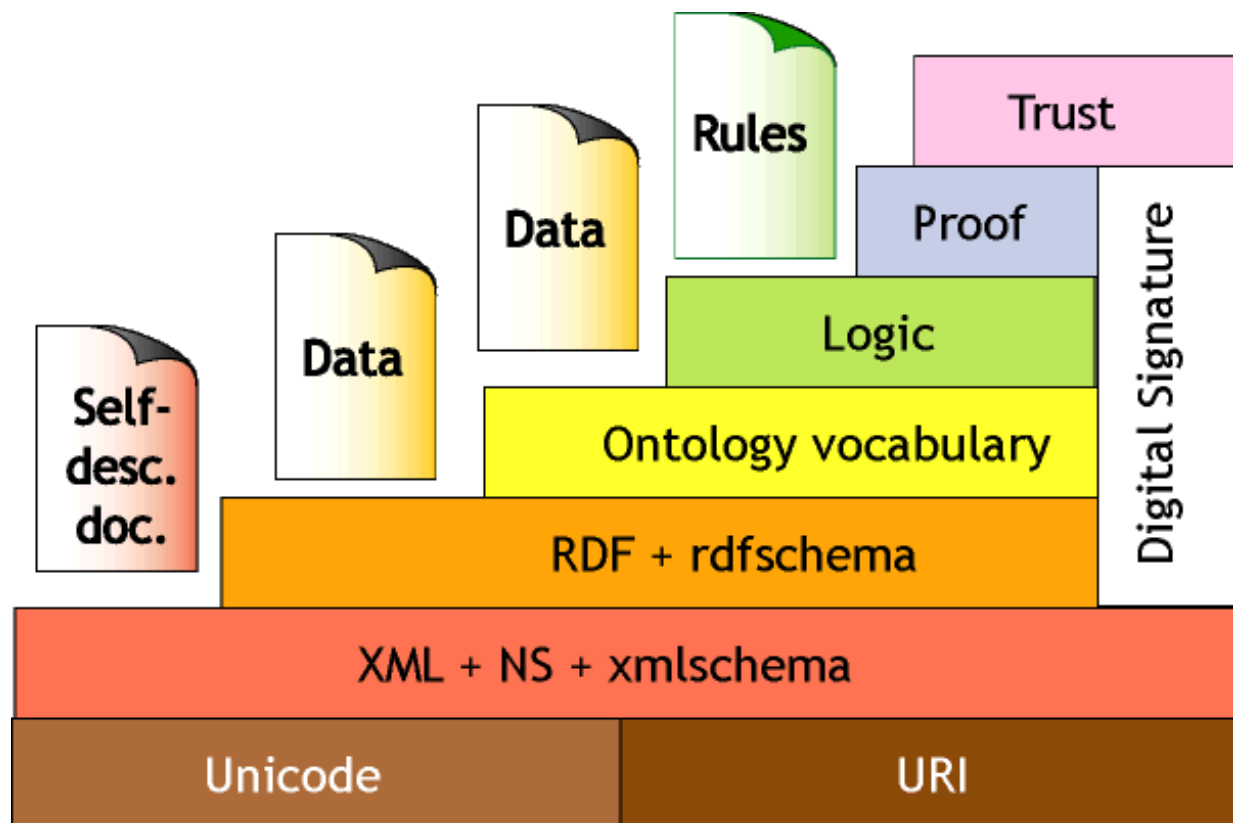


Abbildung 3.7.: Der SEMANTIC WEB-Stack (auch: SEMANTIC WEB Layer Cake) nach Berners-Lee. Quelle: <http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html>

3.2.2.1. RDF und RDF-Schema

Die nächsthöhere Ebene im SEMANTIC WEB-Stack ist durch das bereits erwähnte RDF sowie RDF-Schema gegeben. RDF stellt ein einfaches Graph-Datenmodell bereit. Ähnliche Datenmodelle wurden in der Vergangenheit oft zur Repräsentation semistrukturierter Daten vorgeschlagen. Im Gegensatz zu XML können mittels RDF in angemessener Weise beliebige annotierte Graphen (nicht nur Bäume) dargestellt werden. Zudem ist die Datenrepräsentation in RDF in gewisser Weise kanonischer möglich als in XML (in XML kann ein- und dasselbe Datum auf sehr verschiedene Arten syntaktisch dargestellt werden, z.B. als Attributwert oder als Kind-Knoten im XML-Baum, was das spätere Retrieval des Datums erschwert).

Semistrukturierte Datenmodelle werden oftmals als flexibel und *selbstbeschreibend* charakterisiert (s. auch „Self-desc. doc.“ in Abb. 3.7). Selbstbeschreibend bedeutet, dass die Daten i.d.R. auch ohne Kenntnis des Schemas (durch einen Menschen) interpretiert werden können, und dass ein Schema oft optional ist.

Offensichtlich müssen die Daten selbstbeschreibend sein, wenn das Schema nicht gegeben ist. Kommen wir auf das in Kap. 2.1.2.2 verwendete Beispieldatum 37 zurück, welches entweder eine Hausnummer oder das Alter einer Person charakterisiert, so würde selbstbeschreibend in XML bedeuten, dass diese Mehrdeutigkeit z.B. wie folgt aufgelöst würde: `<alter>37</alter>` bzw. `<haus nr=37 .../>`. Selbst wenn die Daten einem Schema gehorchen, können i.d.R. Abweichungen der Daten von dem Schema toleriert werden, indem das Schema entsprechend flexibel gestaltet wird, z.B., indem Disjunktionen in den Schema-Definitionen oder sehr generelle „Any“-Typen zur Typisierung verwendet werden. Solche „Any“-Typen können mit einem $\top$ -Konzept verglichen werden. Diese Flexibilität ist insbesondere auch für komplexe Datenstrukturen möglich. Im Gegensatz hierzu ist ein Schema im relationalen Datenmodell wesentlich *starrer* und somit *strukturierter* und strenger. Die einzelnen Tupel einer Relation sind auch nicht selbstbeschreibend. Sowohl XML als auch RDF werden als semistrukturiert charakterisiert, das relationale Datenmodell hingegen als (stark) strukturiert.

RDF wird üblicherweise in XML dargestellt. Es gibt jedoch auch alternative, menschenlesbarere Serialisierungen, z.B. „Notation 3 (N3)“.

Ein RDF-Dokument ist eine Menge von RDF-Aussagen (RDF-Statements). Eine RDF-Aussage beschreibt eine beschriftete Kante zwischen zwei Web-Ressourcen oder einer Web-Ressource und einem Literal (XML-Schema-Datum) im RDF-Graphen. Web-Ressourcen werden durch URIRefs identifiziert. Eine Kante zwischen zwei Ressourcen wird auch als *Subjekt-Prädikat-Objekt Tripel (SPO-Tripel)* bezeichnet. Nicht nur Subjekt und Objekt sind durch URIRefs identifiziert – auch das RDF-Prädikat (kurz: Prädikat) ist ein URIRef. Das Prädikat stellt letztlich die Kanten-Beschriftung bereit, bzw. nennt die zwischen Subjekt und Objekt geltende Beziehung. Das Prädikat ist daher ein Element eines RDF-Vokabulars. Ein Beispiel für ein SPO-Tripel in XML-Syntax ist

```
<rdf:Description rdf:about="&STSPeople;mi.wessel">
  <STSPeople:hasBoss>&STSPeople;r.f.moeller</STSPeople:hasBoss>
</rdf:Description>
```

Hier wird eine Beziehung `&STSPeople:hasBoss` zwischen den Ressourcen `&STSPeople;mi.wessel` und `STSPeople;r.f.moeller` etabliert. Bei `&STS`; handelt es sich um eine sog. *XML-Entität (XML Entity)*. Dieses muss zuvor als einfaches „Makro“ für den URI `http://www.sts.tu-harburg.de/people#` definiert werden (hier nicht ausgeführt).

Oftmals sollen *primitive Daten (primitive Literale bzw. einfache Daten-Werte)* statt URIRefs im Bild eines Prädikates² verwendet werden, z.B. für ein Prädikat

²Da Prädikate binäre Relationen darstellen, sind die Begriffe Urbild- und Bild anwendbar.

3.2 Leitdomäne 2: Information Retrieval im SEMANTIC WEB

STSPeople:age. Für diesen Zweck können statt eines URIRefs Instanzen von *XML-Schema*-Datentypen verwendet werden. Für komplex strukturierte Daten bzw. für Standard-Datenstrukturen stellt RDF auch einige vordefinierte *Container-Datenstrukturen* wie Listen, Mengen und Multimengen bereit. Da hier RDF-Einzelaussagen zu komplexeren Aussagen verknüpft werden müssen (ähnlich den Cons-Zellen zur Repräsentation einfach verketteter Listen), sieht RDF das Konzept der *unbeschrifteten Knoten (Blank Nodes)* vor. Diese dienen als interne *anonyme* Hilfsressourcen zur Verknüpfung von Teilaussagen/Tripeln und sind nicht durch URIRefs referenzierbar (sie sind anonym).

Ein komplexeres RDF-Beispiel in XML sieht wie folgt aus; das Beispiel ist in Abb. 3.8 als Graph dargestellt. Das Beispiel sollte aufgrund der vorangegangenen Erläuterungen selbsterklärend sein:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE rdf:RDF [
    <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
    <!ENTITY STSPeople 'http://www.sts.tu-harburg.de/people#'>
    <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
]>
<rdf:RDF xmlns:rdf="&rdf;"
        xmlns:STSPeople="&STSPeople;"
        xmlns:rdfs="&rdfs;">

  <rdf:Description rdf:about="&STSPeople;mi.wessel">
    <rdf:type rdf:resource="&STSPeople;assistant"/>
    <rdfs:label>michael</rdfs:label>
    <STSPeople:age>35</STSPeople:age>
    <STSPeople:name>Michael Wessel</STSPeople:name>
    <STSPeople:hasBoss>&STSPeople;r.f.moeller</STSPeople:hasBoss>
  </rdf:Description>
</rdf:RDF>
```

RDF definiert auch eine abkürzende, äquivalente Darstellung (ihrerseits *kinderelemente* können alternativ als Attribut-Werte notiert werden); anders als in XML wird hierdurch der selbe Graph aufgespannt:

```
<STSPeople:assistant rdf:about="&STSPeople;mi.wessel"
  STSPeople:age="35"
  STSPeople:name="Michael Wessel"
  STSPeople:hasBoss="&STSPeople;r.f.moeller"
  rdfs:label="michael"/>
```

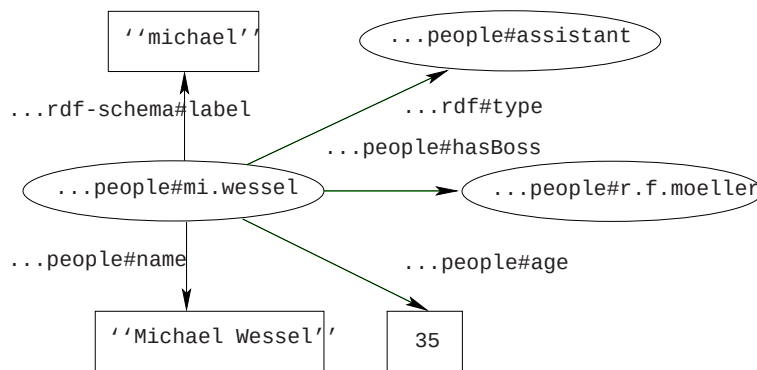



Abbildung 3.8.: Das RDF-Beispiel als Graph

Da auch die Prädikate durch URIRefs identifiziert werden, können Mengen von Prädikaten eindeutig identifiziert, wiederverwendet und auch erweitert werden. Eine Menge von Prädikaten wird auch *Vokabular (Vocabulary)* genannt. Der bereits erwähnte *Dublin Core (DC)* stellt beispielsweise die Prädikate *creator*, *title*, *author*, wie sie u.a. im Bibliothekswesen benötigt werden, bereit. In unserem Beispiel-RDF-Dokument haben wir die domänenspezifischen Prädikate *STSPeople:age*, *STSPeople:name*, *STSPeople:hasBoss* verwendet. Diese müssen keine weitere „Definition“ haben; es handelt sich einfach um URIRefs.

RDF selbst stellt lediglich eine minimale Menge von generischen, vordefinierten Prädikaten zur Verfügung. Hierzu gehört das Prädikate *type* (auch im obigen Beispiel verwendet), und eine Reihe von Prädikaten, die zur *Reifikation von Aussagen* benötigt werden: *subject*, *predicate*, *object*. Eine Aussage (SPO-Tripel) kann selbst wieder durch eine URIRef referenziert werden. Da diese Aussagen referenzierenden URIRefs nun selbst wieder in Aussagen verwendet werden können, können somit *Aussagen über Aussagen* gemacht werden.³

Die Anwendungen müssen sich ihr benötigtes (domänenspezifisches) Vokabular also selbst definieren, oder auf „standardisiertes“ Vokabular zurückgreifen (s. Dublin Core). Für viele Anwendungen werden zudem standardisierte RDF-Prädikate zur grundlegenden konzeptuellen Modellierung benötigt, beispielsweise ein normiertes Prädikat für die Spezialisierungsbeziehung. Die *RDF Vocabulary Description Language 1.0: RDF Schema* [Brickley u. Guha 2000], oder kurz *RDF Schema (RDFS)*, definiert eine Menge von RDF-Prädikaten mit *festgelegter Semantik*, u.a. *subClassOf* und *type*, sodass Deduktion ermöglicht wird. Zudem können Bild- und Urbild-Restriktionen (wie für DL-Rollen) mittels der Prädikate *domain* und *range* formuliert werden. RDFS kennt auch das Analogon zu *Rollenhierarchien* mittels *subPropertyOf*.

³Dies führt potentiell zu Problemen, wie sie in der Mengenlehre bzw. mathematischen Logik bekannt sind, z.B. dem Russel-Paradoxon.

3.2 Leitdomäne 2: Information Retrieval im SEMANTIC WEB

RDF plus RDFS wird als RDF(S) bezeichnet. Die RDF(S)-Definitionen der im obigen Beispiel verwendeten Prädikate könnten unter Nutzung dieser Möglichkeiten wie folgt aussehen:

```
<rdfs:Class rdf:about="&STSPeople;assistant"
    rdfs:label="assistant">
    <rdfs:subClassOf rdf:resource="&STSPeople;employee"/>
</rdfs:Class>

<rdf:Property rdf:about="&STSPeople;age"
    rdfs:label="age">
    <rdfs:domain rdf:resource="&STSPeople;person"/>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

<rdf:Property rdf:about="&STSPeople;hasBoss"
    rdfs:label="hasBoss">
    <rdfs:domain rdf:resource="&STSPeople;person"/>
    <rdfs:range rdf:resource="&STSPeople;professor"/>
</rdf:Property>

<rdf:Property rdf:about="&STSPeople;name"
    rdfs:label="name">
    <rdfs:domain rdf:resource="&STSPeople;person"/>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
```

Hervorzuheben ist, dass für RDF(S) eine *modelltheoretische Semantik* definiert wurde [Hayes 2004]. Diese wurde aufgrund ihrer Kompliziertheit jedoch kritisiert [Pan u. Horrocks 2003b;a]. Die Semantik erfordert zunächst, dass Ressourcen auf Elemente einer Interpretationsdomäne abgebildet werden (ähnlich den ABox-Individuen). Prädikate werden (wie in DLs) auf binäre Relationen über der Interpretationsdomäne abgebildet. Da Prädikate jedoch selbst auch wieder Ressourcen sind, könnten die Extensionen von Prädikaten sich selbst als Elemente enthalten; dies wird durch einen „Trick“ verhindert, indem zwei verschiedene Interpretationsabbildungen verwendet werden. Alternative (vereinfachte) Semantiken für RDF(S) werden in [Pan u. Horrocks 2003b;a, de Bruijn u. a. 2005b;a] untersucht. Durch entsprechende Syntaxeinschränkungen kann diese Problematik ebenfalls vermieden werden; dieser Weg wird für OWL DL gewählt.

RDF(S)-Inferenz wird durch die festgelegte Semantik der Prädikate `subClassOf`, `type`, `domain`, `range` ermöglicht. So ergibt sich beispielsweise aus den Tripeln `<a, subClassOf, b>` und `<b, subClassOf, c>` das implizierte Tripel `<a, subClassOf, c>`. Diese *implizierten Tripel* werden auch *axiomatische Tripel*

genannt. Es existieren (für eine vereinfachte, eingeschränkte RDF-Semantik) regelbasierte Kalküle zur Ableitung solcher Tripel, welche sich effizient implementieren lassen [Antoniou u. v. Harmelen 2004, Kap. 3.7], [Fikes u. McGuinness 2001].

Sogenannte *Triplestores* sind RDF- oder RDF(S)-Datenbanken. Diese implementieren i.d.R. nicht nur einen Graph-Datenspeicher, sondern auch eine RDF-Anfragesprache, z.B. RQL oder SPARQL [Prud'hommeaux u. Seaborne 2006], s. Kap. 3.2.3.

RDF(S) ist als eine einfache Ontologiesprache erkennbar. Gleichwohl die Ausdruckskraft von RDF(S) prinzipiell zur Definition von Thesauri etc. ausreichend ist, fehlen dennoch spezielle *standardisierte* Prädikate, z.B. zur Auszeichnung von Synonymen und Homonymen etc. Aktuell werden diese zur Abbildung von einfachen Wissensorganisationsstrukturen benötigten Prädikate normiert. Diese RDF(S)-Anwendung wird SKOS (*Simple Knowledge Organisation System*) genannt [Miles u. Brickley 2005]. Verglichen mit DLs fehlen RDF(S) viele weitere praktisch relevante Ontologie-Modellierungskonstrukte (es wird noch nicht einmal Aussagenlogik zur Verfügung gestellt; Disjunktion und Negation fehlen). OWL geht hier einen wesentlichen Schritt weiter.

3.2.2.2. Die „Web Ontology Language“ – OWL

OWL ist in die über der RDF-Ebene liegende „Ontology vocabulary“-Ebene im SEMANTIC WEB-Stack in Abb. 3.7 einzuordnen. OWL baut auf RDF(S) auf, stellt nun aber eine sehr viel ausdrucksstärkere Konzept-Beschreibungssprache bereit. OWL existiert in drei unterschiedlichen Ausprägungen: OWL *Lite*, OWL *DL*, und OWL *Full* [McGuinness u. v. Harmelen 2004]. OWL wurde letztlich durch DLs inspiriert; speziell die OWL DL-Ausprägung macht dies deutlich. Während OWL Full unentscheidbar ist, ist zur Abbildung von OWL DL auf DLs $SHOIN(\mathcal{D}_{n^{--}})$ ausreichend (für OWL Lite wird $SHIf(\mathcal{D}_{n^{--}})$ benötigt) [Horrocks u. Patel-Schneider 2003], [Horrocks u. a. 2003].

In einer OWL-Wissensbasis (einem OWL-Dokument) findet man einen intensionalen und einen extensionalen Anteil. Im intensionalen Teil werden *Klassen (Classes)* und *Eigenschaften (Properties)* definiert. *Properties* repräsentieren binäre Relationen und sind mit benutzerdefinierten RDF(S)-Prädikaten und DL-Rollen vergleichbar. Die Individuen (Individuals) und Beziehungen werden in der extensionalen Komponente als RDF-Graph dargestellt (tatsächlich wird die RDF-Repräsentation auch für den intensionalen Anteil eines OWL-Dokuments genutzt). Die Individuen referenzieren per `<rdf:type rdf:about=... />` einfach die entsprechende Klasse, ähnlich wie bereits für RDF(S) gezeigt.

Während es in RDF(S) nur möglich ist, TBox-Axiome der Art $C \sqsubseteq D$ für *atomare*

3.2 Leitdomäne 2: Information Retrieval im SEMANTIC WEB

C, D anzugeben, können in OWL nun auch Konzeptdefinitionen der Art $C \doteq D$ vorkommen, und D muss nicht mehr atomar sein.

Zur Illustration sei eine einfache OWL-Wissensbasis angegeben, die die Modellierung des oben diskutierten RDF(S)-Beispiels vertieft. Wir wollen für das Beispiel folgende TBox-Axiome in OWL abbilden:

$$\begin{aligned} \text{BossOf} &\doteq \text{hasBoss}^{-1} \\ \text{BossOf} &\sqsubseteq \text{responsibleFor} \\ \text{employee} &\sqsubseteq \exists \text{hasBoss}.\text{employee} \sqcap \exists (\text{hasOfficeNumber}).\top_{\mathbf{N}} \\ \text{assistant} &\sqsubseteq \text{employee} \sqcap \neg \text{professor} \sqcap \forall \text{hasBoss}.\text{professor} \\ \text{professor} &\sqsubseteq \text{employee} \sqcap \neg \text{assistant} \sqcap \forall \text{hasBoss}.\text{professor} \end{aligned}$$

Die Rollen haben zudem besondere Eigenschaften: *hasBoss* soll funktional sein ($\text{hasBoss} \in \mathcal{N}_{\mathcal{F}}$), *responsibleFor* transitiv ($\text{responsibleFor} \in \mathcal{N}_{\mathcal{R}^+}$), und *hasOfficeNumber* ein $\mathbf{N}$ -wertiges Attribut ($\text{hasOfficeNumber} \in \mathcal{N}_{\mathcal{A}}$); dies entspricht dem XML Schema-Datentyp `XMLSchema#integer`:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns="http://www.sts.tu-hamburg.de/people.owl#"
  xml:base="http://www.sts.tu-hamburg.de/people.owl">
  <owl:Ontology rdf:about="" />

  <owl:Class rdf:ID="professor">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:FunctionalProperty rdf:ID="hasBoss"/>
        </owl:onProperty>
        <owl:allValuesFrom rdf:resource="#professor"/>
      </owl:Restriction>
    </rdfs:subClassOf>

    <rdfs:subClassOf>
      <owl:Class rdf:ID="employee"/>
    </rdfs:subClassOf>
  </owl:Class>
</rdf:RDF>
```

```

    <owl:disjointWith>
      <owl:Class rdf:ID="assistant"/>
    </owl:disjointWith>
  </owl:Class>

  <owl:Class rdf:about="#employee">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="hasOfficeNumber"/>
        </owl:onProperty>
        <owl:someValuesFrom
          rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
        </owl:Restriction>
      </rdfs:subClassOf>

      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:someValuesFrom rdf:resource="#employee"/>
          <owl:onProperty>
            <owl:FunctionalProperty rdf:about="#hasBoss"/>
          </owl:onProperty>
        </owl:Restriction>
      </rdfs:subClassOf>
    </rdfs:subClassOf>
    <rdfs:subClassOf
      rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  </owl:Class>

  <owl:Class rdf:about="#assistant">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:FunctionalProperty rdf:about="#hasBoss"/>
        </owl:onProperty>
        <owl:allValuesFrom rdf:resource="#professor"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <owl:disjointWith rdf:resource="#professor"/>
    <rdfs:subClassOf rdf:resource="#employee"/>
  </owl:Class>

  <owl:ObjectProperty rdf:ID="responsibleFor">
    <rdfs:type
      rdf:resource="http://www.w3.org/2002/07/owl#TransitiveProperty"/>
  </owl:ObjectProperty>

```

```
<owl:ObjectProperty rdf:ID="BossOf">
  <rdfs:subPropertyOf rdf:resource="#responsibleFor"/>
  <rdf:type
    rdf:resource=
      "http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/>
  <owl:inverseOf>
    <owl:FunctionalProperty rdf:about="#hasBoss"/>
  </owl:inverseOf>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:about="#hasOfficeNumber">
  <rdfs:domain rdf:resource="#employee"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdf:type rdf:resource=
    "http://www.w3.org/2002/07/owl#FunctionalProperty"/>
</owl:DatatypeProperty>

<owl:FunctionalProperty rdf:about="#hasBoss">
  <rdf:type rdf:resource=
    "http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <owl:inverseOf rdf:resource="#BossOf"/>
</owl:FunctionalProperty>

</rdf:RDF>
```

OWL kennt prinzipiell drei Arten von Eigenschaften (Properties):

- *Objekt-Eigenschaften (Object Properties)* entsprechen den klassischen DL-Rollen und werden verwendet, um Assoziationen zu modellieren (z.B. hasBoss).
- *Datentyp-Eigenschaften (Datatype Properties)* werden zur Repräsentation datenwertiger Attribute verwendet (z.B. hasOfficeNumber; hier werden Instanzen der Klasse employee mit XMLSchema#integer-Datenwerten assoziiert).
- *Annotations-Eigenschaften (Annotation Properties)* werden zur (optionalen) Annotation von Ressourcen verwendet. Es findet keine Inferenz mit diesen statt. Annotiert werden können u.a. Ressourcen, welche Klassen, Eigenschaften, und Individuen repräsentieren. Das Objekt des Annotations-Tripels muss entweder ein Datenwert, eine URIRef, oder ein OWL-Individuum sein. Häufig werden die Prädikate des Dublin Core als Annotations-Eigenschaften verwendet.

Eigenschaften können als transitiv abgeschlossen, symmetrisch, zueinander invers, funktional und invers-funktional deklariert werden. Von RDFS wur-

den bereits die Möglichkeiten zur Bild- und Urbild-Restriktion sowie zur Bildung von „Rollenhierarchien“ geerbt. Properties können auch als zueinander äquivalent definiert werden (`equivalentProperty`). Diese Deklarationen können gleichermaßen für Object wie auch Datatype Properties gemacht werden, nicht jedoch für Annotation Properties.

In der extensionalen Komponente wird in OWL standardmäßig keine UNA verwendet. Individuen können mittels `sameAs` als identisch, mittels `differentFrom` als verschieden (zu jedem Element in einer Gruppe von Individuen), und mittels `AllDifferent` als in einer Gruppe paarweise zueinander verschieden deklariert werden.

Die drei Ausprägungen von OWL stellen die folgenden Modellierungskonstrukte zur Verfügung:

- OWL DL hat die gleiche Expressivität wie $\mathcal{SHOIN}(\mathcal{D}_{n^{--}})$ mit eingeschränkten konkreten Domänen für primitive XML-Schemadatentypen ($\mathcal{D}_{n^{--}}$). Der prädikatsbasierte Existenz-Operator ist hier noch weiter eingeschränkt als für DLs mit $\mathcal{D}^-$ im Namen, denn $\mathcal{D}_{n^{--}}$ erlaubt letztlich nur unäre Prädikate.

Die booleschen DL-Konstruktoren haben folgende OWL-Entsprechungen:
 $\sqcap$ = `intersectionOf`, $\sqcup$ = `unionOf`, $\neg$ = `complementOf`.

Restriktionen auf Rollen werden mittels `Restriction` und $\exists R.C$ = `someValuesFrom` bzw. $\forall R.C$ = `allValuesFrom` umgesetzt (s. obiges Beispiel). Prinzipiell können diese Restriktionen auch für Datentyp-Eigenschaften gemacht werden; die Qualifikationen sind dann entspr. XML-Schema Basis-Datentypen.

Unqualifizierte Anzahlrestriktionen ($\mathcal{N}$) werden mittels `minCardinality`, `maxCardinality` und `cardinality` angeboten. Wie auch in allen entscheidbaren DLs, die $\mathcal{SHN}$ erweitern, werden keine Anzahlrestriktionen auf Properties erlaubt, die transitiv sind oder transitive Unterproperties haben (wir hatten diese Rollen in Kap. 2.2.4 „einfach“ genannt).

Nominale ($\mathcal{O}$) werden durch sog. *enumerierte Klassen*, also durch Enumeration von Individuen mittels `owl:oneOf` ermöglicht. Beispielsweise kann die Klasse `AmpelFarbe` wie folgt definiert werden:

```
<owl:Class rdf:ID="AmpelFarbe">
  <rdfs:subClassOf
    rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#Rot"/>
    <owl:Thing rdf:about="#Gelb"/>
    <owl:Thing rdf:about="#Gruen"/>
  </owl:oneOf>
</owl:Class>
```

IDRefs können zunächst sowohl Klassen (Ressourcen vom Typ `owl:Class`), Properties (Ressourcen vom Typ `owl:ObjectProperty`, `owl:DatatypeProperty`), als auch Individuen (Ressourcen vom Typ `owl:Thing`) referenzieren, doch führt dies zu semantischen Problemen; offensichtlich sollten in Argumentpositionen der Klassen-Konstruktoren nicht Ressourcen beliebigen Typs referenziert werden. In OWL Lite und OWL Full wird daher eine strikte *Sortentrennung* eingehalten – dies bedeutet, dass die *Interpretation (Extensionen)* der Prädikate `owl:Class`, `owl:ObjectProperty` und `owl:Thing` als zueinander disjunkt angenommen werden. Dies impliziert u.a., dass Klassen nicht als Individuen, und Individuen nicht als Klassen interpretiert werden können. Wir hatten erläutert, dass die RDF(S)-Semantik dies zulässt (sind a und b vom Typ `owl:Class`, so deklariert ein Tripel der Art $\langle a, \text{type}, b \rangle$ die Klasse a als *Instanz* der Klasse b). Dies führt dazu, dass nicht jedes RDF(S)-Dokument ein gültiges OWL Lite oder OWL DL-Dokument ist (die Schichtung ist hier also „gebrochen“). Erst in OWL Full wird (zum Preis der Unentscheidbarkeit) diese Sortentrennung aufgehoben. Da $SHOIN(\mathcal{D}_n^{--})$ NEXPTIME-vollständig ist, gilt dies auch für OWL DL.

- OWL *Lite* „entfernt“ syntaktisch die Operatoren Negation und Disjunktion sowie die Nominale aus OWL DL. Implizit sind Negation und Disjunktion jedoch weiterhin formulierbar. Zudem werden im $\mathcal{N}$ -Operator lediglich die Anzahlrestriktionen 0 und 1 zugelassen. Die DL $SHIf(\mathcal{D}_n^{--})$ ist daher hinreichend ausdrucksstark, um OWL Lite abzubilden. OWL Lite ist daher EXPTIME-vollständig.
- In OWL *Full* wird die Sortentrennung für Individuen und Klassen aufgehoben. OWL Full ist unentscheidbar und daher nicht sonderlich relevant für praktische Anwendungen.

In den Konstruktoren können jetzt auch RDFS-Prädikate (die ja eine festgelegte Semantik haben), wie z.B. `rdf:type`, verwendet werden (dies ist in OWL DL nicht möglich). So ist es in OWL Full z.B. möglich, eine Anzahlrestriktion bzgl. des `rdfs:subclassOf`-Prädikats anzugeben, womit die *Anzahl der Unterklassen einer Klasse* eingeschränkt würde.

Das berühmte Russelsche Paradoxon sieht in OWL wie folgt aus [Herman 2004], vgl. auch [Horrocks u. a. 2003]:

```
<owl:Class rdf:ID="A">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource=".../22-rdf-syntax-ns#type"/>
      <owl:allValuesFrom rdf:about="#B"/>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
```

```
<owl:class rdf:ID="B">
  <owl:complementOf rdf:parseType="Collection">
    <owl:Class rdf:about="#A"/>
  </owl:complementOf>
</owl:class>

<owl:Thing rdf:ID="C">
  <rdf:type rdf:resource="#A"/>
</owl:Thing>
```

Hier wird eine Klasse A definiert, die genau die Instanzen enthält, die in B enthalten sind. Die Klasse B ist jedoch als Komplement von A definiert. Offensichtlich muss nun jedes Individuum in A oder im Komplement von A enthalten sein. Da in OWL Full Klassen nun ebenfalls Individuen sind, stellt sich die Frage: Ist beispielsweise die Klasse A in der Klasse A enthalten? Wenn ja, so muss A in B und daher im Komplement von A liegen, ein Widerspruch. Ist A hingegen nicht in A enthalten, so ist es lt. Definition von B in B. Lt. Definition von A ist A dann aber wieder in A enthalten, ein Widerspruch. Das Elementproblem für die Klasse A und somit OWL Full ist daher unentscheidbar. Die gleiche Problematik ergibt sich natürlich auch für eine „einfache“ Instanz, hier die A-Instanz namens C.

Man beachte, dass dieser Widerspruchsbeweis dem Unentscheidbarkeitsbeweis für das Halteproblem einer Turingmaschine sehr ähnlich ist (denn hier hält die Maschine genau dann, wenn der vermeintliche Algorithmus zur Entscheidung des Halteproblems „hält nicht“ sagt, und andersrum).

3.2.3. Anfragesprachen für das SEMANTIC WEB

Anfragesprachen für das SEMANTIC WEB existieren u.a. in Form von RDF(S)-Anfragesprachen. Hiervon gibt es mehrere dutzend; namentlich sollen nur RQL [Karvounarakis u. a. 2002] und SPARQL [Prud'hommeaux u. Seaborne 2006] als prominente Vertreter erwähnt werden. Unter der Sichtweise, dass RDF(S) eine sehr einfache Ontologiesprache ist, können wir zu recht sagen, dass diese Anfragesprachen *ontologiebasierte Anfragebeantwortung* bzw. *ontologiebasiertes IR* im SEMANTIC WEB ermöglichen. Diese gilt umso mehr für OWL-QL, s.u.

Die RDF(S)-Anfragesprachen verwenden üblicherweise eine SQL-inspirierte SELECT WHERE-Syntax; idealerweise sollten auch *inferierte* bzw. *axiomatische Tripel* für die Anfragebeantwortung herangezogen bzw. berücksichtigt werden. Diese Anfragesprachen bieten üblicherweise auch Prädikate, mit denen die XML-Datenwerte in dem zu befragenden RDFS-Dokument auf bestimmte Eigenschaften überprüft werden können (z.B. Prädikate, mit den Datumsangaben

verglichen werden können, Enthaltensein einer Zeichenkette in einer anderen Zeichenkette überprüft werden kann, etc.). In SPARQL werden diese Prädikate mittels FILTER notiert und als *Werte Einschränkungen (Value Constraints)* bezeichnet.

Bezogen auf obiges RDF-Dokument kann die folgende SPARQL-Anfrage verwendet werden, um Name und Alter aller Personen, die jünger als 40 sind, zu ermitteln:

```
PREFIX STSPeople: <http://www.sts.tu-harburg.de/people>
SELECT ?name ?age
WHERE { ?x STSPeople:age ?age .
        FILTER (?age < 40) .
        ?x STSPeople:name ?name . }
```

Für das wesentlich ausdrucksstärkere OWL existieren ebenfalls Vorschläge für Anfragesprachen. Die Standardisierungsprozesse für OWL-Anfragesprachen sind jedoch noch nicht abgeschlossen. Die OWL-Anfragesprache OWL-QL [Fikes u. a. 2003; 2005] verwendet keine SELECT WHERE-Syntax, sondern es wird die Sichtweise eingenommen, dass eine OWL-QL-Anfrage ein beliebiges OWL-Dokument ist, in der einige der URIRRefs als Variablen gedeutet werden. Die Semantik einer Anfrage wird durch *logische Folgerbarkeit* definiert – werden die durch ein Anfrageergebnis gegebenen Variablen-Bindungen gegen die Variablen in der OWL-QL-Anfrage substituiert, so muss das so erhaltene OWL-Dokument eine *logische Folgerung* aus dem befragten OWL-Dokument sein. Eine Folgerungsbeziehung („|=“) wurde für OWL DL in [Patel-Schneider u. a. 2004, Abschnitt 3] definiert. Diese kann auf Erfüllbarkeit von DL-Wissensbasen reduziert werden [Horrocks u. Patel-Schneider 2003].

OWL-QL unterscheidet drei Arten von Variablen – sog. *must bind*- (auch: *distinguished*), *don't bind*- und *may bind*-Variablen. Dabei müssen *must bind*-Variablen gegen erfüllende, explizit bekannte Ressourcen gebunden werden. Für die *don't bind*-Variablen wird dies nicht gefordert; stattdessen ist es ausreichend, wenn die Existenz einer entspr. Ressource anhand der OWL-Wissensbasis logisch impliziert ist bzw. bewiesen werden kann. Die Ressource muss also nicht explizit bekannt sein. Die *may bind*-Variablen haben die gleiche Semantik wie die *don't bind*-Variablen, allerdings *kann* für sie eine Bindung im Anfrageergebnis zurückgegeben werden, falls eine erfüllende Ressource doch explizit vorliegt.

Zur Illustration der drei Variablenarten betrachten wir die (prädikatenlogische) Formelmenge $\{\exists z. R(i, z)\}$. Offensichtlich existiert in allen Modellen ein R^I -Nachfolger von i^I , der jedoch keinen Namen hat. Auf die Anfrage $R(x, y)$ kann daher nur für x eine Bindung im Anfrageergebnis zurückgegeben werden, nämlich $x = i$. Daher könnte x als *must bind*-Variable deklariert werden; y

hingegen kann nur als *don't bind*-Variable deklariert werden und muss somit als *existentiell quantifiziert* betrachtet werden. Offensichtlich gilt $\{\exists z.R(i, z)\} \models \exists y.R(x, y)_{x \leftarrow i}$, also $\{\exists z.R(i, z)\} \models \exists y.R(i, y)$. Daher ist $x = i$ eine Antwort.

Eine weitere Dimension ist durch die *dynamischen Aspekte* dieser Anfragesprachen gegeben. OWL-QL definiert ein Protokoll zur Client-Server-Kommunikation. Der Client stellt eine OWL-QL-Anfrage an den Server. Dieser liefert dann eine Anfrage-ID zur Identifikation der Anfrage an den Client zurück. Dem Client wird durch Verwendung der ID das *inkrementelle Nachfordern von Teilmengen der Antwortmenge* ermöglicht. Zudem kann der Client die maximal erwartete Anzahl an „Antworttupeln“ spezifizieren.

3.2.4. Nutzungspotentiale und Defizite von DLs und DL-Systemen

Wie kann IR im SEMANTIC WEB unter Verwendung von DL-Technologie nun ermöglicht werden? Im Folgenden betrachten wir die Anforderungen, die sich für einen SEMANTIC WEB *IR-Dienst* ergeben, von nun an *SWIR-Dienst* genannt.

Natürlich wird für einen solchen SWIR-Dienst eine Anfragesprache und (mindestens) ein inferenzfähiger Triplestore benötigt. Da die Ausdrucksmächtigkeit von RDF(S) für viele Anwendungen nicht ausreichend ist, entscheiden wir uns dafür, ein OWL-fähiges DL-System als Basis-OWL-Inferenzmaschine für diesen Dienst zu verwenden: RACERPRO.

RACERPRO unterstützt OWL Lite vollständig, und OWL DL mit Approximationen für Nominale. RACERPRO repräsentiert ein OWL-Dokument als DL-Wissensbasis. Der intensionale Anteil des OWL-Dokuments wird in der TBox repräsentiert. Der extensionale Anteil des OWL-Dokuments bzw. der RDF-Graph wird in der ABox repräsentiert; die ABox ist also in gewisser Weise der Triplestore. Eine ausdrucksstarke ABox-Anfragesprache mit Variablen kann somit als Basis für eine SEMANTIC WEB-Anfragesprache dienen. Somit ist RACERPRO ein idealer Kandidat, um als Basis für eine SWIR-Maschine Anwendung zu finden. Bis zum Abschluss dieser Arbeit stellte RACERPRO jedoch lediglich API-Funktionen für die *ABox-Basis-Retrieval-Probleme* (s. Kap. 2.2.2.5) bereit. Eine ausdrucksstarke ABox-Anfragesprache mit Variablen gab es noch nicht.

Warum implementiert der Autor nicht einfach OWL-QL zur Realisierung dieses SWIR-Dienstes (und verwendet RACERPRO als OWL-Inferenzmaschine)? Zunächst einmal wäre es aus rein pragmatischen Gründen unsinnig, zwar eine OWL-QL-Implementation für RACERPRO zu erstellen, jedoch keine ausdrucksstarke ABox-Anfragesprache anzubieten, wenn diese doch fehlte und von Nutzern bereits angefragt wurde. Im Rahmen einer *geschichteten Architektur* erscheint es daher sinnvoll, zunächst die ausdrucksstarke ABox-Anfragesprache

zu entwerfen und zu realisieren, die dann als Basis-Maschine für eine OWL-QL-Implementation verwendet werden kann. Tatsächlich wurde zwischenzeitlich eine (prototypische) OWL-QL-Implementation erstellt, die NRQL als Basis-Maschine verwendet [Galinski u. a. 2005].

Zudem kann die Anfragesprache OWL-QL an mehreren Stellen kritisiert werden:

1. Die Unterscheidung von drei Arten von Variablen in OWL-QL verkompliziert die Sprache für Endnutzer. Der praktische Wert von *don't bind*-Variablen ist aus der Perspektive des Autors fraglich. Wir hatten erläutert, dass diese als existentiell quantifiziert behandelt werden. Existentiell quantifizierte Variablen sind jedoch bereits in einigen DL- bzw. OWL-Konstrukten implizit enthalten: Für unser schon oben verwendetes Beispiel gilt $\{\exists x.R(i, x)\} \models \exists y.R(i, y)$ und somit $\{\exists x.R(i, x)\} \models i : \exists R.\top$ in DL-Syntax.

Betrachten wir die Variablen in einer Anfrage als Knoten in einem Graphen, und die binären Prädikate als gerichtete Kanten in diesem Graphen, so können wir diese existentiell quantifizierten Variablen genau dann gegen einen Schachtelung von $\exists R\dots$ -Konzepten ersetzen, wenn dieser Graph ein Baum ist (die Situation verkompliziert sich, wenn auch *must bind*-Variablen betrachtet werden). Tatsächlich wird diese sog. *Technik des „Aufrollens“ (Unrolling)*, baumförmige konjunktive Anfragen in einfache Anfragekonzepte zu übersetzen, die dann mittels einer einfachen Instanzenretrieval-Anfrage bearbeitet werden können, in der Literatur untersucht. Dies funktioniert unmittelbar jedoch nur, wenn lediglich die Wurzel-Variable dieser Anfrage eine *must bind*-Variable ist, und alle anderen Variablen *don't bind*-Variablen sind, s. auch [Horrocks u. Tessaris 2002]. In der Praxis werden viele Anfragen baumförmig sein; für diese Anfragen kann daher auf *don't bind*-Variablen verzichtet werden.

2. Aufgrund der Tatsache, dass OWL-QL-Anfragen letztlich OWL-Dokumente sind (mit einigen Zusatzdeklarationen, welche URIRefs als Variablen welchen Typs zu interpretieren sind), ergeben sich einige Einschränkungen:

- a) OWL erlaubt keine benutzerdefinierten Datentypen. So kann das Konzept des Erwachsenen für ein $\mathbb{R}$ -wertiges Attribut *has_age* in $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (und somit in RACERPRO) mittels

$$adult \doteq person \sqcap \exists(has\_age). \geq_{18}$$

formuliert werden. OWL kennt jedoch keine (benutzerdefinierten) Prädikate (bzw. Einschränkungen von XML Schema-Datentypen) wie $\geq_{18}$. SPARQL erlaubt mittels FILTER (*?age* >= 18) die Ermittlung aller Erwachsenen; in OWL-QL ist dies nicht möglich.⁴

⁴Eine OWL-Erweiterungen, die dies ermöglicht, ist z.B. OWL-E. Für $\mathbb{N}$ -wertiges *has_age*

- b) OWL bzw. $\mathcal{SHOIN}(\mathcal{D}_{n^{--}})$ bietet lediglich einen eingeschränkten Prädikats-Existenz-Operator ($\mathcal{D}_{n^{--}}$), der nur unäre Prädikate erlaubt. Es ist daher nicht möglich, *Vergleichsprädikate* zwischen (XML-Schema basisdatentypwertigen) Füllern von Datentyp-Eigenschaften in einer Anfrage zu spezifizieren. Wir können somit keine OWL-QL-Anfrage schreiben, die beispielsweise Paare von Personen gleichen Alters liefert.
- c) OWL bietet *keine nichtmonotonen Operatoren*. Aufgrund der OWA ist es in OWL-QL nicht möglich, eine Anfrage zu stellen, die alle Frauen liefert, von denen *momentan nicht bekannt ist bzw. nicht bewiesen werden kann*, dass sie Mütter sind (eine solche Anfrage ist nichtmonoton - sollte eine Frau zu einem späteren Zeitpunkt doch Mutter werden, wird diese Instanz in der Antwortmenge nicht mehr auftauchen, d.h., die Antwortmenge ist geschrumpft). Wir werden später einen *Negation as Failure (NAF)*-Operator in unsere Anfragesprache semantisch fundiert integrieren.
- d) Das gleiche Argument gilt auch für andere für eine Anfragesprache wichtige Erweiterungen, wie z.B. *negierte Rollen* (bzw. *negierte Properties*). Wir können in OWL-QL keine Anfrage stellen, die die Paare von Individuen (x, y) liefert, sodass x *beweisbar nicht* der Vater von y ist. Man beachte, dass *beweisbar nicht* eine klassische Negationssemantik hat (und keine NAF-Semantik).

All diese Erweiterungen sind in einer Anfragesprache realisierbar, fehlen jedoch (zumeist aus Gründen der Entscheidbarkeit) in OWL und OWL-QL. Wir werden demonstrieren, wie diese Anfragemöglichkeiten in einer SEMANTIC WEB-Anfragesprache zur Verfügung gestellt werden können, und wollen diese Ausdrucksmittel daher beim Entwurf unserer Anfragesprache berücksichtigen.

Im SEMANTIC WEB muss mit hunderttausenden von (durch die Anfragebeantwortungsmaschine zu betrachtenden) Ressourcen bzw. Individuen gerechnet werden – die *Datenintensität* der Domäne steht daher außer Frage. Die Annahme, für eine Anfrage stets eine *vollständige* Antwortmenge zu erhalten, ist nicht zuletzt aus Gründen einer imperfekten Internet-Infrastruktur unrealistisch. Aus diesem Grund ist auch die OWA im SEMANTIC WEB-Kontext sehr sinnvoll, denn es ist unrealistisch anzunehmen, es könne so etwas wie „vollständiges Wissen“ auch nur über einen kleinen Teilbereich des SEMANTIC WEBS geben.

Oftmals werden von Nutzern (womit sowohl menschliche Nutzer als auch Programme bzw. Client-Anwendungen gemeint sind) lediglich einige wenige Ant-

könnte $\geq_{18}$ durch expliziten Ausschluss der Werte von $\{0, \dots, 17\}$ erreicht werden, jedoch nicht für $\mathbb{R}$ -wertiges *has_age*.

worttupel einer Anfrage angefordert, z.B. wenn der Dienst über eine Web-Oberfläche – eine Art „Semantisches Google“ – in Anspruch genommen wird. In OWL-QL finden wir die oben geschilderte Idee des *inkrementellen Nachforderns* von Teilergebnissen einer Antwortmenge mittels eines Client-Server-Dialoges. Es ist daher wichtig, dass ein solcher Dienst nicht nur eine mengenbasierte *Set-at-a-time-Schnittstelle*, sondern auch eine tupelbasierte *Tuple-at-a-time-Schnittstelle* bietet. Optimal ist eine mengenbasierte Schnittstelle, die es erlaubt, *n*-elementige Teilmengen der Antwortmenge inkrementell an- bzw. nachzufordern. Zudem sollten diese Inkremente *verzögert, also nur bei Bedarf* berechnet werden, da so die *Verfügbarkeit* des SWIR-Dienstes erhöht wird (es wird weniger Last durch die doch sehr berechnungsaufwendigen Inferenzprozeduren generiert).

Leider verfügte RACERPRO über keine derartige Schnittstelle. Offensichtlich muss der SWIR-Dienst die *ABox-Basis-Retrieval-Dienste* (s. Kap. 2.2.2.5) in Anspruch nehmen, insbesondere den Instanzenretrieval-Dienst (also `concept_instances`). Dieser wird jedoch stets die gesamte, vollständige Antwortmenge berechnen und zurückliefern, auch wenn nicht alle Ergebnisse benötigt werden. RACERPRO ist dann also unnötigerweise lange blockiert, und die Verfügbarkeit des SWIR-Dienstes ist in dieser Zeit nicht gegeben. Die Nutzung eines existierenden DL-Systems bringt also auch Probleme mit sich. RACERPRO wurde nicht für diese Anforderungen entwickelt. Wir wollen daher die Anforderung *inkrementelles Instanzenretrieval* bzw. *inkrementelle concept_instances-Implementation* für das MIDELORA-System auf die Liste der Anforderungen setzen und zudem unsere Implementation so gestalten, dass das Basis-DL-System bei Bedarf einfach als Komponente betrachtet und ausgetauscht werden kann. *Lose Kopplung* ist hier das Stichwort. Durch Verwendung von MIDELORA soll jedoch auch eine starke Kopplung und somit die in Kap. 1 anvisierte hohe Kohäsion realisiert werden.

Zudem sollte der SWIR-Dienst auch *mehrbenutzerfähig* sein.

Da OWL-Anfragebeantwortung i.d.R. NEXPTIME-vollständige Inferenzprozeduren anstoßen muss, wird deutlich, dass die Verfügbarkeit eines solchen Servers im Mehrbenutzerbetrieb aufgrund der Berechnungskomplexität nur durch spezielle Maßnahmen gegeben sein wird. Es wird daher essentiell sein, dass *Ressourcen sparsam verwendet werden*. Eine Maßnahme hatten wir schon erläutert: *Inkrementelle, „faule“ Beantwortung bei Bedarf*. Weiterhin können *unvollständige Inferenzprozeduren* Approximationen der Antwortmenge berechnen. Speziell für eine inkrementelle Schnittstelle liegt daher die Idee nahe, zunächst eine (maximal große Teilmenge) dieser Antwortmenge mittels berechnungsgünstiger, unvollständiger Verfahren (inkrementell) zu berechnen und auszuliefern. Werden nun noch weitere Inkremente der Antwortmenge angefordert, so müssen die teuren (NEXPTIME-vollständigen) Inferenzprozeduren zur

Vervollständigung der Antwortmenge verwendet werden. Wir sprechen hier von einer *zweiphasigen Berechnung der Antwortmenge*: Zunächst werden die (mittels unvollständigen Methoden errechneten) „billigen“ Tupel ausgeliefert, und dann erst bei Bedarf bzw. auf Anforderung die (mittels vollständiger Methoden errechneten) „teuren“ Tupel zur Komplettierung der Antwortmenge.

Die Stärke von RACERPRO bzgl. der konkreten Domäne ist leider auch gleichzeitig eine potentielle Schwäche in diesem Nutzungsszenario: Während SPARQL beispielsweise ein FILTER-Prädikat zum Test auf Enthaltensein einer Zeichenkette in einer anderen Zeichenkette anbieten kann, fehlen diese für eine Anfragesprache relevanten Prädikate in $ALCQHI_{\mathcal{R}^+}(\mathcal{D}^-)$ aus Gründen der Entscheidbarkeit.

Zudem ist die Bereitstellung von $\mathcal{D}^-$ wesentlich aufwendiger (und damit berechnungssteuerter) als die Bereitstellung von $\mathcal{D}^{n--}$, wie es für OWL DL ausreichen würde. Da RACERPRO die Datatype Properties nun in der konkreten Domäne repräsentiert, können Anfragekonzepte (unter Verwendung des Prädikats-Existenz-Operators) wie $\exists(has\_age). \geq_{18}$ zum Retrieval verwendet werden, was auf der einen Seite ein großer Vorteil ist (komplexe Prädikate können formuliert werden, und Inferenz bzgl. $\mathcal{D}^{n--}$ ist möglich).

Auf der anderen Seite ist diese Kodierung jedoch auch mit Nachteilen behaftet. Füller von OWL Properties sind nämlich entweder explizit modelliert und somit bekannt, oder bleiben vollständig unspezifiziert. Im ersten Fall ist nun *Model Checking* ausreichend, um Retrieval mit FILTER-Prädikaten wie in SPARQL zu ermöglichen. Im zweiten Fall kann aufgrund der Beschränkungen von OWL Retrieval ebenfalls einfacher ermöglicht werden, da OWL weder binäre Prädikate noch benutzerdefinierte Prädikate in der konkreten Domäne bietet. Aus diesem Grund sind komplizierte Inferenzen bzgl. der konkreten Domäne, wie z.B. in Kap. 2.2.5 anhand der Folgerung *birgit* : $\exists(age). \geq_{18}$ demonstriert, zur OWL-Anfragebeantwortung nicht erforderlich.

Zum einen fehlen also wünschenswerte FILTER-Prädikate. Zum anderen ist die Repräsentation von literalen Datenwerten in der konkreten Domäne in gewisser Weise *zu schwergewichtig*.

Eine *pragmatische Lösung* (für beide Probleme) ist wiederum die Verwendung einer *hybriden Repräsentation*. Wir können einer RACERPRO-ABox einfach eine weitere Repräsentationsebene zur Speicherung der literalen „Daten“ eines OWL-Dokuments zur Seite stellen. Eine entspr. hybride Anfragesprache wird dann (wie bereits für die GIS-Domäne vorgeschlagen) entsprechende FILTER-Prädikate bzw. *Anfrage-Atome* für diese Ebene bereitstellen, die durch einfaches Model Checking ausgewertet werden. So können sowohl die für eine Anfragesprache geforderten Prädikate bereitgestellt werden, und auch das Repräsentationsproblem wird gelöst.

Ein weiteres Defizit ist, dass RACERPRO momentan ein rein Hauptspeicherbasiertes DL-System ist. Wir wollen daher für MIDELORA auch die Konsequenzen untersuchen, die sich ergeben, wenn eine relationale Datenbank zur Realisierung einer persistenten (und evtl. besser skalierbaren) Speicherschicht verwendet wird. Tatsächlich sind heutzutage auch noch viele Triplestores rein Hauptspeicherbasiert.

3.2.4.1. Ein praxisrelevantes SWIR-Szenario – Der Lehigh University Benchmark

Der sog. „*Lehigh University Benchmark*“ (LUBM) wurde zur Evaluierung der Performanz und Vollständigkeit der Anfragebeantwortung von „SEMANTIC WEB Repositories“ in DAML+OIL (dem OWL-Vorgänger) definiert; inzwischen wird OWL verwendet [Guo u. a. 2004a;b]. Die LUBM-Ontologie ist im Anhang A.1 in KRSS-Syntax angegeben; zudem ist die Taxonomie in A.2 gezeigt. Der LUBM ist nicht nur zur Performanzevaluierung interessant, sondern stellt gleichzeitig ein interessantes SWIR-Nutzungsszenario bereit – das eines *Universitäts-ISS*. Der LUBM besteht aus

1. einer *OWL-DL-Ontologie*, die eine Universitätsdomäne modelliert (hier finden sich OWL-Klassen für Professoren, Studenten, Kurse, Arbeitsbereiche etc.),
2. einem *Java-Programm*, das umfangreiche (zufällige) RDF-Instanzenmengen dieser OWL-Klassen generiert (also eine Menge von Professoren, Studenten, etc.), und schließlich
3. einer Menge von *14 Benchmarking- bzw. Referenz-Anfragen*, die (nach Meinung der Autoren) typische ontologiebasierte Anfragen an einen solchen „Datenbestand“ darstellen.

Die Autoren präsentieren ihre Referenz-Anfragen (in einer KIF-artigen Syntax) als einfache konjunktive Anfragen der Art

```
(type GraduateStudent ?X)
(type University ?Y)
(type Department ?Z)
(memberOf ?X ?Z)
(subOrganizationOf ?Z ?Y)
(undergraduateDegreeFrom ?X ?Y)
```

Hier werden also Tripel $?x$, $?y$, $?z$ gesucht, sodass $?x$ Instanz der OWL-Klasse GraduateStudent, und $?x$ ist ein Mitglied (memberOf) eines Departments $?z$ ist, welches eine Unterorganisation (subOrganizationOf) derjenigen Universität (University) $?y$ ist, von der $?x$ sein Vordiplom erhalten hat (undergraduateDegreeFrom).

Diese Anfragen können mit einer RDF(S)-Anfragesprache *nicht* vollständig beantwortet werden, da in der OWL-Ontologie komplexe Klassendefinitionen für das Vokabular vorliegen. Beispielsweise fragt die Anfrage 12 nach Instanzen der Klasse Chair. Ein Chair ist definiert als eine Person, die Leiter einer Abteilung ist. In DL-Syntax entspricht dieser OWL-Klassendefinition dem TBox-Axiom

$$\text{chair} \triangleq \text{person} \sqcap \exists \text{headOf}.\text{department}.$$

In den vom Generator-Programm generierten RDF-Datenbeständen sind jedoch keine expliziten Chair-Instanzen vorhanden, wohl aber Instanzen der Klasse Professor (welche wiederum auch Instanzen von Person sind), die mit Instanzen von Department in einer headOf-Beziehung stehen und daher (per Inferenz) als Instanzen von Chair erkannt werden müssen. Eine RDF(S)-Anfragesprache kann daher keine Ergebnisse für diese Anfrage liefern.

Würde die Anfrage (type Chair ?X) jedoch *expandiert* und gegen

(type Person ?X)
(type Department ?Y)
(headOf ?X ?Y)

ersetzt, so können solche Instanzen gefunden werden [Möller u. a. 2006b]. Wir hatten in Kap. 1 erläutert, dass eine ähnliche Technik der Anfrage-Expansion auch im QUONTO-System verwendet wird [Acciarri u. a. 2005].

Die Autoren evaluieren in [Guo u. a. 2003, Pan u. Heflin 2003, Guo u. a. 2004b] Performanz und Vollständigkeit einer Reihe von SEMANTIC WEB-Repositorien.

Wir nehmen also als weitere Anforderung mit auf, dass wir mit der zu entwerfenden ABox- bzw. SEMANTIC WEB-Anfragesprache die LUBM-Anfragen vollständig beantwortet werden sollen.

3.2.5. Fazit

Die Diskussion machte deutlich, dass die Realisierung eines SWIR-Dienstes mit DL-Technologie (DLs und DL-Systemen wie RACERPRO) ein nichttriviales Unterfangen ist. Wir haben eine Reihe von Anforderungen an einen solchen SWIR-Dienst skizziert. Es wird eine ausdrucksstarke Anfragesprache benötigt, die in einigen Punkten über das hinausgehen soll, was beispielsweise durch SPARQL und/oder OWL-QL geboten wird. Zudem sind Mehrbenutzerfähigkeit, inkrementelle und ressourcenkonfigurierbare Berechnung von Teilergebnissen sowie performante und vollständige Beantwortung der LUBM-Anfragen auf der Liste der Anforderungen. Wir hatten uns entschieden, eine ABox-Anfragesprache (mit den geforderten Eigenschaften) als Basis für die SWIR-Anfragesprache bereitzustellen, und diese Entscheidung auch motiviert.

Ein OWL-Beweiser wird als Komponente des SWIR-Dienstes benötigt. Auf der einen Seite ist die Implementation eines OWL-Beweisers ein nichttriviales Unterfangen; insofern sollte existierende Technologie (wie RACERPRO) im Rahmen eines „Bottom Up“-Ansatzes und zur Evaluierung der Praxistauglichkeit solcher Systeme zur Realisierung eines SWIR-Dienstes verwendet werden. Auf der anderen Seite sind Beweiser wie RACERPRO unter anderen Anforderungen entwickelt worden, und es stellt sich die Frage, ob nicht *Spezial-Beweiser* einige dieser Anforderungen besser umsetzen können (dies wäre der „Top Down“-Ansatz). Um letztlich beide Perspektiven im zu entwerfenden Rahmenwerk unterstützen zu können, sollte der OWL-Beweiser in diesem Szenario als *austauschbare Komponente* betrachtet werden. Die ABox-Anfragesprache sollte daher nicht nur für beliebige DLs *definierbar*, sondern auch mittels beliebiger DL-Systeme *realisierbar* sein.

Um die benötigte Flexibilität bereitzustellen, wird die Anfragesprache daher auf der Basis eines eigenen Datenmodells definiert. Das Datenmodell erlaubt mittels hybrider Repräsentation die Entschärfung der Datenwert-Repräsentationsproblematik. Die hybride Anfragesprache entschärft das Problem der fehlenden FILTER-Prädikate. Ein eigenes Datenmodell ist auch nicht zuletzt deswegen erforderlich, weil verschiedene OWL-Beweiser wahrscheinlich verschiedene Basis-Repräsentationen für z.B. literale Datenwerte in OWL-Dokumenten verwenden werden (die Repräsentation in der konkreten Domäne einer ABox ist keinesfalls zwingend); *Datenunabhängigkeit* ist hier das Stichwort. Die Bereitstellung flexibler Abstraktionen ist in dieser Arbeit sehr wesentlich.

Mittels des MIDELORA-Systems wollen wir im Rahmen des „Top Down“-Ansatzes zudem die Punkte *echte Inkrementalität* von Basis-ABox-Retrievaldiensten (u.a. von `concept_instances`) angehen, und auch datenbankbasierte ABoxen untersuchen, um das Skalierbarkeitsproblem zu adressieren. Hier ist eine hohe Kohäsion bzw. enge Kopplung anzustreben.

3.3. Zusammenfassung

Aus der Darstellung und Analyse der beiden Leitdomänen haben wir eine Reihe von *gemeinsamen Anforderungen* an ontologiebasierte ISe in diesen Leitdomänen gewonnen, die die weiteren Arbeitspunkte dieser Arbeit wie folgt motivieren:

- Es wird ein flexibles, bei Bedarf *hybrides*, semistrukturiertes graphbasiertes Repräsentationsmedium bzw. Datenmodell benötigt. Wir wollen prinzipiell auch Persistenz ermöglichen. Das Datenmodell soll zudem eine formale Semantik haben, sodass die Semantik der Anfragesprachen, die wir für dieses Datenmodell bzw. spezielle *Instantiierungen* des Datenmodells definieren wollen, formal definiert werden kann. Wir werden Prädikatenlogik erster Ordnung als hinreichend mächtigen Semantik-Beschreibungsrahmen sowohl für das Datenmodell als auch für die Anfragesprachen nutzen.

Das Datenmodell soll *flexibel* sein: Es soll sowohl ABoxen, RDF-Graphen, RCC-Netzwerke, die Karten-Objekte der DISK (welche Instanzen räumlicher Datentypen sind), als auch die literalen Datenwerte eines OWL-Dokuments darstellen können. Spezielle Indexstrukturen sollen für spezielle Instantiierungen des Datenmodells angeboten werden, z.B. räumliche Indexstrukturen für die DISK-Instantiierung. Das Datenmodell muss bei Bedarf also Objekte (bzw. Knoten im Graphen) mit *reicher innerer Struktur* darstellen können, also *reichhaltige Repräsentationen* ermöglichen. Dies ist für die Repräsentation der DISK, oder für die *räumlichen ABoxen* notwendig. Aus softwaretechnischer Perspektive soll das Datenmodell zudem eine einheitliche, *abstrakte Sicht* auf die unterschiedlichen ABox-Repräsentationen verschiedener, existierender DL-Systeme bereitstellen und an die Dienste dieser Systeme ankoppeln können.

Ist es wirklich nötig, hierfür ein eigenes Datenmodell zu definieren? Wir meinen ja. Datenmodelle wie RDF(S) sind für die hier betrachteten IS-Domänen prinzipiell anwendbar, doch sind reichhaltige Repräsentationen nicht ohne weiteres realisierbar. Durch Nutzung von RDF(S)- bzw. von existierenden RDF(S)-Rahmenwerken könnten wir nicht die für diese Arbeit erforderliche Breite und Flexibilität (in der zur Verfügung stehenden Zeit) realisieren. Beispielsweise ist nicht klar, wie RDF(S) zur Repräsentation von DISK-Karten-Objekten bzw. Instanzen räumlicher Datentypen genutzt werden kann. Es erscheint nicht sinnvoll, die Geometrie dieser Objekte mittels RDF(S)-Prädikaten beschreiben zu wollen (es existieren für diese Zwecke Sprachen wie GeoML). Letztlich muss die beschriebene Geometrie bei der Anfragebeantwortung mittels räumlicher Indexstrukturen genutzt werden. Es ist nicht ersichtlich, wie dies mit RDF(S)-

Anfragesprachen ermöglicht werden kann.

Für unser Datenmodell wollen wir zudem nicht nur Kanten-Annotationen, sondern auch Knoten-Annotationen zulassen (letztlich können natürlich auch Knoten-Annotationen – also Attribut-Wert bzw. Eigenschaft-Wert-Paare – als Kanten-Annotationen repräsentiert werden; in OWL werden Rollen ja auch als „Properties“ bezeichnet). Unter der Perspektive, dass Knoten eine reiche innere Struktur haben, ist diese Sichtweise jedoch nicht adäquat, da die *Struktur des Knotens selbst* nicht recht als Menge von Eigenschaft-Wert-Paaren verstanden werden kann.

Aus diesen Gründen erscheint ein eigener, formaler und softwaretechnischer Rahmen zur prototypischen Realisierung der Leitdomänen in dieser Arbeit notwendig. Wir möchten betonen, dass dieser Rahmen nicht primär durch theoretische, sondern durch *pragmatische und softwaretechnische Gründe* motiviert ist. Die Argumentation machte deutlich, dass es sich hierbei um Aspekte handelt, denen mindestens ebensoviel Gewicht bei der Realisierung von ontologiebasierten ISen wie den rein theoretischen Aspekten zugesprochen werden sollte, zumal derartige Aspekte in der heutzutage primär theoriegetriebenen Forschung in diesem Gebiet oftmals vernachlässigt werden.

- Es wird eine flexible Anfragesprache bzw. ein *Rahmen zur Definition von Anfragesprachen* benötigt. Dieser Rahmen muss zu dem flexiblen Datenmodell passen. Anfragesprachen sollen bei Bedarf hybrid sein können, und für verschiedene Spezialisierungen bzw. Instantiierungen des Datenmodells flexibel anpassbar und erweiterbar sein. Beispielsweise wird eine Anfragesprache für die GIS-Domäne räumliche Anfragekonstrukte enthalten. Die Anfragesprache soll zudem formalisier- und effizient implementierbar sein.

Der Rahmen für die Anfragesprachen muss also flexibel genug sein, um die Realisierung einer ABox-Anfragesprache, einer ontologiebasierten DISK-Anfragesprache, als auch einer Anfragesprache für den SWIR-Dienst zu ermöglichen. Existierende Anfragesprachen haben in unseren Kontexten entweder Defizite, oder weisen nicht die notwendige Generizität bzw. Flexibilität auf.

- *Spezial-Beweiser* können gewisse bei der Realisierung ontologiebasierter ISe auftauchende Probleme besser adressieren, als dies mit Standard-DL-Systemen wie RACERPRO möglich wäre. Diese Spezial-Beweiser implementieren entweder Spezial-DLs oder aber *spezielle Inferenzverfahren*, wie beispielsweise das *inkrementelle Instanzenretrieval*, oder das Instanzenretrieval auf *räumlichen ABoxen*. Dies motiviert die Schaffung des MIDELORA-Systems, welches langfristig auch zum OWL-System ausgebaut werden soll.

MIDELORA wird zudem so entworfen, dass es direkt auf dem zu entwerfenden flexiblen Datenmodell arbeitet. Eine MIDELORA-ABox kann dann als Instanz einer Spezialisierung des Datenmodells betrachtet werden. Durch die Verwendung gemeinsamer Repräsentations- und Indexstrukturen, die sowohl von der ontologiebasierten Anfragebearbeitungsmaschine als auch von den hierfür benötigten MIDELORA-Beweisern genutzt werden, vermeiden wir Modellbrüche durch unterschiedliche Repräsentationen und schaffen eine hohe Kohäsion zwischen der Repräsentations-, Anfrage-, und Inferenzschicht, wie in Kap. 1 diskutiert. Das System ist dann „aus einem Guss“ und vermeidet Ineffizienzen und Modellbrüche an den Schnittstellen, und kann insbesondere flexibel an unterschiedliche IS-Domänen adaptiert werden, sofern die benötigten formalen und softwaretechnischen Rahmenwerke entsprechend entworfen werden. Insbesondere sollen die Inferenzverfahren, die zur ontologiebasierten Anfragebeantwortung durch die Anfragebeantwortungsmaschine gerufen werden, ebenfalls in MIDELORA definiert werden können.

4. Ein formales Rahmenwerk für ontologiebasierte Informationssysteme

Inhaltsangabe

4.1	Das Substrat-Datenmodell	193
4.1.1	Substrate mit Hintergrundtheorie	195
4.1.2	Flexibilität des Datenmodells	195
4.1.3	Inferenzprobleme für Substrate	201
4.1.4	Substrate für die Leitdomänen	201
4.2	Eine generische Anfragesprache für Substrate	202
4.2.1	Die Basis-Idee	202
4.2.2	Atomare Anfragen	205
4.2.3	Vereinfachte Semantik für einfache Substrate	208
4.2.4	„Negation as Failure“ in der Anfragesprache	209
4.2.5	Komplexe Anfragen	211
4.2.6	Semantikerhaltende Transformationen	215
4.2.7	Deutung der Semantik mit prädikatenlogischen Mitteln	219
4.2.8	Eine Logikprogrammierungs-Perspektive	222
4.2.9	Oft verwendete Anfrage-Idiome	222
4.2.10	Eine alternative algebraische Semantik	223
4.2.11	Inferenzprobleme für Anfragen	226
4.2.12	Hybride Anfragen	228
4.2.13	Anfragesprachen für die Leitdomänen	229
4.3	Zusammenfassung	230

Wir hatten in Kap. 3 motiviert, dass zur Realisierung der extensionalen Komponente eines ontologiebasierten ISs mittels eines flexiblen softwaretechnischen Rahmens ein semistrukturiertes und bei Bedarf hybrides Datenmodell wünschenswert ist. Ein Datenmodell ist nach [Rechenberg u. Pomberger 2006, Kap. E4.2.1]

... ein Satz von abgestimmten Konzepten zur Beschreibung der Repräsentation von Informationen durch Daten.

Der Argumentation aus Kap. 2 und 3 folgend, sollte solch ein Satz von abgestimmten Konzepten formal sein. Wir hatten in Kap. 3 durch Analyse der

beiden Leitdomänen Anforderungen an das Datenmodell definiert: Das Datenmodell muss sowohl ABoxen, RDF-Graphen, RCC-Netzwerke, die Karten-Objekte der DISK (welche Instanzen räumlicher Datentypen sind), als auch die literalen Datenwerte eines OWL-Dokuments darstellen können. Solche extensionalen „Datenbestände“ können als Instanzen einer *Spezialisierung*, wir sagen auch *Instantiierung*, des generischen Datenmodells dargestellt werden. Eine solche Instanz bzw. extensionaler Datenbestand wird im Folgenden ein *Substrat* genannt, wodurch die Sichtweise betont wird, dass eine solche Instanz ein „Träger von Information“ ist. In der Biologie ist ein Substrat ein Nährboden; im allgemeineren Sinne handelt es sich um eine „materielle Grundlage“. Solche Substrate sind wiederum Instanzen von *Substrat-Typen* bzw. -*Klassen*. Ein Substrat-Typ wird eine *Instantiierung des Substrat-Modells* genannt.

Wir beschreiben in diesem Kapitel die konzeptuellen Aspekte des Datenmodells (den *Knowledge Level*). Später werden wir zeigen, dass das Datenmodell auch graphartig implementiert wird (den *Symbol Level*); so wird ein *Impedance Mismatch* vermieden.

Das Substrat-Datenmodell ist durch die in [Sattler 1996b, Horrocks 1997b] verwendeten Tableau-Datenstrukturen bzw. Techniken aus der Modallogik, RDF und letztlich auch durch die sog. *$\mathcal{E}$ -Verbindungen* (*$\mathcal{E}$ -Connections*) und *abstrakten Beschreibungs-Systeme* (*Abstract Description Systems, ADS*) inspiriert [Kutz u. a. 2004]; die Details weichen jedoch ab. Natürlich kann der Autor hier nicht für sich in Anspruch nehmen, mit logischen Formelmengen beschriftete Graphen „erfunden“ zu haben – ganz ähnliche Modelle werden schon seit den frühen 1960er Jahren in der KI verwendet (s. auch Kap. 2.2.1). Wir behaupten nicht, dass dieses Modell *aus theoretischer Perspektive* interessant ist, werden jedoch belegen, dass es aus *pragmatischer Perspektive* interessant ist (dies gilt auch für RDF(S)). Insbesondere liegt der Fokus des Datenmodells auf der extensionalen Ebene, nicht auf der intensionalen. Insofern ist es auch nicht mit intensionalen Wissensrepräsentationen wie z.B. *Conceptual Graphs* [Sowa 1984], Semantischen Netzen o.Ä. vergleichbar. Der intensionale Anteil einer Wissensbasis wird im Substrat-Datenmodell als *Hintergrundtheorie* (ebenfalls prädikatenlogisch) hinterlegt. Primärer Zweck des Substrat-Datenmodells ist die Aufspannung eines einheitlich konzeptuellen, generischen und formalen Rahmens.

Die für dieses Datenmodell bereitgestellte *generische Substrat-Anfragesprache* muss der *Flexibilität und Generizität* des Datenmodells Rechnung tragen. Ein einheitlicher, formaler und softwaretechnischer Rahmen wird die formale Definition und softwaretechnisch saubere Realisierung unterschiedlicher *Instantiierungen von Anfragesprachen* bzw. von *Elementen einer Familie von Anfragesprachen*, passend zu den verschiedenen Instantiierungen des Substrat-Datenmodells, ermöglichen.

4.1. Das Substrat-Datenmodell

Wir definieren nun ein Substrat wie folgt:

Definition 55 (Substrat) Ein *Substrat* ist ein knoten- und kantenbeschrifteter Graph $(V, E, L_V, L_E, \mathcal{L}_V, \mathcal{L}_E)$, wobei V die Menge der *Knoten* (Vertices), und $E \subseteq V \times V$ die Menge der *Kanten* (Edg es) ist.

Die *Knoten-Beschriftungsfunktion* $L_V : V \rightarrow \mathcal{L}_V$ bildet Knoten auf Ausdrücke einer Knoten-Beschriftungssprache $\mathcal{L}_V$ ab. Die *Kanten-Beschriftungsfunktion* $L_E : E \rightarrow \mathcal{L}_E$ bildet Kanten auf Ausdrücke einer entsprechenden Kanten-Beschriftungssprache $\mathcal{L}_E$ ab.

Für gegebenes $i \in V$ nennen wir $L_V(i)$ auch die *Knoten-Beschriftung* oder *Knoten-Annotation*, und analog für die Kanten $(i, j) \in E$ und $L_E((i, j))$. ■

Die Sprachen $\mathcal{L}_V$ und $\mathcal{L}_E$ sind anwendungsspezifisch und können für spezielle Instantiierungen den Anforderungen entsprechend definiert werden. In der Regel wird es sich hierbei um variablenfreie Sprachen handeln. Für $\mathcal{L}_V$ sind beispielsweise Modallogiken oder DLs denkbar wie in [Kutz u. a. 2004]; beispielsweise kann $\mathcal{L}_E$ durch eine Menge von DL-Rollenamen ($\mathcal{N}_R$) gegeben sein. Es ist für unsere Zwecke nicht erforderlich, diese Sprachen genauer festzulegen, denn zunächst soll hier lediglich ein Beschreibungsrahmen bereitgestellt werden, der die Implementierung leiten soll. Die Sprachen $\mathcal{L}_V$ und $\mathcal{L}_E$ können auch durch Aussagenlogik $\mathcal{L}_{PROP}$ gegeben sein, oder eine Menge von $\mathcal{ALC}$ -Konzepten und eine Menge von Rollenamen $\mathcal{N}_R$.

Die *Semantik* eines Substrates kann aus einer prädikatenlogischen Sichtweise verstanden werden. Unter dieser Sichtweise entsprechen den Knoten prädikatenlogische Konstanten. Eine Knoten-Annotation $v = L_V(i)$ für einen Knoten i entspricht einem prädikatenlogischen Satz erster Ordnung in Pränormalform (s. Def. 2), dessen Matrix in DNF ist, sodass jedes Disjunkt der Matrix die Konstante i referenziert. Die Kanten-Annotationen können ähnlich verstanden werden, jedoch wird $v = L_E((i, j))$ eine *Grundformel* in DNF sein, sodass die einzelnen Disjunkte *binäre Grundatome der Bauart* $R(i, j)$ sind (also enthält v weder Variablen noch Quantoren). Wie auch in Kap. 2.2.2.6 für DLs dargestellt, kann diese prädikatenlogische Übersetzung durch eine ϕ -Übersetzungsfunktion gegeben sein.

Insofern kann ein Substrat einfach als eine Menge prädikatenlogischer Formeln einer (durch die Instantiierung bestimmten) gewissen Bauart verstanden werden, und L_V und L_E sind aus dieser Perspektive einfach *Indexfunktionen*.

Zur Verdeutlichung sei ein *aussagenlogisches Substrat* S betrachtet: Hier sind $\mathcal{L}_V$ und $\mathcal{L}_E$ durch Aussagenlogik $\mathcal{L}_{PROP}$ gegeben. Sei beispielsweise $L_V(i) = C \wedge (D \vee \neg E)$. Offensichtlich ist $C \wedge (D \vee \neg E) \in \mathcal{L}_{PROP}$, $\Phi(L_V(i))$ ist nun jedoch eine

4.1 Das Substrat-Datenmodell

prädikatenlogische Grundformel: $C(i) \wedge (D(i) \vee \neg E(i))$, bzw. $(C(i) \wedge D(i)) \vee (C(i) \wedge \neg E(i))$. Offensichtlich ist diese Formel in DNF, und jedes Disjunkt referenziert i (sogar in jedem Konjunkt).

Ein Substrat ist somit generell genug, um sowohl RDF-Graphen als auch ABoxen darzustellen. Für die GIS-Domäne können wir annehmen, dass Knoten Instanzen räumlicher Datentypen sind, deren Geometrie mittels einer räumlichen Beschreibungssprache $\mathcal{L}_{\mathcal{V}_R}$, und deren Thematik mittels einer thematischen Beschreibungssprache $\mathcal{L}_{\mathcal{V}_T}$ beschrieben ist, sodass $\mathcal{L}_{\mathcal{V}} = \mathcal{L}_{\mathcal{V}_T} \cup \mathcal{L}_{\mathcal{V}_R}$, mit $\mathcal{L}_{\mathcal{V}_T} \cap \mathcal{L}_{\mathcal{V}_R} = \emptyset$.

Die Semantik eines Substrates ist wie folgt definiert:

Definition 56 (Semantik eines Substrates) Sei S ein Substrat $(V, E, L_V, L_E, \mathcal{L}_{\mathcal{V}}, \mathcal{L}_{\mathcal{E}})$, und $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ eine Tarski-Interpretation, sodass $\cdot^{\mathcal{I}}$

1. Elemente von V auf Elemente in $\Delta^{\mathcal{I}}$,
2. Ausdrücke in $\text{sub}(\bigcup_{i \in V} \{L_V(i)\})$ auf Teilmengen von $\Delta^{\mathcal{I}}$, und
3. Ausdrücke in $\text{sub}(\bigcup_{(i,j) \in E} \{L_E((i,j))\})$ auf Teilmengen von $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$

abbildet werden.

Hierbei bezeichnet $\text{sub}(\mathcal{X})$ für eine Menge von Ausdrücken $\mathcal{X} \subseteq \mathcal{L}_{\mathcal{V}}$ (bzw. $\mathcal{X} \subseteq \mathcal{L}_{\mathcal{E}}$) die Menge aller Teilausdrücke von Elementen aus $\mathcal{X}$:

$$\text{sub}(\mathcal{X}) =_{\text{def}} \bigcup_{x \in \mathcal{X}} \text{sub}(x).$$

Da für $x \in \mathcal{X}$ stets $x \in \text{sub}(x)$ gelte, gilt auch $\mathcal{X} \subseteq \text{sub}(\mathcal{X})$. ■

Wie üblich, wird eine (ein Substrat) erfüllende Tarski-Interpretation ein Modell genannt:

Definition 57 (Modell-Beziehung für Substrate) Eine Tarski-Interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ ist ein Modell eines Substrates $(V, E, L_V, L_E, \mathcal{L}_{\mathcal{V}}, \mathcal{L}_{\mathcal{E}})$, $\mathcal{I} \models S$, wenn gilt:

1. Für $i \in V$ gilt stets $i^{\mathcal{I}} \in L_V(i)^{\mathcal{I}}$.
2. Für $(i,j) \in E$ gilt stets $(i^{\mathcal{I}}, j^{\mathcal{I}}) \in L_E((i,j))^{\mathcal{I}}$.
3. $\mathcal{I}$ ist ein Modell für $\bigcup_{i \in V} \{L_V(i)\}$.
4. $\mathcal{I}$ ist ein Modell für $\bigcup_{(i,j) \in E} \{L_E((i,j))\}$. ■

Ein Substrat, welches ein Modell hat, heißt erfüllbar (und andernfalls unerfüllbar). Offensichtlich erfordert die obige Definition, dass

$$\mathcal{I} \models \bigcup_{i \in V} \{L_V(i)\} \cup \bigcup_{(i,j) \in E} \{L_E((i,j))\}$$

definiert ist. Entweder muss für eine Instantiierung des Modells daher „ $\models$ “ für verwendetes $\mathcal{L}_{\mathcal{V}}$ und $\mathcal{L}_{\mathcal{E}}$ sprachspezifisch festgelegt werden, oder eine ϕ -

Übersetzungsfunktion angegeben werden. In diesem Falle müssen die Knoten V als prädikatenlogische Konstanten interpretiert werden, wie oben diskutiert, und entsprechende Bedingungen an ϕ werden gestellt. Wir können dann festsetzen, dass $\mathcal{I} \models \bigcup_{i \in V} \{L_V(i)\} \cup \bigcup_{(i,j) \in E} \{L_E((i,j))\}$ genau dann gilt, wenn für alle $i \in V$ $\mathcal{I} \models \Phi(L_V(i))$, und für alle $(i,j) \in E$ $\mathcal{I} \models \Phi(L_E((i,j)))$ gilt. Zusätzlich wird dann gefordert, dass $L_V(i)^{\mathcal{I}} = \{j \mid j \in V, \mathcal{I} \models \phi(L_V(i))_{i \leftarrow j}\}$ gilt, und analog für die Kanten.

4.1.1. Substrate mit Hintergrundtheorie

Oftmals müssen noch zusätzliche Bedingungen an die Modelle $\mathcal{I}$ eines Substrates gestellt werden. Beispielsweise sollte ein ABox-Substrat eine TBox referenzieren können, deren Axiome weitere Bedingungen für $\cdot^{\mathcal{I}}$ festlegen. Eine solche TBox kann, wie in Kap. 2.2.2.6 erläutert, in eine Menge universeller prädikatenlogischer Sätze überführt werden. Wir definieren daher, was ein Modell eines Substrates mit Hintergrundtheorie ist:

Definition 58 (Modell eines Substrates mit Hintergrundtheorie) Eine Hintergrundtheorie $\mathcal{T}$ ist eine Menge prädikatenlogischer *universeller Sätze* (s. Def. 2).

Sei S ein Substrat, und $\mathcal{T}$ eine Hintergrundtheorie. Dann ist $\mathcal{I}$ ein Modell von $(S, \mathcal{T})$, $\mathcal{I} \models (S, \mathcal{T})$ gdw. $\mathcal{I} \models S$ und $\mathcal{I} \models \mathcal{T}$ gilt. ■

Die Sprache $\mathcal{T}$ ist ebenfalls anwendungs- bzw. instantiierungsspezifisch, und es muss wieder festgelegt werden, wann $\mathcal{I} \models \mathcal{T}$ gilt. Die Begriffe Erfüllbarkeit eines Substrates mit Hintergrundtheorie etc. ergeben sich analog.

Konzeptionell kann $\mathcal{T}$ wieder als eine Menge prädikatenlogischer Formeln erster Ordnung betrachtet werden; diese Sichtweise ist jedoch nicht zwingend. Wir stellen uns auf den Standpunkt, dass alle zusätzlichen Axiome, die für spezielle Substrat-Instantiierungen benötigt werden, einfach in $\mathcal{T}$ abgelegt werden können. Für ein RCC-Substrat können wir hier z.B. die Formeln zur Erzwingung *relationaler Konsistenz* (s. Def. 53), oder Formeln zur Erzwingung der UNA hinterlegen.

4.1.2. Flexibilität des Datenmodells

Die Flexibilität des Datenmodells illustrieren wir durch Angabe einiger Instantiierungen. Wie angedeutet, nennen wir hier spezialisierte Typen von Substraten auch Instantiierungen, da diese durch Festlegung bzw. Instantiierung von $\mathcal{L}_V$ und $\mathcal{L}_E$ entstehen. Die folgenden Substrat-Instantiierungen sind für viele

4.1 Das Substrat-Datenmodell

Anwendungen nützlich; wir werden später ihre Nutzung zur Realisierung der IS-Prototypen für die Leitdomänen darstellen.

Wir unterscheiden zwischen logischen (symbolischen), räumlichen, und hybriden Substraten.

4.1.2.1. Logische Substrate

Zunächst betrachten wir das bereits erwähnte *aussagenlogische Substrat*:

Definition 59 (Aussagenlogisches Substrat) Sei $\mathcal{P}_V$ eine Menge von Knotenpropositionen, und $\mathcal{P}_E$ eine Menge von Kantenpropositionen mit $\mathcal{P}_V \cap \mathcal{P}_E = \emptyset$.

Für ein *aussagenlogisches Substrat* lassen wir für $\mathcal{L}_V$ bzw. $\mathcal{L}_E$ die Menge der aussagenlogischen Formeln zu, welche sich unter Verwendung der Propositionen $\mathcal{P}_V$ bzw. $\mathcal{P}_E$ aufbauen lassen. ■

Obwohl die prädikatenlogische Semantik eines aussagenlogischen Substrates aufgrund obiger Erläuterungen klar sein sollte, definieren wir hier zur Illustration, wann eine Interpretation (im Sinne obiger Definitionen) ein Modell einer Menge von Knoten- und Kanten-Beschriftungen ist.

Definition 60 (Semantik eines aussagenlogischen Substrates) Sei $S = (V, E, L_V, L_E, \mathcal{L}_V, \mathcal{L}_E)$ ein aussagenlogisches Substrat, und $(\Delta^I, \cdot^I)$ eine Interpretation für S .

Nun ist $\mathcal{I}$ genau dann ein Modell für $\bigcup_{i \in V} \{L_V(i)\}$, wenn $\mathcal{I} \models w$ für alle $w \in \bigcup_{i \in V} \{L_V(i)\}$ gilt, und $\mathcal{I} \models w$ gilt genau dann, wenn die Interpretationsfunktion folgende *Bedingungen* erfüllt (man beachte, dass wir in Def. 56 gefordert haben, dass auch alle *Teilausdrücke* abgebildet werden):

1. Für $w = \neg P$: $(\neg P)^I = \Delta^I \setminus P^I$
2. Für $w = w_1 \wedge w_2$: $w^I = w_1^I \cap w_2^I$.
3. Für $w = w_1 \vee w_2$: $w^I = w_1^I \cup w_2^I$.

Für die Kanten-Beschriftungen gilt dies analog, allerdings muss hier $(\neg P)^I = (\Delta^I \times \Delta^I) \setminus P^I$ gefordert werden. ■

Eine anderer praxisrelevanter Substrat-Typ ist das *relationale Substrat*. Hiermit können beispielsweise RDF-Graphen dargestellt werden:

Definition 61 (Relationales Substrat) Ein *relationales Substrat* ist ein Substrat mit $\mathcal{L}_V = \emptyset$ und $\mathcal{L}_E = \{R_1, \dots, R_n\}$. Hierbei ist $\{R_1, \dots, R_n\}$ eine Menge von (binären) Relationsnamen (RDF-Prädikaten, Rollen, Kanten-Beschriftungen, ...). ■

Offensichtlich ist ein relationales Substrat ein spezielles aussagenlogisches Substrat; wir brauchen seine Semantik daher nicht zu definieren. Im Gegensatz zum aussagenlogischen Substrat ist ein relationales Substrat offensichtlich immer erfüllbar (seine Modelle sind relationale Strukturen, und es ist in gewisser Weise isomorph zu seinem minimalen Herbrand-Modell).

Zur Darstellung von RCC-Netzen reicht prinzipiell auch ein aussagenlogisches Substrat mit disjunktiven Kanten-Beschriftungen, doch müssen wir dann die Erfüllbarkeit bzgl. einer Hintergrundtheorie betrachten (s. Def. 53, S. 140).

Definition 62 (RCC-Substrat) Ein *RCC-Substrat* ist ein Substrat S mit $\mathcal{L}_V = \emptyset$ und $\mathcal{L}_E$ ist die Menge aller RCC-Relationen (also 2^{RCC}). ■

Es ist offensichtlich, dass eine ABox im Substrat-Datenmodell dargestellt werden kann. Zur Illustration betrachten wir eine $\mathcal{ALC}$ -ABox $\mathcal{A}$. Wir identifizieren dann $\text{inds}(\mathcal{A})$ mit der Menge der Knoten $V_{\mathcal{A}}$ im Substrat, und eine Kante entsteht durch eine vorhandene Rollen-Assertion. Die Beschriftungen ergeben sich auf offensichtliche Weise; offensichtlich wird Konjunktion für Rollen und Konzepte benötigt (natürlich müssen wir $\mathcal{L}_V = \mathcal{ALC}$ setzen, daher kann Konjunktion ausgedrückt werden; für $\mathcal{L}_E$ müssen wir einen Rollenkonjunktionsoperator zur Verfügung stellen).

Proposition 2 (Substrat-Perspektive auf eine ABox) Sei $\mathcal{A}$ eine $\mathcal{ALC}$ -ABox, und $S_{\mathcal{A}}$ das aus $\mathcal{A}$ wie folgt abgeleitete Substrat:

- $S_{\mathcal{A}} = (V, E, L_{V_{\mathcal{A}}}, L_{E_{\mathcal{A}}}, \mathcal{L}_V, \mathcal{L}_E)$
- $\mathcal{L}_V = \mathcal{ALC}$
- Die Menge der Kanten-Beschriftungen $\mathcal{L}_E$ ist definiert als der Abschluss von $\mathcal{N}_{\mathcal{R}}$ (der Menge der $\mathcal{ALC}$ -Rollennamen) unter dem Konjunktionsoperator „ $\sqcap$ “ (denn $\mathcal{ALC}$ kennt keine konjunktiven Rollen).
- $V =_{\text{def}} \text{inds}(\mathcal{A})$
- $E =_{\text{def}} \{ (i, j) \mid (i, j) : R \in \mathcal{A}, R \in \mathcal{N}_{\mathcal{R}} \}$
- für alle $i \in V$, $L_V(i) =_{\text{def}} \sqcap \{ C \mid i : C \in \mathcal{A} \}$
- für alle $(i, j) \in E$, $L_E(i, j) =_{\text{def}} \sqcap \{ R \mid (i, j) : R \in \mathcal{A} \}$

Offensichtlich gilt: $\mathcal{A}$ ist erfüllbar gdw. $S_{\mathcal{A}}$ erfüllbar ist.

Sei $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ ein Modell von $\mathcal{A}$, $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}) \models \mathcal{A}$. Dann ist $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}'})$ ein Modell von $S_{\mathcal{A}}$, wenn wir $\cdot^{\mathcal{I}}$ auf offensichtliche Art und Weise wie folgt zu $\cdot^{\mathcal{I}'}$ erweitern: $(R \sqcap S)^{\mathcal{I}'} =_{\text{def}} R^{\mathcal{I}} \cap S^{\mathcal{I}}$ ($\mathcal{ALC}$ kennt keine Rollenkonjunktion). Hingegen ist bereits jedes Modell $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}) \models S_{\mathcal{A}}$ auch Modell von $\mathcal{A}$: $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}) \models \mathcal{A}$. ■

Die Übersetzung von $\mathcal{A}$ in $S_{\mathcal{A}}$ ist möglich, solange $\mathcal{A}$ lediglich unäre und binäre Assertionen enthält. Andernfalls wären Hyperkanten erforderlich. Wird eine

4.1 Das Substrat-Datenmodell

ABox mit Assertionen aus der konkreten Domäne verwendet, so kann V prinzipiell auch die KD-Individuen enthalten; Attributs-Assertionen entsprechen wieder Kanten. Mehrstellige Prädikats-Assertionen können nicht abgebildet werden.

Sollte die ABox $\mathcal{A}$ über eine assoziierte TBox $\mathcal{T}_A$ verfügen, so verwenden wir Erfüllbarkeit eines Substrates bzgl. einer Hintergrundtheorie, wie erläutert.

4.1.2.2. Räumliche Substrate

Wir hatten motiviert, dass es manchmal sinnvoll ist, die Knoten nicht als einfache symbolische prädikatenlogische Konstanten zu betrachten, sondern als Individuen, die eine *komplexe innere Struktur haben*, also z.B. Instanzen räumlicher Datentypen sind. Für das GIS-Szenario verwenden wir ein *geometrisches Substrat* zur Repräsentation der Geometrie:

Definition 63 (Geometrisches Substrat) Ein *geometrisches Substrat* ist ein spezielles Substrat, dessen Knoten als Instanzen räumlicher Datentypen (Punkte, Linien, Ketten, Polygone) betrachtet werden.

Zur Beschreibung der Geometrie eines Knotens wird eine speziellen Knoten-Beschreibungssprache $\mathcal{L}_{\mathcal{V}_R}$ verwendet, die *Geometrie-Beschreibungssprache*. Die Worte in $\mathcal{L}_{\mathcal{V}_R}$ sind *prädikatenlogische Grundterme*. Dementsprechend sprechen wir auch von *Geometrie-Beschreibungstermen*. Zudem kann $\mathcal{L}_{\mathcal{E}} = \emptyset$ zunächst als leer angenommen werden.

Wir nehmen an, dass $p, l, kette, polygon$ Funktionssymbole sind. Die Menge $\mathcal{L}_{\mathcal{V}_R}$ ist induktiv wie folgt definiert:

1. Jede reelle Zahl ist ein Term.
2. Wenn x und y reelle Zahlen sind, so ist $p(x, y)$ ein Term (Punkt-Konstruktor).
3. Für Terme $t_1 = p(x_1, y_1)$ und $t_2 = p(x_2, y_2)$ mit $(x_1, y_1) \neq (x_2, y_2)$ ist $l(t_1, t_2)$ ein Term (Segment- oder Linien-Konstruktor).
4. Für Terme $t_1 = l(p1_1, p2_1)$ bis $t_n = l(p1_n, p2_n)$, $n > 1$, mit $\forall i \in 2 \dots n. p1_i = p2_{i-1}$ gilt ist $kette(t_1, \dots, t_n)$ ein Term (Ketten-Konstruktor).
5. für Terme $t_1 = l(p1_1, p2_1)$ bis $t_n = l(p1_n, p2_n)$, $n > 2$ mit $\forall i \in 2 \dots n. p1_i = p2_{i-1}$ und $p1_1 = p2_n$ ist $polygon(t_1, \dots, t_n)$ ein Term (Polygon-Konstruktor). ■

Die Idee hinter diesen Annotationen ist, dass Geometrieterme wie $p(10, 20)$ oder $l(p(10, 10), p(20, 20))$ eindeutig ein Objekt im Universum $\Delta^{\mathcal{I}}$, in diesem Falle also eine Teilmenge von $\mathbb{R}^2$ denotieren. Definition 56 fordert ja, dass $i^{\mathcal{I}} \in L_V(i)^{\mathcal{I}}$ ist. Wir wollen daher für $L_V(i)^{\mathcal{I}}$ bzw. die Geometrie-Beschreibungsterme sicherstellen, dass $L_V(i)^{\mathcal{I}}$ stets eine einelementige Teilmenge (Singleton) von $\mathbb{R}^2$ ist. Daher muss beispielsweise $l(p(10, 10), p(20, 20))^{\mathcal{I}} = \{(10, 10), \dots, (20, 20)\}$ und nicht

$l(p(10, 10), p(20, 20))^{\mathcal{I}} = \{(10, 10), \dots, (20, 20)\}$ gelten. Da Terme auf Elemente in $\Delta^{\mathcal{I}}$ abgebildet werden, muss $\Delta^{\mathcal{I}}$ eine komplexe Struktur haben. Normalerweise würden wir $\Delta^{\mathcal{I}} = 2^{\mathbb{R}^2}$ setzen, doch macht die Singleton-Forderung $\Delta^{\mathcal{I}} = 2^{\mathbb{R}^2}$ erforderlich.¹ Wir definieren daher $\mathcal{I} \models \bigcup_{i \in V} \{L_V(i)\}$ wie folgt:

Definition 64 (Modell eines geometrischen Substrates) Ein *geometrisches Substrat* ist erfüllt in einer Struktur $(2^{2^{\mathbb{R}^2}}, \cdot^{\mathcal{I}})$, wenn $\cdot^{\mathcal{I}}$ die folgenden Bedingungen erfüllt:

1. $x^{\mathcal{I}} = \{(x, 0)\}$, für jede reelle Zahl $x \in \mathbb{R}$,
2. $p(x, y)^{\mathcal{I}} = \{(x, y)\}$,
3. $l(p_1, p_2)^{\mathcal{I}} = \{ \{ (x, y) \mid \begin{array}{l} (x, y) \in \mathbb{R}^2, \lambda \in [0; 1], \\ (x_1, y_1) \in p_1^{\mathcal{I}}, (x_2, y_2) \in p_2^{\mathcal{I}}, \\ x = x_1 + \lambda(x_2 - x_1), y = y_1 + \lambda(y_2 - y_1) \} \} \},$
4. $kette(l_1, \dots, l_n)^{\mathcal{I}} = \{ \bigcup_{1 \leq i \leq n, l \in l_i^{\mathcal{I}}} l \},$
5. $polygon(l_1, \dots, l_n)^{\mathcal{I}}$ wird interpretiert als der Abschluss des offenen Kerns M° des durch die Jordan-Kurve $\bigcup_{1 \leq i \leq n, l \in l_i^{\mathcal{I}}} l$ definierten Gebietes M (vgl. [Dudenverlag 1994, Definition Gebiet, S. 201]). ■

Proposition 3 Ein geometrisches Substrat ist immer erfüllbar. ■

Beweis: Eine erfüllende Interpretation kann induktiv anhand der Termstruktur wie oben angegeben konstruiert werden. □

Ein geometrisches Substrat repräsentiert bzw. steht daher in gewisser Weise für sich selbst.

Bei Bedarf lassen sich anhand der Reichhaltigkeit eines geometrischen Substrates qualitative räumliche Beschreibungen mittels *Inspektionsmethoden* (s. S. 135) errechnen. Ein *qualitatives geometrisches Substrat* ist eine Spezialisierung eines geometrischen Substrates und eines relationalen Substrates. Hier werden errechnete qualitative Relationen zwischen den Knoten als Kanten eingetragen:

Definition 65 (Qualitatives geometrisches Substrat) Ein *qualitatives geometrisches Substrat* vereinigt in sich die Eigenschaften eines *geometrischen Substrates* und eines *relationalen Substrates*, d.h., $\mathcal{L}_{\mathcal{E}}$ ist nicht länger leer, sondern $\mathcal{L}_{\mathcal{E}} = \{R_1, \dots, R_n\}$ für eine Menge von qualitativen Relationsnamen. ■

¹Dem Autor ist bewusst, dass die reellen Zahlen prädikatenlogisch nicht axiomatisiert werden können. Die Semantik der Geometrie-Beschreibungsterme kann dennoch wie angegeben beschrieben werden.

4.1 Das Substrat-Datenmodell

Ein Modell eines solchen Substrates soll daher sowohl ein Modell eines geometrischen Substrates als auch eines relationalen Substrates sein. An den Semantikdefinitionen muss nichts geändert werden; bei der Definition der logischen Substrate haben wir keine Bedingungen an $\Delta^{\mathcal{I}}$ gestellt (offensichtlich muss $\Delta^{\mathcal{I}}$ wie für das geometrische Substrat definiert sein). Sind die Relationen R_i prädikatenlogisch definierbar, so kann der Prozess des Berechnens dieser Relationen durch Inspektionsmethoden als Model Checking auf einem Modell eines geometrischen Substrates verstanden werden, gefolgt von einer anschließenden Materialisierung dieser Relationen als Kanten.

Ein *qualitatives geometrisches RCC-Substrat* ist eine Spezialisierung eines qualitativen geometrischen Substrates und eines RCC-Substrates:

Definition 66 (Qualitatives geometrisches RCC-Substrat) Ein *qualitatives geometrisches RCC-Substrat* vereinigt in sich die Eigenschaften des *qualitativen geometrischen Substrates* und des *RCC-Substrates*; im Unterschied zu Def. 62 ist $\mathcal{L}_{\mathcal{V}} \neq \emptyset$, da $\mathcal{L}_{\mathcal{V}} = \mathcal{L}_{\mathcal{V}_{\mathcal{R}}}$. ■

Werden die Kanten in E und Beschreibungen in L_E anhand der Geometrie der Knoten-Beschreibungen durch Inspektionsmethoden berechnet, so ist die *relationale Konsistenz* dieses RCC-Substrates automatisch gewährleistet. Die explizit vorhandenen RCC-Kanten können als *Index* zur Beschleunigung räumlicher RCC-Selektionen („Welche Objekte stehen mit Objekt i in der RCC-Relation R ?“) verwendet und verstanden werden. Aufgrund der explizit vorhandenen Kanten müssen dann keine teuren Inspektionsmethoden während der Anfragebeantwortung angewendet werden.

4.1.2.3. Hybride Substrate

Die Anforderungsanalyse in Kap. 3 hat ergeben, dass *hybride Repräsentationen* wünschenswert für die extensionalen Komponenten der ontologiebasierten ISe sind.

Ein *hybrides Substrat* ist wie folgt definiert:

Definition 67 Ein *hybrides Substrat* ist ein Tripel $(S_1, S_2, *)$, wobei $S_1 = (V_1, E_1, L_{V_1}, L_{E_1}, \mathcal{L}_{\mathcal{V}_1}, \mathcal{L}_{\mathcal{E}_1})$ und $S_2 = (V_2, E_2, L_{V_2}, L_{E_2}, \mathcal{L}_{\mathcal{V}_2}, \mathcal{L}_{\mathcal{E}_2})$ Substrate sind. Dabei ist $*$ eine partielle injektive Funktion von V_1 nach V_2 , also $*$: $V_1 \mapsto V_2$. ■

Die Forderung, dass die Knotenmengen disjunkt sind, führt dazu, dass die Interpretationen und somit Erfüllbarkeit von S_1 und S_2 unabhängig voneinander betrachtet werden können. Es findet keine Interaktion statt.

Die injektive, partielle Funktion $* : V_1 \mapsto V_2$ assoziiert einige Knoten in V_1 injektiv mit einigen anderen Knoten in V_2 . Da die Funktion partiell und injektiv (und nicht notwendigerweise surjektiv) ist, ist $*^{-1} : V_2 \mapsto V_1$ ebenfalls partiell und injektiv. Für einen Knoten $i \in V_1$ in S_1 bezeichnen wir $j = *(i)$ bzw. $*i$ als seinen *korrespondieren* oder *assoziierten* Knoten.

Der primäre Zweck eines hybriden Substrates ist daher der eines einfachen „Containers“, der die beiden Repräsentationsschichten bzw. Substrate mittels der $*$ -Funktion „zusammenbindet“.

4.1.3. Inferenzprobleme für Substrate

Ein Substrat S kann als eine prädikatenlogische Formelmenge $\Phi(S)$ angesehen werden. Handelt es sich um ein Substrat mit Hintergrundtheorie $\mathcal{T}$, so gilt $\mathcal{T} \subseteq \Phi(S)$. Unter dieser Perspektive sind die klassischen Inferenzprobleme, wie etwa Erfüllbarkeit, automatisch mit prädikatenlogischer Standard-Semantik definiert.

4.1.4. Substrate für die Leitdomänen

Das Substrat-Datenmodell kann zur Realisierung der extensionalen Komponente für die ISe in den Leitdomänen wie folgt verwendet werden:

- In Kap. 3.1.6 hatten wir verschiedene Repräsentationsmöglichkeiten der DISK-Kartendaten (als prototypisch für Domäne 1) diskutiert. Die dort identifizierte wünschenswerte hybride Repräsentation wollen wir nun durch ein hybrides Substrat bestehend aus einem *geometrischen RCC-Substrat* und einer *ABox* umsetzen. Explizite RCC-Kanten im geometrischen RCC-Substrat können als Index zur Beschleunigung von RCC-Anfragen genutzt werden (s.u.). Die durch die ABox repräsentierten Aspekte wurden bereits in Kap. 3.1.6 diskutierte.

Zur Realisierung der *räumlichen ABox* können wir eine gemeinsame Spezialisierung der Substrat-Typen *ABox* und *geometrisches Substrat* verwenden.

- In Kap. 3.2.4 hatten wir die Möglichkeiten und Defizite von DL-Technologie zur Realisierung von ISen in Domäne 2 analysiert. Auch in dieser Domäne hatten wir festgehalten, dass eine hybride Repräsentation nicht nur aus Performanzgründen wünschenswert ist („Entlastung der ABox für ohnehin literal spezifizierte Datenwerte in OWL“), sondern auch, um erweiterte Anfrageprädikate (analog zu den SPARQL-FILTER-Prädikaten) in einer Anfragesprache anbieten zu können. Wir verwenden ein hybrides Sub-

4.2 Eine generische Anfragesprache für Substrate

strat, welches ein sog. *Daten-Substrat* zur Speicherung (von Teilen des) RDF-Graphen enthält, sowie eine RACERPRO-ABox.

4.2. Eine generische Anfragesprache für Substrate

Um das Substrat-Datenmodell zur Realisierung ontologiebasierter Anfragebeantwortung für die Leitdomänen nutzen zu können, definieren wir nun eine generische logik-basierte Anfragesprache für Substrate. Die Definition der Anfragesprache ist allgemein gehalten, um der Generizität des Datenmodells Rechnung zu tragen. Wir hatten gezeigt, wie für spezielle Repräsentationszwecke Instantiierungen des generischen Substrat-Datenmodells angelegt werden können. Wir können den speziellen Aspekten dieser Instantiierungen mittels spezieller *Anfrage-Atome* gerecht werden. Für ein geometrisches Substrat können wir später z.B. *räumliche Anfrage-Atome* definieren.

4.2.1. Die Basis-Idee

Die Basis-Idee der generischen Anfragesprache beruht auf der Vorstellung des *Graph Matching*. Intuitiv ist eine Anfrage ebenfalls als Substrat-Graph zu begreifen, dessen Knoten durch die *Variablen* der Anfrage gegeben sind, und dessen Knoten- und Kantenbeschriftungen *Einschränkungen (Constraints)* an die möglichen Variablenbindungen darstellen. Das Anfrageergebnis (die Menge der möglichen Antworten) ist dann durch die Menge aller Graph-Morphismen des Anfragegraphen in den Substrat(graphen) gegeben, die die Einschränkungen der Anfrage (also die Knoten- und Kanten-Prädikate) erfüllen. Das Problem kann daher auch als *Einschränkungserfüllungsproblem (Constraint Satisfaction Problem)*, s. z.B. [Kumar 1992], begriffen werden.

Der Substrat-Idee folgend, sehen wir für die Anfragesprache daher zwei Arten von *atomaren Anfragen*, sog. *Anfrage-Atomen* (kurz: *Atome*) vor:

- Knoten-Anfrageatome (auch: Knoten-Anfragen) und
- Kanten-Anfrageatome (auch: Kanten-Anfragen).

Atome referenzieren Variablen und/oder Substrat-Knoten. Diese Atome bilden die Basis-Bausteine, die zu komplexeren Anfragen zusammengesetzt werden können. Wir hatten in Kap. 3 analysiert, dass zur Formulierung der LUBM-, aber auch der DISK-Referenzanfragen mindestens *konjunktive Anfragen* benötigt werden.

Wir wollen zunächst die Anfrage-Atome und ihre Semantik informell erläutern, bevor wir die entsprechenden formalen Definitionen nachreichen.

Ein *Knoten-Anfrageatom* ist ein Paar der Form (a, C) . Hierbei ist a eine Variable oder ein Individuum (dieses referenziert einen Knoten des zu befragenden Substrates), und C ist ein *Knoten-Anfrageprädikat* bzw. die *Knoten-Anfragebedingung*. Das Knoten-Anfrageprädikat ist ein Wort einer *Knoten-Anfragesprache*, die variabel gehalten wird. Dies ist einleuchtend, da die Knoten-Anfragesprache nicht unabhängig von $\mathcal{L}_V$ des befragten Substrates sein wird, und $\mathcal{L}_V$ ebenfalls variabel ist. Wir fordern bzgl. C lediglich, dass C als prädikatenlogische Formel mit einer freien Variablen x angesehen werden kann; mittels $\Phi(C)$ erhalten wir diese Formel. Wir hatten in Kap. 2 die Schreibweise $\Phi(C)[x]$ (zur Notation freier Variablen) eingeführt.

Wie erläutert, kann im Atom (a, C) das Objekt a entweder eine Variable oder ein Individuum sein:

- Ist a eine Variable, so ist ein Substrat-Knoten i nun ein *Treffer* bzw. *erfüllt* ein Knoten-Anfrageatom der Art (a, C) , wenn C eine *logische Folgerung* aus dem befragten Substrat S für den Knoten i ist, also $S \models (i, C)$ bzw. $S \models \Phi(C)_{x \leftarrow i}$ gilt.²

Offensichtlich muss logische Folgerbarkeit für Substrat und Knoten-Anfragesprache definiert sein. Da sowohl ein Substrat S als auch C in prädikatenlogische Äquivalente $\Phi(S)$ und $\Phi(C)$ übersetzt werden können, können wir prinzipiell auch einfach auf die prädikatenlogische $\models$ -Relation verweisen: $\Phi(S) \models \Phi(C)_{x \leftarrow i}$. Im Folgenden unterscheiden wir daher nicht mehr so streng zwischen S und $\Phi(S)$. Sollte das Substrat über eine Hintergrundtheorie $\mathcal{T}$ verfügen, so wollen wir stillschweigend annehmen, dass $\mathcal{T} \subseteq \Phi(S)$ gilt, und daher *ontologiebasierte Anfragebeantwortung* ermöglicht wird, falls $\mathcal{T}$ als Ontologie gedeutet werden kann.

- Ist a hingegen ein Individuum, so muss für einen potentiellen Treffer i auch noch $a = i$ gelten, also $S \models \Phi(C)_{x \leftarrow i} \wedge i = a$.

Hier stellt sich eine wichtige Frage: Wollen wir für den Fall, dass $S \models i = j$ in einem Substrat gilt (dies kann der Fall sein, wenn die UNA nicht gilt) auch j als Treffer von (i, C) ansehen, falls i ein Treffer ist (also $S \models \Phi(C)_{x \leftarrow i}$ gilt)? Prädikatenlogisch muss natürlich $S \models \Phi(C)_{x \leftarrow j}$ gelten.

Wir möchten hier gleich vorwegschicken, dass unsere Entwurfsentscheidung diesbez. „Nein“ lautet. Selbst wenn in allen Modellen eines Substrates $i^{\mathcal{I}} = j^{\mathcal{I}}$ und somit $S \models i = j$ gilt, so wollen wir dennoch i und j als *unterschiedliche Knoten* betrachten, und daher j nicht als Treffer einer Anfrage (i, C) , in der der Nutzer nur nach i gefragt hat, betrachten. Jedes Atom soll eine *extensionale Semantik* haben. Wir möchten, dass die extensionale Semantik der Anfrage

²Da $\Phi(C)$ eine prädikatenlogische Formel mit einer freien Variable x ist, bezeichnet $\Phi(C)_{x \leftarrow i}$ die Formel, die durch Substitution von x in C durch i entsteht; wir hatten bereits erwähnt, dass ein Substrat-Knoten wie i als prädikatenlogische Konstante begriffen werden kann.

4.2 Eine generische Anfragesprache für Substrate

(i, C) durch die Menge $\{i\}$ gegeben ist, und nicht durch $\{i, j\}$.

Die formale Definition erfordert nun jedoch ein Nicht-Standard-Gleichheitsprädikat, welches auf *Namensgleichheit (von Konstanten)* und nicht *semantische Gleichheit* prüft, d.h., „ $=$ “ ist für unsere Zwecke nicht streng genug. Wir notieren das strengere Gleichheitsprädikat durch $\boxed{=}$. Es gilt stets $S \models i \boxed{=} i$ und $S \models \neg(i \boxed{=} j)$, also $S \models (i \boxed{\neq} j)$.

Wir können $\boxed{=}$ prädikatenlogisch definieren, doch ist hierfür zusätzlicher Aufwand notwendig. Hierzu führen wir zusätzlich zur Menge der Konstanten V (für die Knoten des Substrates) eine Menge $\{\boxed{i} \mid i \in V\}$ von „frischen“ Konstanten ein, die wir *Namenskonstanten* nennen.

Die Formelmenge, die durch Übersetzung eines Substrates S mit Knoten V in prädikatenlogische Formeln entsteht, enthalte als Teilmengen nun auch noch die Formelmengen

$$\{ N(\boxed{i}) \wedge \neg N(i) \wedge \text{ref}(\boxed{i}) = i \mid i \in V \}$$

sowie

$$\{ \neg(\boxed{i} = \boxed{j}) \mid i, j \in V, i \neq j \},$$

wobei ref ein Funktionssymbol, und N ein Prädikatssymbol ist. Diese dürfen weder an anderer Position im Substrat (im Bild von L_V oder L_E) noch in der Hintergrundtheorie verwendet werden.

Offensichtlich gilt nun in allen Modellen dieser Formelmenge $\boxed{i}^{\mathcal{I}} \neq \boxed{j}^{\mathcal{I}}$ für Knoten $i, j \in V$ mit $i \neq j$, aber u.U. $i^{\mathcal{I}} = j^{\mathcal{I}}$.

Wir können daher $\boxed{=}[x, y]$ wie folgt definieren:

$$x \boxed{=} y =_{\text{def}} x = y \wedge N(x) \wedge N(y).$$

Enthält ein Atom wie (a, C) nun eine Variable a , so richten wir es so ein, dass diese Variable in $\Phi(C)[x]$ nun nicht gegen eine Konstante $i \in V$ zur Prüfung auf $S \models \Phi(C)_{x \leftarrow i}$ substituiert wird, sondern gegen $\text{ref}(\boxed{i})$. Wir prüfen dann also, ob

$$S \models \Phi(C)_{x \leftarrow \text{ref}(\boxed{i})}$$

gilt. Wir müssen die Anfragesprache also so definieren, dass die freien Variablen in den prädikatenlogischen Φ -Übersetzungen der Anfrageprädikate nicht gegen Konstanten aus V , sondern gegen Namenskonstanten substituiert werden. Im Anfrageergebnis einer Anfrage sollen jedoch nur Elemente aus V erscheinen.

Zudem verwenden wir die sog. *Active Domain Semantic (ADS)* – die prädikatenlogischen Variablen können ausschließlich an *Namenskonstanten* gebunden

werden. Daher muss jede Bindung das Prädikat N erfüllen. Aufgrund der Substitutionssemantik und der Semantik der Folgerungsrelation ist klar, dass eine Bindung nur dann etabliert wird, wenn diese Bindung *in allen Modellen* des befragten Substrates gilt.

Ein *Kanten-Anfrageatom* ist ein Tripel der Art (a, b, R) . Hier sind a, b entweder Variablen und/oder Individuen, und R ist das *Kanten-Anfrageprädikat* bzw. die *Kanten-Anfragebedingung*, welches ein Wort einer *Kanten-Anfragesprache* ist. Wie auch für die Knoten-Anfragebedingung fordern wir, dass R als prädikatenlogische Formel $\Phi(R)$ angesehen werden kann, nun jedoch mit zwei freien Variablen x und y , also $\Phi(R)[x, y]$.

Da das Kanten-Anfrageprädikat zwei freie Variablen hat, müssen nun *Paare von Namens-Konstanten* $(\boxed{i}, \boxed{j})$ betrachtet werden, und $S \models \Phi(R)_{x \leftarrow \text{ref}(\boxed{i}), y \leftarrow \text{ref}(\boxed{j})}$ soll für einen Treffer i, j gelten. Sind a und/oder b Individuen, so werden wiederum entsprechende $\equiv$ -Konjunkte hinzugefügt.

Anfrage-Atome können nun zu *komplexeren Anfragen* zusammengesetzt werden. Beispielsweise ist folgender Ausdruck eine einfache *konjunktive Anfrage*:

$$\text{ans}(x, y) \leftarrow (x, \text{mother}) \wedge (x, y, \text{has\_child}) \wedge (y, \text{male}).$$

Hier sind wir an allen Knoten-Paaren aus $V \times V$ interessiert, die den Rumpf der Anfrage erfüllen. Ist beispielsweise $(\text{betty}, \text{charles})$ ein solches Antworttupel (d.h., es gilt $\text{ans}(\text{betty}, \text{charles})$), dann gilt

$$S \models \text{mother}(\text{ref}(\boxed{\text{betty}})) \wedge \text{has\_child}(\text{ref}(\boxed{\text{betty}}), \text{ref}(\boxed{\text{charles}})) \wedge \text{male}(\text{ref}(\boxed{\text{charles}})).$$

Nach diesen Vorbemerkungen sollte die folgende Darstellung gut nachvollziehbar sein.

4.2.2. Atomare Anfragen

Die (abstrakte) Syntax der Anfrage-Atome ist wie folgt definiert:

Definition 68 (Abstrakte Syntax atomarer Anfragen) Sei

$S = (V, E, L_V, L_E, \mathcal{L}_V, \mathcal{L}_E)$ ein Substrat. Sei $O_S = V \cup \text{Vars}$ die Menge der sog. *Objekte*, die in die Menge der Knoten (V) bzw. *Individuen* und die Menge der Variablen (Vars) partitioniert ist, sodass $V \cap \text{Vars} = \emptyset$ gilt.

Die *atomaren Anfragen* (kurz: *Atome*) sind folgende:

4.2 Eine generische Anfragesprache für Substrate

- Ein *Knoten-Atom* ist ein Paar (a, v) , wobei $a \in O_S$, und $v \in \mathcal{L}_{QV}$. Der Ausdruck v ist das Knoten-Anfrageprädikat, ein Wort aus $\mathcal{L}_{QV}$, der *Knoten-Anfragesprache*.

$\Phi(v)$ ist eine prädikatenlogische Formel mit einer freien Variable x , notiert als $\Phi(v)[x]$.

Wir fordern, dass $\top \in \mathcal{L}_{QV}$ und $\perp \in \mathcal{L}_{QV}$ gilt. Das $\top$ -Prädikat ist stets wahr, und das $\perp$ -Prädikat ist stets falsch.

- Ein *Kanten-Atom* ist ein Tripel (a, b, w) , wobei $a, b \in O_S$, und $w \in \mathcal{L}_{QE}$. Der Ausdruck w ist das Kanten-Anfrageprädikat, ein Wort aus $\mathcal{L}_{QE}$, der *Kanten-Anfragesprache*.

$\Phi(w)$ ist eine prädikatenlogische Formel mit zwei freien Variable x, y , notiert als $\Phi(w)[x, y]$.

- Ein *Gleichheits-Atom* hat die Form $(a, b, \boxed{=})$, mit $a \in Vars$ und $b \in V$.

Dieses Atom wird in erster Linie als Hilfsatom bzw. für Implementierungszwecke verwendet. ■

Zur Illustration der *Semantik* der Atome sei ein aussagenlogisches Substrat S betrachtet, mit $C \wedge (D \vee \neg E) \in L_V(i)$. Also gilt $S \models C(i) \wedge (D(i) \vee \neg E(i))$. Betrachten wir ein Knoten-Atom (x, C) ; also ist $\Phi(C) = C(x)$. Offensichtlich gilt nun $S \models C(x)_{x \leftarrow ref(\boxed{i})}$, d.h., i ist ein Treffer für (x, C) . Für diese „Inferenz“ ist hier lediglich aussagenlogische Deduktion notwendig, es muss also für $L_V(i) = L_V(ref(\boxed{i}))$ geprüft werden, ob $C \wedge (D \vee \neg E) \models C$ gilt, bzw. $C \wedge (D \vee \neg E) \wedge \neg C$ unerfüllbar ist. Die Semantik der atomaren Anfragen ist daher wie folgt gegeben:

Definition 69 (Semantik der atomaren Anfragen) Sei S ein Substrat.

- Ein Knoten $i \in V$ ist ein Treffer eines Gleichheits-Atoms der Form $(a, i, \boxed{=})$ (lt. Definition gilt stets $a \in Vars$).
- Bei den Knoten-Anfragen gibt es zwei Fälle zu betrachten:
 - Ein Substrat-Knoten $i \in V$ ist ein Treffer einer Knoten-Anfrage (a, v) , wenn $S \models \Phi(v)_{x \leftarrow ref(\boxed{i})}$. Ist i ein Treffer, so schreiben wir $S \models (i, v)$, aber auch $S \models (\boxed{i}, v)$.
 - Ein Substrat-Knoten $i \in V$ ist ein Treffer eine Knoten-Anfrage (a, v) mit $a \in Vars$, wenn $S \models \Phi(v)_{x \leftarrow ref(\boxed{i})}$.

Da ein Knotenprädikat $\Phi(v)[x]$ stets die freie prädikatenlogische Variable x verwendet, muss zunächst durch Umbenennung sichergestellt werden, dass x der Variable bzw. dem Objekt entspricht, welches im Atom (a, v)

verwendet wird. Für ein Objekt wie a notieren wir im Folgenden die *repräsentative prädikatenlogische Variablen* mit x_a . Wir erweitern daher die Φ -Funktion auf Atome wie folgt:

$$\Phi((a, v)) =_{def} \begin{cases} \Phi(v)_{x \leftarrow ref(x_a)} & \text{wenn } a \in Vars \\ \Phi(v)_{x \leftarrow ref(x_a)} \wedge \Phi(\boxed{\equiv})_{x \leftarrow x_a, y \leftarrow \boxed{a}} & \text{sonst} \end{cases}$$

Statt $\Phi(\boxed{\equiv})_{x \leftarrow x_a, y \leftarrow \boxed{a}}$ verwenden wir auch eine Infix-Schreibweise: $x_a \boxed{\equiv} \boxed{a}$.

$\Phi((a, v))$ liefert nun also (unabhängig davon, ob $a \in Vars$ oder nicht) eine prädikatenlogische Formel mit freier Variablen x_a , wobei es sich um die korrespondierende prädikatenlogische Variable für Objekt a handelt.

Ein Knoten i ist somit ein Treffer eines Knoten-Atoms (a, v) , wenn $S \models \Phi((a, v))_{x_a \leftarrow \boxed{i}}$ gilt.

- Bei den Kantenabfragen gibt es vier Fälle zu betrachten:
 - Ein Knoten-Paar $(i, j) \in V \times V$ ist ein Treffer einer Kantenabfrage (i, j, w) , wenn $S \models \Phi(w)_{x \leftarrow ref(\boxed{i}), y \leftarrow ref(\boxed{j})}$. Ist (i, j) ein Treffer, so schreiben wir $S \models (i, j, w)$, aber auch $S \models (\boxed{i}, \boxed{j}, w)$.
 - Ein Knoten-Paar $(i, j) \in V \times V$ ist ein Treffer einer Kantenabfrage (i, b, w) mit $b \in Vars$, wenn $S \models \Phi(w)_{x \leftarrow ref(\boxed{i}), y \leftarrow ref(\boxed{j})}$.
 - Ein Knoten-Paar $(i, j) \in V \times V$ ist ein Treffer einer Kantenabfrage (a, j, w) mit $a \in Vars$, wenn $S \models \Phi(w)_{x \leftarrow ref(\boxed{i}), y \leftarrow ref(\boxed{j})}$.
 - Ein Knoten-Paar $(i, j) \in V \times V$ ist ein Treffer einer Kantenabfrage (a, b, w) mit $a, b \in Vars$, wenn $S \models \Phi(w)_{x \leftarrow ref(\boxed{i}), y \leftarrow ref(\boxed{j})}$.

Wir definieren analog wie für die Knoten-Anfragen die Φ -Funktion:

$$\Phi((a, b, w)) =_{def} \begin{cases} \Phi(w)_{x \leftarrow ref(x_a), y \leftarrow ref(x_b)} & \text{wenn } a \in Vars, \\ & b \in Vars \\ \Phi(w)_{x \leftarrow ref(x_a), y \leftarrow ref(x_b)} \wedge x_b \boxed{\equiv} \boxed{b} & \text{wenn } a \in Vars, \\ & b \in V \\ \Phi(w)_{x \leftarrow ref(x_a), y \leftarrow ref(x_b)} \wedge x_a \boxed{\equiv} \boxed{a} & \text{wenn } a \in V, \\ & b \in Vars \\ \Phi(w)_{x \leftarrow ref(x_a), y \leftarrow ref(x_b)} \wedge x_a \boxed{\equiv} \boxed{a} \wedge x_b \boxed{\equiv} \boxed{b} & \text{wenn } a \in V, \\ & b \in V \end{cases}$$

Ein Knoten-Paar $(i, j) \in V \times V$ ist somit ein Treffer einer Kantenabfrage

4.2 Eine generische Anfragesprache für Substrate

(a, b, w) , wenn $S \models \Phi((a, b, w))_{x_a \leftarrow \boxed{i}, x_b \leftarrow \boxed{j}}$ gilt.

- Zudem definieren wir Φ noch für $\boxed{\equiv}[x, y]$ formal wie folgt:

$$\Phi((a, b, \boxed{\equiv})) =_{def} x_a = x_b \wedge N(x_a) \wedge N(x_b).$$

■

Mit Hilfe dieser Definitionen kann die *Extension* einer atomaren Anfragen definiert werden.

Definition 70 (Extension einer atomaren Anfrage) Die Extension einer atomaren Anfrage bzgl. eines Substrates S ist wie folgt definiert:

- $(a, v)^{\mathcal{E}} =_{def} \{ i \in V \mid S \models \Phi((a, v))_{x_a \leftarrow \boxed{i}} \},$
- $(a, b, w)^{\mathcal{E}} =_{def} \{ (i, j) \in V \times V \mid S \models \Phi((a, b, w))_{x_a \leftarrow \boxed{i}, x_b \leftarrow \boxed{j}} \},$
- $(a, b, \boxed{\equiv})^{\mathcal{E}} =_{def} \{ (i, j) \in V \times V \mid S \models \Phi((a, b, \boxed{\equiv}))_{x_a \leftarrow \boxed{i}, x_b \leftarrow \boxed{j}} \}.$

■

Zur Illustration betrachten wir eine Instantiierung der Anfrage-Atome für ABox-Substrate, sog. *ABox-Atome*. Sei $\mathcal{A}$ eine ABox in der die UNA gilt, und $S_{\mathcal{A}}$ das aus ihr abgeleitete Substrat. Ist C ein DL-Konzept, so gilt mit diesen Definitionen offensichtlich

- $\text{concepts\_instances}(C) = (a, C)^{\mathcal{E}},$ und
- $\text{individual\_instance?}(i, C) = \text{TRUE}$ gdw. $S \models (i, C).$

Analoge Äquivalenzen gelten für die Kanten- bzw. „Rollen“-Atome. Ist $R \in \mathcal{N}_{\mathcal{R}}$, so gilt offensichtlich $\text{role\_pairs}(R) = (a, b, R)^{\mathcal{E}},$ etc. Diese ABox-Anfrage-Atome stellen daher Generalisierungen der Basis-ABox-Inferenzprobleme (s. Kap. 2.2.2.5, S. 75) bereit.

4.2.3. Vereinfachte Semantik für einfache Substrate

Für einfache Substrat-Instantiierungen wird für Knoten-Atome (a, v) und $i \in V$ oftmals

$$S \models \Phi((a, v))_{x_a \leftarrow \boxed{i}} \quad \text{gdw.} \quad L_V(i) \models \Phi((a, v))_{x_a \leftarrow \boxed{i}}$$

gelten, und analog für die Kanten-Atome (a, b, w) und $i, j \in V$

$$S \models \Phi((a, b, w))_{x_a \leftarrow \boxed{i}, x_b \leftarrow \boxed{j}} \quad \text{gdw.} \quad L_E((i, j)) \models \Phi((a, b, w))_{x_a \leftarrow \boxed{i}, x_b \leftarrow \boxed{j}}.$$

Für diese einfachen Substrate vereinfacht sich daher die Folgerbarkeitsrelation – zur Entscheidung auf Folgerbarkeit muss nicht mehr das gesamte Substrat S betrachtet werden, sondern lediglich ein Knoten bzw. eine *explizit vorhandene* Kante.

Diese *vereinfachte Semantik* gilt beispielsweise für das aussagenlogische Substrat, nicht jedoch für eine ABox (eine ABox kann insbesondere „implizite Kanten“ enthalten, z.B. wenn sie transitive Rollen-Assertionen enthält). Wir kommen in Kap. 5.2 darauf zurück. Offensichtlich gilt die vereinfachte Semantik auch dann, wenn Model Checking ausreichend ist zur Entscheidung von Folgerbarkeit, d.h., wenn das Substrat als „isomorph“ zu seinem (eindeutigen) minimalem Herbrandmodell angesehen werden kann, s. Kap. 2.

4.2.4. „Negation as Failure“ in der Anfragesprache

Oftmals ist man auch an der *Komplementmenge* der Extensionen $(a, v)^\varepsilon$ und $(a, b, w)^\varepsilon$ interessiert, z.B. an der Menge aller Individuen bzw. Knoten, für die *nicht beweisbar bzw. nicht bekannt ist*, dass sie Instanzen von C sind. Selbst wenn $\mathcal{L}_{QV}$ Negation bietet, wie zum Beispiel im Falle von $\mathcal{ALC}$ -Instanzenanfragen ($\mathcal{L}_{QV} = \mathcal{ALC}$), so wird das Komplement von $(a, C)^\varepsilon$ dennoch nur in Ausnahmefällen mit der Menge der *beweisbar nicht-C-Instanzen* $(a, \neg C)^\varepsilon$ übereinstimmen, da die meisten Substrate (und insbesondere auch ein aus einer ABox abgeleitetes Substrat S_A) keine CWA verwenden. Wir hatten bereits für die DISK-Referenzanfrage **95** (s. S. 147) als auch für eine OWL-Anfragesprache (s. S. 181) motiviert, dass eine solche NAF-Semantik für negierte Anfragen wünschenswert ist.

Die Komplementmengen können wir wie folgt erhalten:

- $V \setminus (a, v)^\varepsilon = \{i \in V \mid S \not\models \Phi((a, v))_{x_a \leftarrow \boxed{i}}\},$
- $(V \times V) \setminus (a, b, w)^\varepsilon = \{(i, j) \in V \times V \mid S \not\models \Phi((a, b, w))_{x_a \leftarrow \boxed{i}, x_b \leftarrow \boxed{j}}\},$
- $(V \times V) \setminus (a, b, \equiv)^\varepsilon = \{(i, j) \in V \times V \mid S \not\models \Phi((a, b, \equiv))_{x_a \leftarrow \boxed{i}, x_b \leftarrow \boxed{j}}\}.$

Diese Anfragen sind immer dann nützlich, wenn die OWA für das zugrundeliegende Substrat verwendet wird. Betrachten wir eine ABox $\{i : C, j : \top\}$. Offensichtlich gilt dann:

- $(x, C)^\varepsilon = \{i\}$ und
- $(x, \neg C)^\varepsilon = \{j\}.$

4.2 Eine generische Anfragesprache für Substrate

Für ein Substrat mit CWA ergäbe sich hingegen $(x, \neg C)^\mathcal{E} = \{j\}$.

Für ein Substrat mit OWA kann die Tatsache, dass für j weder $j \in (x, \neg C)^\mathcal{E}$ noch $j \in (x, C)^\mathcal{E}$ bekannt ist, als Indikator für eine *unvollständige Modellierung* bzw. unvollständige Datenbasis betrachtet werden. Oftmals möchte man genau solche Instanzen durch Anfragen identifizieren. Wir können dazu Anfragen definieren, die die Mengen $V \setminus (x, C)^\mathcal{E}$ und $V \setminus (x, \neg C)^\mathcal{E}$ als Extensionen liefern und bekommen somit eine *positive Bestätigung*, dass unser Wissen bzw. unsere Datenbasis bzgl. j unvollständig ist.

Wir sehen zu diesem Zweck einen *Negation as Failure (NAF)*-Operator in unserer Anfragesprache vor. Der *NAF-Operator* wird durch ein „\“ vor dem entspr. Atom notiert, um ihn von der klassischen Negation „¬“ unterscheiden zu können:

Definition 71 (NAF-Atome) Ist $atom$ eine atomare Anfrage, so ist auch $\backslash atom$ eine atomare Anfrage, ein sog. *Negation as Failure (NAF)-Atom*. ■

Die Semantik des NAF-Operators ist gerade so definiert, dass er zur Konstruktion der obigen Komplementmengen genutzt werden kann:

Definition 72 (Semantik der NAF-Atome) Die Semantik der NAF-Atome ergibt sich wie folgt:

- $(\backslash(a, v))^\mathcal{E} =_{def} \{i \in V \mid S \not\models \Phi((a, v))_{x_a \leftarrow \boxed{i}}\},$
- $(\backslash(a, b, w))^\mathcal{E} =_{def} \{(i, j) \in V \times V \mid S \not\models \Phi((a, b, w))_{x_a \leftarrow \boxed{i}, x_b \leftarrow \boxed{j}}\},$
- $(\backslash(a, b, \boxed{\equiv}))^\mathcal{E} =_{def} \{(i, j) \in V \times V \mid S \not\models \Phi((a, b, \boxed{\equiv}))_{x_a \leftarrow \boxed{i}, x_b \leftarrow \boxed{j}}\} .$ ■

Nun gilt

- $(a, \top)^\mathcal{E} = V,$
- $(a, \perp)^\mathcal{E} = \emptyset,$
- $(a, v)^\mathcal{E} \cup (\backslash(a, v))^\mathcal{E} = V$ mit $(a, v)^\mathcal{E} \cap (\backslash(a, v))^\mathcal{E} = \emptyset,$ und
- $(a, b, w)^\mathcal{E} \cup (\backslash(a, b, w))^\mathcal{E} = V \times V$ mit $(a, b, w)^\mathcal{E} \cap (\backslash(a, b, w))^\mathcal{E} = \emptyset.$

Für ein Atom $atom$ gilt daher stets $(\backslash(\backslash atom))^\mathcal{E} = atom^\mathcal{E}$, etc. Ein Atom und seine NAF-Variante sind daher stets *komplementär* zueinander. Diese wichtige Äquivalenz kann für algebraische Umformungen (z.B. zum Zwecke der Anfrageumschreibung und -optimierung) genutzt werden.

Zur Illustration dieser Äquivalenzen sei ein Substrat S mit $V = \{i, j, k\}$, $L_V(i) = \{C\}$ betrachtet, in dem $S \models i = j$ gilt. Wir setzen $\Phi(C) = C(x)$.

Betrachten wir nun die Anfrage (i, C) , so ist

$$\Phi((i, C)) = \Phi(C)_{x \leftarrow ref(x_i)} \wedge \Phi(\boxed{\equiv})_{x \leftarrow x_i, y \leftarrow \boxed{i}},$$

also

$$\Phi((i, C)) = C(\text{ref}(x_i)) \wedge x_i = \boxed{i} \wedge N(x_i) \wedge N(\boxed{i}).$$

Es gilt daher

$$(i, C)^{\mathcal{E}} = \{l \in V \mid S \models (C(\text{ref}(x_i)) \wedge x_i = \boxed{i} \wedge N(x_i) \wedge N(\boxed{i}))_{x_i \leftarrow l}\} = \{i\},$$

da lediglich $\boxed{i}$ das verschärfte Gleichheitsprädikat erfüllt. Hätten wir dagegen das Standard-Gleichheitsprädikat verwendet, so wäre

$$(i, C)^{\mathcal{E}} = \{l \in V \mid S \models (C(x_i) \wedge x_i = i)_{x_i \leftarrow l}\} = \{i, j\}$$

das Ergebnis gewesen, da auch $S \models i = j$ gilt.

Betrachten wir nun das NAF-Atom $\neg(i, C)$, so gilt lt. unserer Definition

$$(\neg(i, C))^{\mathcal{E}} = \{l \in V \mid S \not\models (C(\text{ref}(x_i)) \wedge x_i = \boxed{i} \wedge N(x_i) \wedge N(\boxed{i}))_{x_i \leftarrow l}\} = \{j, k\},$$

da $S \not\models c_1 \wedge \dots \wedge c_n$ gdw. $S \not\models c_i$ für ein $i \in 1 \dots n$ gilt, und daher zwar

$$\{l \in V \mid S \not\models (C(\text{ref}(x_i)))_{x_i \leftarrow l}\} = \{k\},$$

aber auch

$$\{l \in V \mid S \not\models (x_i = \boxed{i} \wedge N(x_i) \wedge N(\boxed{i}))_{x_i \leftarrow l}\} = \{j, k\}$$

gilt. Unter Verwendung des Standard-Gleichheitsprädikats wäre das Ergebnis

$$(\neg(i, C))^{\mathcal{E}} = \{l \in V \mid S \not\models (C(x_i) \wedge x_i = i)_{x_i \leftarrow l}\} = \{k\},$$

da

$$\{l \in V \mid S \not\models C(x_i)_{x_i \leftarrow l}\} = \{k\}$$

und

$$\{l \in V \mid S \not\models (x_i = i)_{x_i \leftarrow l}\} = \{k\}$$

gilt. In beiden Fällen gilt also die Komplementarität der beiden Atome; das verschärfte Gleichheitsprädikat erscheint uns jedoch deutlich intuitiver, wodurch der „prädikatenlogische Kodierungsaufwand“ gerechtfertigt sei.

4.2.5. Komplexe Anfragen

Komplexe Anfragen gehorchen nun folgender Syntaxdefinition:

Definition 73 (Abstrakte Syntax von Anfragen) Eine *Anfrage* an ein Substrat S hat einen *Kopf* und einen *Rumpf*.

- Ein *Kopf* hat die Form $\text{antwort}(\vec{x})$, wobei $\vec{x}$ ein Vektor von Objekten ist. antwort wird auch das Antwort-Prädikat genannt, und $\vec{x}$ der *Ergebnis-Vektor*.
- Der *Rumpf der Anfrage* ist wie folgt induktiv definiert:
 - Jede atomare Anfrage atom ist ein Rumpf.

4.2 Eine generische Anfragesprache für Substrate

– Wenn $k_1, \dots, k_n$ Rümpfe sind, so sind auch die folgenden Ausdrücke Rümpfe:

- * $\backslash(k_i)$ (NAF-Operator),
- * $(k_1 \wedge \dots \wedge k_n)$ (Konjunktionsoperator),
- * $(k_1 \vee \dots \vee k_n)$ (Disjunktionsoperator), und
- * $\pi(\vec{x}) \ k_1$.

π ist der *Projektionsoperator*; $\vec{x}$ wird der *Kopf*, und k_1 der Rumpf des π -Rumpfes genannt. Hierbei ist $\vec{x}$ ein Vektor von Elementen aus O_S (also Objekten) für die $\text{objekte}(\vec{x}) \subseteq \text{objekte}(k_1)$ (s.u.) gilt, d.h., alle Objekte in $\vec{x}$ müssen im Rumpf k_1 vorkommen.

- Eine *Anfrage* ist eine Formel der Art $(\text{antwort}(\vec{x}) \leftarrow k)$, wobei $\text{antwort}(\vec{x})$ ein Kopf ist, und k ein Rumpf. Zudem muss gelten: $\text{objekte}(\vec{x}) \subseteq \text{objekte}(k)$.
- Eine *disjunktionsfreie Anfrage* ist eine Anfrage, deren Rumpf keine $\vee$ -Operatoren enthält.
- Eine π -freie Anfrage ist eine Anfrage, deren Rumpf keine π -Operatoren enthält.
- Eine *konjunktive Anfrage* (*Conjunctive Query*) ist eine disjunktionsfreie und π -freie Anfrage, deren Rumpf keine NAF-Atome enthält. ■

Die verwendete Funktion objekte ist für Vektoren $\vec{x} = (x_1, \dots, x_n)$ einfach als $\text{objekte}(\vec{x}) =_{\text{def}} \{x_1, \dots, x_n\}$, und für einen Rumpf k induktiv wie folgt definiert:

Definition 74 (Definition von $\text{objekte}(k)$, $\text{vars}(k)$, $\text{inds}(k)$) Sei k ein Rumpf.

Dann definieren wir:

$$\text{objekte}(k) =_{\text{def}} \begin{cases} \text{objekte}(k') & \text{wenn } k = \backslash(k') \\ \{a\} & \text{wenn } k = (a, v) \\ \{a, b\} & \text{wenn } k = (a, b, w) \text{ oder } k = (a, b, \boxed{\Xi}) \\ \bigcup_{1 \leq i \leq n} \text{objekte}(k_i) & \text{wenn } k = (k_1 \wedge \dots \wedge k_n) \text{ oder } k = (k_1 \vee \dots \vee k_n) \\ \text{objekte}(\vec{x}) & \text{wenn } k = \pi(\vec{x}) \ k'. \end{cases}$$

Zusätzlich definieren wir

- $\text{vars}(k) =_{\text{def}} \text{objekte}(k) \cap \text{Vars}$ und
- $\text{inds}(k) =_{\text{def}} \text{objekte}(k) \cap V$. ■

Aufgrund des NAF-Operators und der hiermit verbundenen Nichtmonotonie können wir nun mittels Φ keine rein prädikatenlogische Übersetzung (komplexer) Rümpfe mehr angeben. $\Phi(k)$ wird daher eine *prädikatenlogikähnliche*

Formel sein. Wir werden die zur Definition der Extension bzw. der Semantik von k benötigte Relation $\models_{NAF}$ weiter unten definieren und dabei so viel wie möglich aus der Prädikatenlogik „erben“.

Wir definieren Φ für komplexe Rumpfe wie folgt:

Definition 75 (Definition von Φ für Rumpfe) Sei k ein Rumpf:

- Wenn k ein positives Atom ist, so ist $\Phi(k)$ bereits definiert,
- Wenn $k = \backslash k'$, dann ist $\Phi(k) =_{def} \backslash \Phi(k')$.

Offensichtlich ist $\backslash \Phi(k')$ keine prädikatenlogische Formel im eigentlichen Sinne, da der $\backslash$ -Operator speziell interpretiert werden muss (s.u.). Wir werden den Operator daher bei der Definition der Semantik berücksichtigen müssen.

- Wenn $k = (k_1 \wedge \dots \wedge k_n)$, dann ist $\Phi(k) =_{def} \Phi(k_1) \wedge \dots \wedge \Phi(k_n)$.
- Wenn $k = (k_1 \vee \dots \vee k_n)$, dann ist $\Phi(k) =_{def} \Phi(k_1) \vee \dots \vee \Phi(k_n)$.
- Wenn $k = \pi(\vec{x})k'$, dann ist $\Phi(k) =_{def} \boxed{\exists} \vec{z}.\Phi(k')$.

Auch der $\boxed{\exists}$ -Quantor hat nicht die prädikatenlogische Standardsemantik.

Der Vektor $\vec{z}$ von prädikatenlogischen Variablen wird wie folgt bestimmt: Seien die freien prädikatenlogischen Variablen in $\Phi(k')$ durch den Vektor $\vec{y}$ gegeben.

Sei $\vec{x}'$ der Vektor der durch Ersetzung der Objekte in $\vec{x}$ durch ihre repräsentativen prädikatenlogischen Variablen entsteht. Für $\vec{x} = (a_1, \dots, a_n)$ ist also $\vec{x}' = (x_{a_1}, \dots, x_{a_n})$.

Dann ist $\vec{z}$ definiert als der Vektor von prädikatenlogischen Variablen aus $\vec{y}'$, die nicht im Vektor $\vec{x}'$ auftauchen: $\text{objekte}(\vec{z}) =_{def} \text{objekte}(\vec{y}') \setminus \text{objekte}(\vec{x}')$. Die Reihenfolge der Variablen in den Vektoren ist hier irrelevant (wir können hier einfach eine lexikographische Ordnung annehmen). ■

Zur Illustration betrachten wir den Rumpf $k = (a, C) \wedge \pi(a)(a, b, R)$ mit $a \in \text{Vars}$ und $b \in V$. Dann ist $\Phi(k) = C(\text{ref}(x_a)) \wedge \boxed{\exists} x_b.(R(\text{ref}(x_a), \text{ref}(x_b)) \wedge x_b \boxed{= b})$.

Die Extension einer komplexen Anfrage ergibt sich ganz analog wie im Falle eines Atoms – die freien (prädikatenlogischen) Variablen in q werden gegen Namenskonstanten substituiert. Gilt nach erfolgter Substitution für die entstandene Formel (die nun keine freien Variablen mehr enthält; der Begriff sei hier wie in der Prädikatenlogik gebraucht) $\models_{NAF} q_{x_{a_1} \leftarrow \boxed{v_1}, \dots, x_{a_n} \leftarrow \boxed{v_n}}$, so ist $(v_1, \dots, v_n)$ ein Ergebnistupel:

Definition 76 (Semantik einer Anfrage) Sei $\text{antwort}(\vec{x}) \leftarrow k$ eine Anfrage mit $\vec{x} = (a_1, \dots, a_n)$, und sei $q = \Phi(\pi(\vec{x}) k)$. Dann ist die Extension der Anfrage

4.2 Eine generische Anfragesprache für Substrate

$(antwort(\vec{x}) \leftarrow k)^\varepsilon$ als folgende Tupel-Menge definiert:

$$\{ (v_1, \dots, v_n) \mid v_1, \dots, v_n \in V, S \models_{NAF} q_{x_{a_1} \leftarrow \boxed{v_1}, \dots, x_{a_n} \leftarrow \boxed{v_n}} \}.$$

Analog definieren wir für einen Rumpf k mit $objekte(k) = \{a_1, \dots, a_n\}$ seine Extension k^ε durch

$$\{ (v_1, \dots, v_n) \mid v_1, \dots, v_n \in V, S \models_{NAF} k_{x_{a_1} \leftarrow \boxed{v_1}, \dots, x_{a_n} \leftarrow \boxed{v_n}} \}.$$

Zudem sei $\{()\} =_{def} \text{TRUE}$ und $\{\} =_{def} \text{FALSE}$. ■

Die benötigte $\models_{NAF}$ -Relation ist wie folgt definiert:

Definition 77 (Die „ $\models_{NAF}$ “-Relation) Sei k ein Rumpf, und $k' = \Phi(k)$. Per Induktion über den Formelaufbau von k' definieren wir die Relation $S \models_{NAF} k'$ und somit auch $S \models_{NAF} k$ wie folgt:

- Wenn k ein Atom (aber kein NAF-Atom) ist, dann gilt $S \models_{NAF} k'$ gdw. $S \models k'$ gilt.
- Wenn $k' = \neg k''$, dann gilt $S \models_{NAF} k'$ gdw. $S \not\models_{NAF} k''$ gilt (bzw. wenn *nicht* $S \models_{NAF} k''$ gilt).
- Wenn $k' = (k_1 \wedge \dots \wedge k_n)$, dann gilt $S \models_{NAF} k$ gdw. für alle $i \in \{1, \dots, n\}$ $S \models_{NAF} k_i$ gilt.
- Wenn $k' = (k_1 \vee \dots \vee k_n)$, dann gilt $S \models_{NAF} k$ gdw. es ein $i \in \{1, \dots, n\}$ mit $S \models_{NAF} k_i$ gibt.
- Wenn $k' = \exists x_{a_1}, \dots, x_{a_n}. \Phi(k'')$, dann gilt $S \models_{NAF} k'$ gdw. es eine Substitution der Variablen $x_{a_1}, \dots, x_{a_n}$ gegen Knoten $v_1, \dots, v_n$ aus V gibt, sodass $S \models_{NAF} \Phi(k'')_{x_{a_1} \leftarrow \boxed{v_1}, \dots, x_{a_n} \leftarrow \boxed{v_n}}$ gilt. ■

Prinzipiell lässt diese Definition zu, dass verschiedene Variablen, etwa a, b , an den gleichen Knoten gebunden werden. Oftmals möchte man diese Möglichkeit ausschließen. Zu diesem Zweck können NAF-negierte $\equiv$ -Konjunkte verwendet werden.

Wir wollen hier noch erwähnen, dass wir für Anfragen bzw. Rümpfe ohne NAF-Operator $\models_{NAF}$ einfach gegen $\models$ ersetzen können, und wir somit eine klassische prädikatenlogische Semantik erhalten. Wir müssen dann in Def. 75 Φ für Rümpfe der Form $\pi(\vec{x})k'$ wie folgt redefinieren:

- Wenn $k = \pi(\vec{x})k'$, dann ist $\Phi(k) =_{def} \exists \vec{z}. N(\vec{z}) \wedge \Phi(k')$.

4.2.6. Semantikerhaltende Transformationen

Zur Implementation werden später einige semantikerhaltende Transformationen benötigt, die wir im Folgenden definieren. Die Beweise sind relativ einfach und können per struktureller Induktion über den Aufbau eines Rumpfes geführt werden. Wir verzichten daher auf deren Angabe.

Die *Negationsnormalform (NNF)* einer Anfrage wird erzeugt, indem NAF-negierte komplexe Teil-Rümpfe in einer Anfrage unter Verwendung von *DeMorgans Gesetz* soweit wie möglich in das Innere einer Anfrage gebracht werden. Prinzipiell ist der Projektionsoperator π jedoch *nicht überwindbar*:

Definition 78 (Negationsnormalform einer Anfrage / eines Rumpfes) Sei k ein Rumpf. Dann ist $\text{NNF}(k)$ wie folgt definiert:

$$\text{NNF}(k) =_{\text{def}} \begin{cases} (a, \perp) & \text{wenn } k = \setminus(a, \top) \\ (a, \top) & \text{wenn } k = \setminus(a, \perp) \\ k & \text{wenn } k \text{ atomar ist oder } k = \pi(\vec{y}) \ k' \\ \text{NNF}(\setminus k_1) \vee \dots \vee \text{NNF}(\setminus k_n) & \text{wenn } k = \setminus(k_1 \wedge \dots \wedge k_n) \\ \text{NNF}(\setminus k_1) \wedge \dots \wedge \text{NNF}(\setminus k_n) & \text{wenn } k = \setminus(k_1 \vee \dots \vee k_n) \\ \text{NNF}(k_1) \wedge \dots \wedge \text{NNF}(k_n) & \text{wenn } k = (k_1 \wedge \dots \wedge k_n) \\ \text{NNF}(k_1) \vee \dots \vee \text{NNF}(k_n) & \text{wenn } k = (k_1 \vee \dots \vee k_n) \\ \setminus \text{NNF}(k') & \text{wenn } k = \setminus k'. \end{cases}$$

Für eine Anfrage $q = (\text{antwort}(\vec{x}) \leftarrow k)$ definieren wir

$$\text{NNF}(q) =_{\text{def}} (\text{antwort}(\vec{x}) \leftarrow \text{NNF}(k.))$$

■

Proposition 4 Für alle S, q gilt: $q^{\mathcal{E}} = \text{NNF}(q)^{\mathcal{E}}$. ■

In einem weiteren Schritt können wir die *disjunktive Normalform (DNF)* herstellen:

Definition 79 (Disjunktive Normalform einer Anfrage / eines Rumpfes) Sei k ein Rumpf in NNF. $\text{DNF}(k)$ kann aus k in offensichtlicher Weise erhalten werden, indem in k Vorkommen der Art $k_1 \wedge (k_2 \vee \dots \vee k_n)$ durch $(k_1 \wedge k_2) \vee \dots \vee (k_1 \wedge k_n)$ erschöpfend ersetzt, also „Disjunktionen ausmultipliziert“ werden.

Wir definieren die DNF einer Anfrage $q = (\text{antwort}(\vec{x}) \leftarrow k)$ in NNF entsprechend: $\text{DNF}(q) =_{\text{def}} (\text{antwort}(\vec{x}) \leftarrow \text{DNF}(k))$. ■

4.2 Eine generische Anfragesprache für Substrate

Proposition 5 Für alle S, q gilt: $q^\mathcal{E} = \text{DNF}(q)^\mathcal{E}$. ■

Eine weitere Transformation entfernt die π -Operatoren aus dem Rumpf k einer Anfrage q . Hierzu ersetzen wir π -Rümpfe $\pi(\vec{y}) k'$ in k gegen „Unterabfragen“, die zuvor ausgewertet werden müssen, und deren Ergebnisse dann *referenziert* werden können. Tatsächlich hatten wir bereits in Def. 76 die Extension einer Anfrage durch

$$(\text{antwort}(\vec{x}) \leftarrow k)^\mathcal{E} = (\pi(\vec{x}) k)^\mathcal{E}$$

definiert. Wir ersetzen π -Rümpfe nun gegen Hilfs-Atome, die wir *Referenz-Atome* nennen. Diese Atome sind *nicht* Teil der Sprache, d.h., stehen dem Nutzer nicht zur Verfügung. Wir definieren also für jedes Vorkommen $\pi(\vec{y}) k'$ im Rumpf k eine neue Anfrage mit eindeutigem Namen bzw. Kopf, $(\text{antwort}_{k'}(\vec{y}) \leftarrow k')$, und ersetzen $\pi(\vec{y}) k'$ in k gegen das Referenz-Atom $(\vec{y}, \text{antwort}_{k'})$. Da $(\vec{y}, \text{antwort}_{k'})^\mathcal{E}$ als äquivalent zu $(\text{antwort}_{k'}(\vec{y}) \leftarrow k')^\mathcal{E}$ definiert wird, ist offensichtlich, dass diese Transformation semantikerhaltend ist.

Wir können nun sukzessive alle π -Operatoren in einem Rumpf k von „Innen nach Außen“ durch einen inkrementellen Prozess ersetzen: Wir starten mit einer Menge $\mathcal{M}$, die lediglich $\{q\}$ enthält. In einer Schleife werden nun solche Anfragen aus $\mathcal{M}$ entfernt, die noch einen π -Operator im Rumpf enthalten. Eine solche Anfrage wird aus $\mathcal{M}$ entfernt und wie folgt *gegen zwei neue Anfragen ersetzt*:

Betrachten wir eine Anfrage $q \in \mathcal{M}$ mit $q = (\text{antwort}(\vec{x}) \leftarrow k)$, sodass k einen π -Operator enthält. Wir nehmen an, dass k in DNF ist: $k = (k_{1,1} \wedge \dots \wedge k_{1,n_1}) \vee \dots \vee (k_{m,1} \wedge \dots \wedge k_{m,n_m})$. Sei $k_{i,j}$ das *erste Vorkommen* eines π -Operators, d.h., $k_{i,j} = \pi(\vec{y}) k'$ oder $k_{i,j} = \neg \pi(\vec{y}) k'$ (i, j sind daher minimal). Die Menge $\mathcal{M}$ wird dann wie folgt verändert:

$$\mathcal{M} := \mathcal{M} \setminus \{q\}$$

$$\mathcal{M} := \mathcal{M} \cup \{\text{DNF}(\text{antwort}_{k'}(\vec{y}) \leftarrow k')\}$$

$$\mathcal{M} := \mathcal{M} \cup \{q'\}, \text{ wobei}$$

$$\begin{aligned} q' = (\text{antwort}(\vec{x}) \leftarrow (& (k_{1,1} \wedge \dots \wedge k_{1,n_1}) \vee \\ & \vdots \\ & (k_{i,1} \wedge \dots \wedge k_{i,j-1} \wedge \mathbf{k}'' \wedge k_{i,j+1} \wedge \dots \wedge k_{i,n_i}) \vee \\ & \vdots \\ & (k_{m,1} \wedge \dots \wedge k_{m,n_m}))) \end{aligned}$$

mit $k'' = \neg(\vec{y}, \text{antwort}_{k'})$, falls $k_{i,j} = \neg \pi(\vec{y}) k'$, und $k'' = (\vec{y}, \text{antwort}_{k'})$ andernfalls.

Wir ersetzen also das Konjunkt $k_{i,j}$ gegen das Referenz-Atom $(\vec{y}, \text{antwort}_{k'})$ zur Referenzierung des Anfrageergebnisses der soeben zu $\mathcal{M}$ hinzugefügten

Anfrage $(antwort_{k'}(\vec{y}) \leftarrow k')$.

Die Iteration ist daher wie folgt definiert:

Definition 80 (Klauselmengen $\mathcal{M}(q)$ für eine Anfrage q_0) Die Klauselmengen $\mathcal{M}(q_0)$ für eine Anfrage $q_0 = (antwort_0(\vec{x}) \leftarrow k)$ wird durch folgende Iteration definiert:

Sei $i := 0$, $\mathcal{M}_0 := \{\text{DNF}(q_0)\}$

repeat

$i := i + 1$

Wenn $\mathcal{M}_{i-1}$ noch eine Anfrage mit π -Operator

$q = (antwort(\vec{y}) \leftarrow ((k_{1,1} \wedge \dots \wedge k_{1,n_1}) \vee \dots \vee (k_{m,1} \wedge \dots \wedge k_{m,n_m})))$

enthält, dann sei $k_{i,j}$ das *erste Vorkommen* eines π -Operators, d.h., $k_{i,j} = \pi(\vec{z}) k'$ oder $k_{i,j} = \neg \pi(\vec{z}) k'$ (i, j sind daher minimal). Dann setze:

$\mathcal{M}_i := \mathcal{M}_{i-1} \setminus \{q\}$

$\mathcal{M}_i := \mathcal{M}_i \cup \{\text{DNF}(antwort_i(\vec{z}) \leftarrow k')\}$

$\mathcal{M}_i := \mathcal{M}_i \cup \{q'\}$, wobei

$q' = (antwort(\vec{y}) \leftarrow ((k_{1,1} \wedge \dots \wedge k_{1,n_1}) \vee$

$\vdots$

$(k_{i,1} \wedge \dots \wedge k_{i,j-1} \wedge k'' \wedge k_{i,j+1} \wedge \dots \wedge k_{i,n_i}) \vee$

$\vdots$

$(k_{m,1} \wedge \dots \wedge k_{m,n_m})))$

mit $k'' = \neg(\vec{z}, antwort_i)$, falls $k_{i,j} = \neg \pi(\vec{z}) k'$, und $k'' = (\vec{z}, antwort_i)$ andernfalls.

sonst sei $\mathcal{M}_i := \mathcal{M}_{i-1}$.

until $\mathcal{M}_i = \mathcal{M}_{i-1}$

$\mathcal{M}(q) =_{\text{def}} \mathcal{M}_i$. ■

Am Ende der Iteration steht für die originale Anfrage q mit n π -Operatoren nun eine Menge von n Anfragen $\mathcal{M}(q)$ der Form $\mathcal{M}(q) = \{(antwort_0(\vec{x}) \leftarrow k), (antwort_1(\vec{x}_1) \leftarrow k_1), \dots, (antwort_n(\vec{x}_n) \leftarrow k_n)\}$, wobei alle k_i nun π -frei und in DNF sind. Auf der Menge der Anfragen ist eine Ordnungsrelation definiert, und zwar gilt $q_i < q_j$ für Anfragen q_i, q_j mit $q_i = (antwort_i \dots)$ und $q_j = (antwort_j \dots)$ gdw. $i < j$.

Die Anfrage q mit

$$q = (antwort(x) \leftarrow (x, C) \wedge \neg \pi(x) ((x, D) \wedge (x, y, R)))$$

4.2 Eine generische Anfragesprache für Substrate

führt zu folgender Menge $\mathcal{M}(q)$:

$$\{(antwort_0(x) \leftarrow (x, C) \wedge \neg(x, antwort_1)), (antwort_1(x) \leftarrow (x, D) \wedge (x, y, R))\}.$$

Dies suggeriert eine *Bottom Up-Auswertung* von Anfragen, da das Ergebnis $(x, antwort_1)^\varepsilon$ bekannt sein muss, bevor $\neg(x, antwort_1)$ gebildet werden kann. Somit muss zur Berechnung von $antwort_0(x)^\varepsilon$ zunächst $antwort_1(x)^\varepsilon$ vorliegen. Wir werden daher zunächst $\mathcal{M}(q)$ berechnen, und dann die Anfragen in absteigender Reihenfolge $i \in n \dots 0$ auswerten. Eine Implementation kann die zur Auswertung der Projektionsoperatoren benötigten Teilergebnisse also schrittweise aufeinander aufbauend berechnen.

Zudem kann von der Implementation ausgenutzt werden, dass $\pi(\vec{x}_1) (\pi(\vec{x}_2) (\dots (\pi(\vec{x}_n) k) \dots))$ gegen $\pi(\vec{x}_1 \cap \vec{x}_2 \cap \dots \cap \vec{x}_n) k$ ersetzt werden kann, d.h., π -Operatoren können „so weit als möglich nach innen geschoben werden“. Dies kann (und wird) genutzt werden, um $\mathcal{M}(q)$ zu minimieren, da die Kardinalität von $\mathcal{M}(q)$ durch die Anzahl der π -Operatoren in q bestimmt wird.

Für Implementationszwecke benötigen wir noch einige weitere Äquivalenzen:

Definition 81 (Vereinheitlichung der Stelligkeiten von Disjunkten) Sei k ein disjunktiver Rumpf, sodass $k = k_1 \vee \dots \vee k_n$ ist, und sei $\mathcal{O}_k = \text{objekte}(k)$.

Wir definieren dann:

$$\text{normalisiere\_disjunkte}(k_i) =_{\text{def}} k_i \wedge \bigwedge_{x \in \mathcal{O}_k \setminus \text{objekte}(k_i)} (x, \top),$$

$$\text{normalisiere\_disjunkte}(k_1 \vee \dots \vee k_n) =_{\text{def}} \bigvee_{i \in \{1, \dots, n\}} \text{normalisiere\_disjunkte}(k_i).$$

Für eine Anfrage $q = (antwort(\vec{x}) \leftarrow k)$ mit disjunktivem Rumpf k definieren wir analog:

$$\text{normalisiere\_disjunkte}(q) =_{\text{def}} (antwort(\vec{x}) \leftarrow \text{normalisiere\_disjunkte}(k)).$$

■

In $\text{normalisiere\_disjunkte}(k) = k'_1 \vee \dots \vee k'_n$ haben also alle Disjunkte die gleiche Stelligkeit, sodass für alle $i \in 1 \dots n$ gilt: $\text{objekte}(k'_i) = \text{objekte}(k)$. Gemäß Def. 68 wird stets $\top \in \mathcal{L}_{QV}$ gefordert, sodass $\text{normalisiere\_disjunkte}$ stets definiert ist. So gilt für $k = ((x, C) \vee (x, y, R))$ beispielsweise $\text{normalisiere\_disjunkte}(k) = (((x, C) \wedge (y, \top)) \vee (x, y, R))$.

Proposition 6 Für alle Substrate S und Anfragen in DNF q gilt:

$$q^{\mathcal{E}} = \text{normalisiere\_disjunkte}(q)^{\mathcal{E}}.$$

■

Individuen bzw. Knoten können aus Atomen entfernt werden, wenn diese gegen entspr. Variablen substituiert und entsprechende Gleichheits-Atome als Konjunkte hinzugefügt werden:

Definition 82 (Individuen-Ersetzung) Sei $atom$ ein Atom. Dann definieren wir die individuenfreie Variante von $atom$ wie folgt:

$$\text{ersetze\_individuen}(atom) =_{def} \begin{cases} atom_{\{a \leftarrow x_a \mid a \in \text{inds}(atom')\}} \vee & \text{wenn } atom = \setminus atom', \\ \bigvee_{a \in \text{inds}(atom')} \setminus(x_a, a, \boxed{=}), & \\ atom_{\{a \leftarrow x_a \mid a \in \text{inds}(atom)\}} \wedge & \text{sonst.} \\ \bigwedge_{a \in \text{inds}(atom)} (x_a, a, \boxed{=}) & \end{cases}$$

Wir erweitern die Definition von $\text{ersetze\_individuen}$ auf offensichtliche Weise für beliebige Rümpfe; in Rümpfen der Form $\pi(\vec{x})\ k$ werden sowohl Individuen in $\vec{x}$ als auch in k ersetzt. ■

Offensichtlich gilt $atom^{\mathcal{E}} = \text{ersetze\_individuen}(atom)^{\mathcal{E}}$. Zum Beispiel ist $\text{ersetze\_individuen}((i, C)) = (x_i, C) \wedge (x_i, i, \boxed{=})$, $\text{ersetze\_individuen}(\setminus(i, C)) = \setminus(x_i, C) \vee \setminus(x_i, i, \boxed{=})$, und $\text{ersetze\_individuen}(\pi(i)(x, i, R)) = \pi(x_i)((x, x_i, R) \wedge (x_i, i, \boxed{=}))$.

4.2.7. Deutung der Semantik mit prädikatenlogischen Mitteln

Es kann als ein Mangel der Anfragesprache angesehen werden, dass zur Definition der Semantik der Anfragesprache eine spezielle Folgerungsrelation $\models_{NAF}$ benötigt wird. Dies ist jedoch nicht verwunderlich, da der NAF-Operator die Monotonie-Eigenschaft von $\mathcal{L}_{\mathcal{FOPL}}$ verletzt, vgl. auch Kap. 2.2.9.2.

Eine Alternative wäre, die Anfragesprache mittels *nichtmonotoner Prädikatenlogik-Erweiterungen* zu definieren, s. z.B. [Bibel 1993, Kap.3] und [Görz u. a. 2003, Kap. 7] für eine Übersicht. In der Beschreibungslogik-Forschung wurden auch sog. *autoepistemische Erweiterungen (K-Operatoren)* definiert [Donini u. a. 1992], mit deren Hilfe sich die Semantik ebenfalls (für eine Beschreibungslogik-Instantiierung des Datenmodells) definieren ließe. Wir sind jedoch der Meinung, dass diese Operatoren für unsere Zwecke zu kompliziert sind und die Darstellung daher unnötigerweise erschweren. Stattdessen zeigen wir hier, wie

4.2 Eine generische Anfragesprache für Substrate

die „Bottom Up“-Auswertung beliebiger Anfragen rein prädikatenlogisch verstanden werden kann. Hierzu müssen jedoch neue Grundatome im Rahmen der „Bottom Up“-Auswertung inkrementell zur Formelmenge $\Phi(S)$ hinzugefügt werden.

Wir gehen hierzu wie folgt vor. Für eine beliebige Anfrage q in DNF berechnen wir zunächst $\mathcal{M}(q)$:

$$\mathcal{M}(q) = \{(antwort_0(\vec{x}) \leftarrow k_0), (antwort_1(\vec{x}_1) \leftarrow k_1), \dots, (antwort_n(\vec{x}_n) \leftarrow k_n)\}.$$

Sei $\text{positive\_knoten\_atome}(\mathcal{M}(q))$ nun die Menge der in $\mathcal{M}(q)$ verwendeten nicht-NAF-negierten Knoten-Atome, und $\text{positive\_kanten\_atome}(\mathcal{M}(q))$ die Menge der nicht-NAF-negierten Kanten-Atome. Dann sei anhand von $\mathcal{M}(q)$ folgende Menge $\mathcal{N}$ von Formeln definiert:

$$\begin{aligned} \mathcal{N} =_{def} \{ & \forall x. (N(x) \rightarrow (\boxed{(a, v)}(x) \leftrightarrow \Phi((a, v)))) \mid (a, v) \in \text{positive\_knoten\_atome}(\mathcal{M}(q)) \} \cup \\ & \{ \forall x, y. (N(x) \wedge N(y) \rightarrow (\boxed{(a, b, w)}(x, y) \leftrightarrow \Phi((a, b, w)))) \mid \\ & (a, b, w) \in \text{positive\_kanten\_atome}(\mathcal{M}(q)) \}. \end{aligned}$$

Bei den $\boxed{\dots}$ -Ausdrücken handelt es sich um Prädikatssymbole, die die gleichen Stelligkeiten wie die Φ -Übersetzungen der originalen Atome haben. Wir berechnen nun aus $\Phi(S) \cup \mathcal{N}$ die *maximale Menge der folgerbaren Fakten* der Form

$$\begin{aligned} \mathcal{F}^+ = \{ & \boxed{(a, v)}(\boxed{i}) \mid i \in V \} \cup \\ & \{ \boxed{(a, b, w)}(\boxed{j}, \boxed{k}) \mid j, k \in V \}, \end{aligned}$$

sodass $\mathcal{F}^+$ maximal ist und $\Phi(S) \cup \mathcal{N} \models \mathcal{F}^+$ gilt. Anhand von $\mathcal{F}^+$ definieren wir dann $\mathcal{F}^-$ als die Menge der *nicht folgerbaren Fakten*:

$$\begin{aligned} \mathcal{F}^- = \{ & \boxed{\neg(a, v)}(\boxed{i}) \mid i \in V, \boxed{(a, v)} \notin \mathcal{F}^+ \} \cup \\ & \{ \boxed{\neg(a, b, w)}(\boxed{j}, \boxed{k}) \mid j, k \in V, \boxed{(a, b, w)} \notin \mathcal{F}^+ \}. \end{aligned}$$

Wir definieren nun eine Funktion Φ' , die stets eine prädikatenlogische Formel (im Gegensatz zu Φ) für die Elemente in $\mathcal{M}(q)$ zurückgibt. Die Elemente in $\mathcal{M}(q)$ enthalten keinen π -Operator, und zudem taucht der NAF-Operator hier nur vor Atomen und/oder Referenz-Atomen auf (da die Rümpe in DNF gebracht wurden):

- $\Phi'((a, b, \boxed{\neg})) =_{def} x_a \boxed{\neg} x_b$
- $\Phi'(\neg(a, b, \boxed{\neg})) =_{def} \neg x_a \boxed{\neg} x_b$
- $\Phi'((a, v)) =_{def} \boxed{(a, v)}$

- $\Phi'((a, b, w)) =_{def} \boxed{(a, b, w)}$
- $\Phi'((\vec{x}, antwort_i)) =_{def} \boxed{(\vec{x}, antwort_i)}$
- $\Phi'(\backslash(a, v)) =_{def} \boxed{\backslash(a, v)}$
- $\Phi'(\backslash(a, b, w)) =_{def} \boxed{\backslash(a, b, w)}$
- $\Phi'(\backslash(\vec{x}, antwort_i)) =_{def} \boxed{\backslash(\vec{x}, antwort_i)}$
- $\Phi'(antwort_i(\vec{x}) \leftarrow k_i) =_{def} antwort_i(\vec{x}) \leftarrow \Phi'(k_i)$
- $\Phi'(k_1 \wedge \dots \wedge k_i) =_{def} \Phi'(k_1) \wedge \dots \wedge \Phi'(k_i)$
- $\Phi'(k_1 \vee \dots \vee k_i) =_{def} \Phi'(k_1) \vee \dots \vee \Phi'(k_i)$

Die schrittweise Bottom-Up-Auswertung der Anfragen in $\mathcal{M}(q)$ kann nun prädikatenlogisch wie folgt verstanden werden: Wiederum werden die Anfragen q_i in $\mathcal{M}(q)$ in absteigender Ordnung $i \in n \dots 0$ ausgewertet. Für jede Auswertung q_i wird ausschließlich klassische Folgerbarkeit benötigt. Nach jeder Auswertung $q_i = antwort_i((a_1, \dots, a_n) \leftarrow k_i)$ wird die berechnete Extension als Menge positiver $\boxed{antwort_i}$ - und negativer $\boxed{\backslash antwort_i}$ -Fakten explizit gemacht und daher für nachfolgende Auswertungen berücksichtigt.

Zunächst können wir lediglich $q_n = (antwort_n(a_1, \dots, a_m) \leftarrow k_n)$ auswerten, was uns zu den Faktenmengen $\mathcal{F}_n^+$ und $\mathcal{F}_n^-$ führt:

$$\mathcal{F}_n^+ = \{ \boxed{antwort_n}(\boxed{v_1}, \dots, \boxed{v_m}) \mid v_1, \dots, v_m \in V, S \cup \mathcal{F}^+ \cup \mathcal{F}^- \models \Phi'(q_n)_{x_{a_1} \leftarrow \boxed{v_1}, \dots, x_{a_m} \leftarrow \boxed{v_m}} \}$$

Schließlich können wir q_i auswerten, wenn alle Mengen $\mathcal{F}_j^+$ und $\mathcal{F}_j^-$ für $j \in (1+i) \dots n$ vorliegen:

$$\mathcal{F}_i^+ = \{ \boxed{antwort_i}(\boxed{v_1}, \dots, \boxed{v_m}) \mid v_1, \dots, v_m \in V, (S \cup \bigcup_{j \in (1+i) \dots n} \mathcal{F}_j^+ \cup \mathcal{F}_j^-) \models \Phi'(q_i)_{x_{a_1} \leftarrow \boxed{v_1}, \dots, x_{a_m} \leftarrow \boxed{v_m}} \}$$

$$\mathcal{F}_i^- = \{ \boxed{\backslash antwort_i}(\boxed{v_1}, \dots, \boxed{v_m}) \mid v_1, \dots, v_m \in V, \boxed{antwort_i}(\boxed{v_1}, \dots, \boxed{v_m}) \notin \mathcal{F}_i^+ \}.$$

Am Ende enthält $\mathcal{F}_0^+$ die gewünschten Ergebnis-Tupel.

4.2.8. Eine Logikprogrammierungs-Perspektive

Die Substrat-Anfragesprache kann letztlich auch als eine Art von *nichtrekursivem Datalog mit Negation* verstanden werden [Abiteboul u. a. 1995], [Kemper u. Eickler 2004, Kap. 15]. Der NAF-Operator entspricht dann der Datalog-Negation. Die Menge $\mathcal{M}(q)$ kann als Menge von Datalog-Regeln betrachtet werden. Diese enthalten jedoch u.U. noch den Disjunktionsoperator – es ist jedoch wohlbekannt, dass Vereinigung bzw. Disjunktion in Datalog-Regelrümpfen nicht erforderlich ist, da beispielsweise die Datalog-Regel

$$q(x) \leftarrow r(x) \vee s(x)$$

gegen die äquivalente Regelmenge

$$\begin{aligned} q(x) &\leftarrow r(x) \\ q(x) &\leftarrow s(x) \end{aligned}$$

ersetzt werden kann, s. [Abiteboul u. a. 1995, Kap. 4.5].

Unter dieser Datalog-Betrachtungsweise müssen jedoch die Extensionen aller Anfrage-Atome als sog. Basis- bzw. EDB-Relationen vorliegen, bzw. die Faktenbasis $\mathcal{F}^+$ muss zuvor materialisiert worden sein. Die Datalog-Perspektive ist daher *pragmatisch irreführend* [Wessel u. Möller 2006], da wir die EDB-Relationen erst bei Bedarf während der Anfragebearbeitung (inkrementell) berechnen wollen. Zudem sind in Datalog nur *Prädikatsnamen* in den Anfrage-Atomen möglich. Die Sichtweise, dass die EDB-Relationen anhand *komplexer Anfrageprädikate* vorab materialisiert werden, ist daher auch aus diesem Grund nicht wirklich adäquat (wenn auch prinzipiell möglich, vgl. auch den Abschnitt über *Constraint Query Languages* in [Abiteboul u. a. 1995, Kap. 5.6]). Zudem sind Substrat-Typen vorhanden, die nicht unmittelbar als einfache Faktenmengen interpretiert werden können (z.B. das geometrische Substrat).

Der Autor ist daher der Meinung, dass das Substrat-Rahmenwerk mehr Flexibilität als ein entspr. Datalog-Rahmenwerk bietet. Wir belegen diese These im weiteren Verlauf der Arbeit durch die Vorstellung von Instantiierungen bzw. Ausprägungen des Rahmenwerks; der skeptische Leser mag sich jedoch stets fragen: „Wie sähe dies in Datalog aus?“

4.2.9. Oft verwendete Anfrage-Idiome

Oftmals soll lediglich das Vorhandensein oder Nicht-Vorhandensein eines R -Nachfolgers oder -Vorgängers eines Knotens überprüft werden. Da diese Rümpfe in vielen Anfragen benötigt werden, definieren wir syntaktische Abkürzungen. Seien $a, b \in O_S$ und $R \in \mathcal{L}_{QV}$:

- $(a, \text{has\_known\_successor}(R)) =_{def} \pi(a)(a, b, R)$
- $(a, \text{has\_known\_predecessor}(R)) =_{def} \pi(a)(b, a, R)$

- $(a, \perp, R) =_{def} \setminus(\pi(a)(a, b, R)) = \setminus(a, has\_known\_successor(R))$
- $(\perp, a, R) =_{def} \setminus(\pi(a)(b, a, R)) = \setminus(a, has\_known\_predecessor(R))$

Soll hingegen überprüft werden, ob ein R -Nachfolger existiert bzw. nicht existiert, der das Knoten-Prädikat C erfüllt, so müssen die folgenden komplexen Rümpfe verwendet werden:

- $\pi(a)((a, b, R) \wedge (b, C))$
- $\setminus(\pi(a)((a, b, R) \wedge (b, C)))$

Man beachte, dass $i \in (\setminus(\pi(a)((a, b, R) \wedge (b, C))))^\varepsilon$ für $a, b \in Vars$ genau dann gilt, wenn i *keinen bekannten R -Nachfolger hat, der C erfüllt*. Dies ist jedoch genau dann der Fall, wenn *alle bekannten R -Nachfolger j das Atom $\setminus(j, C)$ erfüllen*, also $j \in (\setminus(b, C))^\varepsilon$ gilt. Die bedeutet aufgrund der OWA (wie bereits erwähnt) nicht notwendigerweise, dass (sofern $\mathcal{L}_{QV}$ Negation bietet) auch $j \in (b, \neg C)^\varepsilon$ gilt.

Wir hatten bereits in Kap. 3 für die DISK-Referenzanfrage **Q5** festgehalten (s. S. 147), dass ein *Allquantor mit CWA-Semantik* oftmals wünschenswert ist. Dieses Idiom ist insbesondere auch für eine ABox- und eine SEMANTIC WEB-Anfragesprache sehr nützlich, da wir dann zwischen Instanzen, die *beweisbar keine C -erfüllenden R -Nachfolger* unter Verwendung von $(a, \forall R. \neg C)$ und solchen Instanzen, die *lediglich keine bekannten R -Nachfolger, die C erfüllen* mittels $\setminus(\pi(a)((a, b, R) \wedge (b, C)))$ unterscheiden können.

4.2.10. Eine alternative algebraische Semantik

Wir geben nun noch eine alternative Semantik für die Anfragesprache an, die unserer Meinung nach noch einfacher zu verstehen ist als die logikbasierte Semantik. Aufgrund der Äquivalenz von nichtrekursivem Datalog ohne Negation mit relationaler Algebra [Kemper u. Eickler 2004, Kap. 15.9] ist es nicht verwunderlich, dass wir eine algebraische Semantik angeben können.

Die algebraische Semantik basiert auf der Idee, dass die Extension k^ε eines Rumpfes k direkt durch Anwendung von Mengenoperationen anhand der Extensionen der Teilrumpfe des Rumpfes konstruiert werden kann. Die Operationen $\wedge, \vee, \setminus$ entsprechen den booleschen Mengenoperationen $\cap, \cup, \setminus$ auf n -stelligen Tupelmengen.

Da diese Mengenoperationen nur dann definiert sind, wenn sie auf Argumentmengen gleicher Stelligkeit angewendet werden, müssen die Extensionen aller Teilrumpfe eines Rumpfes gleiche Stelligkeit n haben. Zur Reduzierung der Dimensionalität einer n -stelligen Tupelmenge wird die *Projektions-Operation* verwendet; die „inverse Operation“ nennen wir hier *Zylindrisierungs-Operation*. Die Operationen sind wie folgt definiert:

4.2 Eine generische Anfragesprache für Substrate

Definition 83 (Projektions-Operation) Sei $\mathcal{T}$ eine Menge von n -stelligen Tupeln der Form $(t_1, \dots, t_n)$, und $(i_1, \dots, i_m)$ ein *Index-Vektor* mit $1 \leq i_j \leq n$ für alle $1 \leq j \leq m$. Wir bezeichnen dann die Menge $\mathcal{T}'$ von m -stelligen Tupeln

$$\mathcal{T}' =_{\text{def}} \{ (t_{i_1}, \dots, t_{i_m}) \mid (t_1, \dots, t_n) \in \mathcal{T} \}$$

als *Projektion* von $\mathcal{T}$ auf die Tupel-Komponenten, die im Index-Vektor $(i_1, \dots, i_m)$ genannt werden. Die Menge wird mit $\pi_{(i_1, \dots, i_m)}(\mathcal{T})$ bezeichnet.

Die π -Operation sollte nicht mit dem π -Operator verwechselt werden. ■

Zum Beispiel ist $\pi_{(1,3)}\{(1, 2, 3), (2, 3, 4)\} = \{(1, 3), (2, 4)\}$.

Definition 84 (Zylindrisierungs-Operation) Sei $\vec{b} = (b_1, \dots, b_n)$ ein *Bit-Vektor* der Länge n , also $b_i \in \{0, 1\}$. Sei $m \leq n$, und $\vec{b}$ ein Bit-Vektor, der genau m Einsen enthält, und $\mathcal{B}$ eine Grundmenge sowie $\mathcal{T}$ eine Menge von m -stelligen Tupeln. Dann ist die n -dimensionale *Zylindrisierung* von $\mathcal{T}$ bzgl. $\mathcal{B}$ und $\vec{b}$ als

$$\mathcal{T}' =_{\text{def}} \{ (i_1, \dots, i_n) \mid (j_1, \dots, j_m) \in \mathcal{T}, 1 \leq l \leq m, 1 \leq k \leq n \\ \text{mit } i_k = j_l \text{ wenn } b_k = 1, \text{ und } b_k \text{ ist die } l. \text{ Eins in } \vec{b}, \\ \text{andernfalls, } i_k \in \mathcal{B} \}$$

definiert, und wird mit $\chi_{\mathcal{B}, (b_1, \dots, b_n)}(\mathcal{T})$ bezeichnet. ■

Zum Beispiel ist $\chi_{\{a,b\}, (0,1,0,1)}(\{(x, y)\}) = \{(a, x, a, y), (a, x, b, y), (b, x, a, y), (b, x, b, y)\}$.

Wir benötigen auch die n -dimensionale Diagonale und einen speziellen 1-Vektor:

Definition 85 (Diagonale, 1-Vektor) Der n -dimensionale Bit-Vektor, der Einsen an den durch die Index-Menge $\mathcal{I} \subseteq \{1, \dots, n\}$ spezifizierten Positionen hat, wird mit $\vec{1}_{n, \mathcal{I}}$ bezeichnet.

Die n -dimensionalen Identitätsrelation (Diagonale) auf der Grundmenge $\mathcal{B}$ ist definiert als

$$\mathcal{ID}_{n, \mathcal{B}} =_{\text{def}} \{ (\underbrace{x, \dots, x}_n) \mid x \in \mathcal{B} \}.$$

■

Zum Beispiel ist $\vec{1}_{4, \{1,3\}} = (1, 0, 1, 0)$.

Unter Verwendung dieser Operationen kann nun die algebraische Semantik wie folgt angegeben werden:

Definition 86 (Algebraische Semantik der Anfragesprache) Sei $\mathcal{S}$ ein Substrat, und $q = (\text{antwort}(\vec{x}) \leftarrow k_0)$ eine Anfrage.

Zunächst ersetzen wir die Individuen in q gegen repräsentative Variablen, s. Def. 82. Es gilt $(antwort(\vec{x}) \leftarrow k_0)^\mathcal{E} = \text{ersetze\_individuen}(\pi(\vec{x}) k_0)^\mathcal{E}$.

In einem weiteren Schritt betrachten wir die π -Teilrumpfe von k_0 . Sei $k' = \pi(x_1, \dots, x_n)k''$ ein solcher Teilrumpf von k_0 . Wir benennen nun die Variablen in k'' um, die *nicht* in der Projektionsliste $(x_1, \dots, x_n)$ auftauchen. Dies sind gemäß der Definition von *objekte*, Def 74, genau die Variablen in k'' , die nicht in *objekte*(k') auftauchen. Nach erfolgter Umbenennung wird k' in k_0 gegen den so modifizierten Teilrumpf ersetzt (wir sparen uns eine formale Definition dieser Variablenumbenennungsoperation). Dies geschehe für alle π -Rümpfe. Den so erhaltenen Rumpf nennen wir k .

Zum Beispiel wird aus dem Rumpf $(x, C) \wedge \pi(x)((x, D) \wedge (x, y, R))$ der Rumpf $(x, C) \wedge \pi(x)((x, D) \wedge (x, y_{123}, R))$, wobei y_{123} ein frischer, noch nicht verwendeter Variablenname ist.

Sei $\text{vars}(k) = (x_{1,k}, \dots, x_{n,k})$ eine lexikografische Aufzählung der Variablen in k . Das i . Element in diesem Vektor bezeichnen wir mit $x_{i,k}$.

Die Extensionen der Anfrage-Atome sind nun als folgende n -stellige Mengen definiert (man beachte, dass $n = \text{vars}(k)$):

$$\begin{aligned} (x_{i,k}, v)^\mathcal{E} &=_{\text{def}} \chi_{V, \vec{\mathbf{I}}_{n, \{x\}}}(\{s \in V \mid S \models (s, v)\}), \\ (x_{i,k}, x_{j,k}, w)^\mathcal{E} &=_{\text{def}} \chi_{V, \vec{\mathbf{I}}_{n, \{i, j\}}}(\{(s, t) \in V \times V \mid S \models (s, t, w)\}), \\ (x_{i,k}, x_{i,k}, w)^\mathcal{E} &=_{\text{def}} \chi_{V, \vec{\mathbf{I}}_{n, \{i, j\}}}(\{(s, t) \in V \times V \mid S \models (s, t, w)\}) \cap \mathcal{ID}_{2,V}, \\ (x_{i,k}, j, \boxed{=})^\mathcal{E} &=_{\text{def}} \chi_{V, \vec{\mathbf{I}}_{n, \{i\}}}(\{j\}). \end{aligned}$$

Da nun die Extensionen aller Atome bzw. Teilrumpfe einer Anfrage die gleiche Stelligkeit haben, kann die Standard-Semantik der Mengenoperationen zur Definition der Semantik genutzt werden:

$$\begin{aligned} (\bigwedge_{1 \leq j \leq i} k_j)^\mathcal{E} &=_{\text{def}} \bigcap_{1 \leq j \leq i} k_j^\mathcal{E} \\ (\bigvee_{1 \leq j \leq i} k_j)^\mathcal{E} &=_{\text{def}} \bigcup_{1 \leq j \leq i} k_j^\mathcal{E} \\ (\setminus k)^\mathcal{E} &=_{\text{def}} V^n \setminus k^\mathcal{E} \\ (\pi(x_{i_1,k}, \dots, x_{i_k,k})k)^\mathcal{E} &=_{\text{def}} \pi_{(i_1, \dots, i_k)} k^\mathcal{E} \end{aligned}$$

■

Diese Semantik erscheint aufgrund ihres konstruktiven Charakters als besonders implementierungsfreundlich bzw. -nahe.

4.2.11. Inferenzprobleme für Anfragen

Die wichtigsten Inferenzprobleme für Anfragen sind Prüfung auf Erfüllbarkeit sowie Subsumptionsbestimmung zwischen Anfragen. Es geht uns hier um eine generelle Definition dieser Probleme für Substrat-Anfragen; die Entscheidbarkeit bzw. Komplexität dieser Probleme muss natürlich für *spezielle Instantiierungen* sowohl des Substrat-Datenmodells als auch der Substrat-Anfragesprache dediziert untersucht werden.

Unsere Definition der *Erfüllbarkeit einer Anfrage* ist folgende:

Definition 87 (Anfrage-Erfüllbarkeit) Eine Anfrage q heißt erfüllbar, wenn es ein *erfüllbares* (also konsistentes) Substrat S gibt, in dem $q^\varepsilon \neq \emptyset$ gilt. ■

Ist eine Hintergrundtheorie $\mathcal{T}$ mit S assoziiert, so ist stillschweigend natürlich Erfüllbarkeit von $(S, \mathcal{T})$ gemeint.

Definition 88 (Unerfüllbare und tautologische Anfrage) Eine *unerfüllbare Anfrage* q liefert stets (auf allen Substraten S) die leere Menge: $q^\varepsilon = \emptyset$. Eine *tautologische Anfrage* q liefert stets $q^\varepsilon = V^n$, wenn $n = |\text{objekte}(q)|$ (bzw. stets TRUE wenn $n = 0$). ■

Da diese Begriffe letztlich anhand der Extensionen von Rümpfen definiert sind, verwenden wir diese Begriffe analog auch für Rümpfe.

Anfrage-Subsumption (*Query Subsumption* / *Query Entailment* / *Query Containment*) [Abiteboul u. a. 1995, Kap. 6.2] wird in unserem Rahmenwerk primär als *semantische Optimierungsmethode* Anwendung finden.

Unsere Definition von Anfrage-Subsumption ist:

Definition 89 (Anfrage-Subsumption und Anfrage-Äquivalenz) Eine Anfrage q_2 *subsumiert* eine Anfrage q_1 , in Zeichen $q_1 \sqsubseteq q_2$, wenn für q_1, q_2 auf allen erfüllbaren Substraten S gilt: $q_1^\varepsilon \subseteq q_2^\varepsilon$.

Die Anfrage q_2 wird *subsumierende*, und q_1 *subsumierte Anfrage* genannt.

Zwei Anfragen q_1, q_2 heißen *äquivalent*, in Zeichen $q_1 \equiv q_2$, wenn sie sich wechselseitig subsumieren, bzw. $q_1^\varepsilon = q_2^\varepsilon$ für alle erfüllbaren Substrate S gilt.

Unabhängig von $|\text{objekte}(q)|$ gelte für eine unerfüllbare Anfrage q stets $q \equiv \perp$, und für eine tautologische Anfrage stets $q \equiv \top$. ■

Diese Begriffe seien nicht nur für Anfragen, sondern auch für Rümpfe entspr. definiert.

Die Anfragesubsumptions-Beziehung hat die gleichen Eigenschaften wie die Subsumptions-Beziehung zwischen Konzepten; die sog. *QBox* ist ein GAG und

wird anhand von $\sqsubseteq$ zwischen Anfragen exakt wie die Taxonomie im Falle einer TBox berechnet:

Definition 90 (QBox) Eine QBox ist ein GAG (gerichteter azyklischer Graph) (V, E) , wobei V eine Menge von Anfragen ist, $E \subseteq V \times V$ eine Menge von Kanten, und $(q_2, q_1) \in E$ gilt genau dann, wenn $q_2 \sqsubseteq q_1$ und aus $\exists q_3 \in V. q_2 \sqsubseteq q_3, q_3 \sqsubseteq q_1$ stets entweder $q_3 \equiv q_1$ oder $q_3 \equiv q_2$ folgt.

Ein QBox enthält stets eine $\top$ - und eine $\perp$ -Anfrage. Für beliebige Anfragen $q \in V$ gilt: $q \sqsubseteq \top, \perp \sqsubseteq q$.

Zudem seien die Operatoren *parents* und *children* wie folgt definiert: $\text{parents}(q) =_{def} \{p \mid (q, p) \in E\}$, und $\text{children}(q) =_{def} \{p \mid (p, q) \in E\}$. ■

Offensichtlich wird zur Verwaltung bzw. Berechnung der QBox ein Algorithmus zur Entscheidung von Anfrage-Subsumption benötigt. Es ist seit langem bekannt, dass eine QBox zur Beschleunigung der Berechnung von Anfrageergebnissen genutzt werden kann [Buchheit u. a. 1994a; 1993b]:

- Die gespeicherten Ergebnismengen von *subsumierende Anfragen* können zur Beantwortung einer Anfrage q als *Obermengen-Caches* von *Kandidatengeneratoren* genutzt werden.
- Die gespeicherten Ergebnismengen von *subsumierten Anfragen* können sofort als Teilmengen des Ergebnisses von q zurückgeliefert werden.

Damit die Anfrageergebnisse subsumierender Anfragen als Kandidatengeneratoren genutzt werden können, werden tatsächlich nicht die Extensionen der Anfragen, sondern die Extensionen ihrer Teilrumpfe benötigt, denn für das Anfrageergebnis wurden ja evtl. Variablenbindungen durch Projektionen entfernt. Für eine Anfrage $q = (\text{antwort}(\vec{x}) \leftarrow k)$ wird daher die Menge $\bigcap_{p \in \text{parents}(k)} p^\mathcal{E}$ eine Obermenge, und die Menge $\bigcup_{c \in \text{children}(k)} c^\mathcal{E}$ eine Untermenge von $k^\mathcal{E}$ sein. Somit müssen zur Berechnung von $q^\mathcal{E}$ lediglich die verbleibenden Tupel in

$$\bigcap_{p \in \text{parents}(q)} p^\mathcal{E} \setminus \bigcup_{c \in \text{children}(q)} c^\mathcal{E}$$

auf Mitgliedschaft in $k^\mathcal{E}$ getestet werden. Damit dies möglich ist, wird Anfrage-subsumption so verschärft, dass $k_2 \sqsubseteq k_1$ nur dann gilt, wenn k_1 und k_2 Variablenvektoren gleicher Stelligkeit haben, also $|\text{vars}(k_2)| = |\text{vars}(k_1)|$ gilt. Enthält k_2 mehr Variablen als k_1 , so können für einige Variablen in k_2 keine Bindungen anhand von $k_1^\mathcal{E}$ etabliert werden. Tatsächlich wäre der Fall $|\text{vars}(k_1)| > |\text{vars}(k_2)|$ sogar unkritisch; wir fordern aus Gründen der Einfachheit hier dennoch Gleichheit. Für einen neuen Rumpf bzw. eine Anfrage q werden daher ihre speziellsten subsumierenden und die allgemeinsten subsumierten Anfragen in der QBox

4.2 Eine generische Anfragesprache für Substrate

berechnet. Dieser Prozess wird *Anfrage-Klassifikation* genannt. Nach erfolgter Anfragebeantwortung kann q ebenfalls in die QBox eingetragen werden.

Als Seiteneffekt der Klassifikation können Anfragen auch als äquivalent zu bereits in der QBox vorhandenen Anfragen erkannt werden, was natürlich der Optimalfall ist. Zudem können Anfragen als unerfüllbar oder tautologisch erkannt werden (bzw. als äquivalent zur $\perp$ - bzw. $\top$ -Anfrage).

4.2.12. Hybride Anfragen

Wir hatten in Kap. 4.1.4 zwei hybride Substrat-Instantiierungen zur Nutzung als extensionale Komponenten für ISe in den Leitdomänen beschrieben. Wir hatten auch motiviert, dass dann eine hybride Anfragesprache benötigt wird.

Ein *hybrides Substrat* hatten wir als ein Tripel $(S_1, S_2, *)$, bestehend aus zwei Substraten S_1 und S_2 mit injektiver partieller Knoten-Assoziationsfunktion $* : V_1 \mapsto V_2$ definiert. Also sind einige Knoten in V_1 mit anderen Knoten in V_2 über die Funktion $*$ assoziiert.

Die Substrate S_1 und S_2 sind Instanzen spezieller Substrat-Instantiierungen; für beide sind zudem spezialisierte Instantiierungen der Anfragesprache definiert. Es existieren also für S_1 und S_2 „passende“ Knoten- und Kanten-Anfrageatome, sodass für $i \in \{1, 2\}$ sowohl $\mathcal{L}_{QV_i}$ für $\mathcal{L}_{V_i}$ als auch $\mathcal{L}_{QE_i}$ für $\mathcal{L}_{E_i}$ definiert sind.

Für die GIS-Domäne hatten wir ein hybrides Substrat bestehend aus einem *geometrischen RCC-Substrat* (S_1) und einer *ABox* (S_2) als nützlich identifiziert. Wir bezeichnen dieses Substrat im Folgenden als *GIS-Substrat*. Wie könnte nun eine hybride Anfragesprache für das GIS-Substrat aussehen? Eine Anfragesprache für ein *geometrisches RCC-Substrat* (S_1) wird räumliche Anfrage-Atome bereitstellen, darunter auch RCC-Kanten-Atome. Aufgrund der explizit vorhandenen Geometrie können die räumlichen Atome Inspektionsmethoden zur Anfragebeantwortung nutzen. Für eine *ABox* (S_2) wird offensichtlich eine ABox-Anfragesprache benötigt.

Die Idee der hybriden Anfragesprache für $(S_1, S_2, *)$ ist nun, die beiden Anfragesprachen S_1 und S_2 einfach zu vereinigen. Diese Vereinigungssprache lässt daher alle Atome aus $\mathcal{L}_{QV_1} \cup \mathcal{L}_{QV_2}$ und $\mathcal{L}_{QE_1} \cup \mathcal{L}_{QE_2}$ zu. Die komplexen Anfragen haben ja stets die gleiche Struktur (lediglich die Atome sind variabel).

Die hybride Anfragesprache für das GIS-Substrat wird daher sowohl ABox-Atome als auch räumliche Atome bieten. Mittels der räumlichen Atome können die räumlichen Aspekte, und mittels der ABox-Atome die thematischen Aspekte der DISK-Referenzanfragen umgesetzt werden.

Für ein gegebenes Atom einer hybriden Anfrage müssen wir entscheiden

können, ob es sich um ein Atom für S_1 oder für S_2 handelt. Obwohl in vielen Fällen $\mathcal{L}_{QV_1} \cap \mathcal{L}_{QV_2} = \emptyset$ und $\mathcal{L}_{QE_1} \cap \mathcal{L}_{QE_2} = \emptyset$ gelten wird, muss dies nicht notwendigerweise der Fall sein. Als pragmatische Lösung fordern wir hier deshalb einfach, dass Objekte für S_1 (Variablen $Vars_1$ und Individuen V_1) von solchen für S_2 (Variablen $Vars_2$ und Individuen V_2) unterschieden werden können. Wir versehen daher Variablen in $Vars_2$ einfach mit einem *-Präfix.

Für eine Variable $x \in Vars_1$ wird nun $*x \in Vars_2$ die sog. *korrespondierende Variable* genannt, und wird fordern, dass Paare $(x, *x)$ von korrespondierenden Variablen **-konsistent* gebunden werden müssen:

Definition 91 (*-konsistente Variablenbindung) Sei $x \in Vars_1$ eine Variable, und $*x \in Vars_2$ ihre korrespondierende Variable.

Eine Variablenbindung, die x an $i \in V_1$ und $*x$ an $j \in V_2$ bindet, heißt **-konsistent*, wenn $*(i) = j$ gilt. ■

Offensichtlich könnte *-Konsistenz durch entsprechende (substratübergreifende) $\equiv$ -Konjunkte in der Anfrage sichergestellt werden.

Referenziert ein Atom nun ein Individuum in $V_1 \cap V_2$, beispielsweise (i, C) mit $i \in V_1 \cap V_2$, so ist hier eine prinzipielle Mehrdeutigkeit vorhanden. Wir wollen hier nicht $V_1 \cap V_2 = \emptyset$ fordern, denn sonst könnten nicht beliebige Substrate zu einem hybriden Substrat kombiniert werden. Zur Disambiguierung referenzieren wir $i \in V_1$ in der Anfragesprache daher durch i_{V_1} , und $i \in V_2$ durch i_{V_2} . Wie auch für die Variablen wird die *-Präfix-Schreibweise verwendet: Gilt also $*(i_{V_1}) = i_{V_2}$, so ist mit i stets i_{V_1} , und mit $*i$ stets i_{V_2} gemeint.

4.2.13. Anfragesprachen für die Leitdomänen

Wir werden die Instantiierungen der Anfragesprache für die Leitdomänen erst im nächsten Kapitel formal definieren. Wir werden dort vier konkrete Anfragesprachen und ihre softwaretechnische Realisierung darstellen. Wir halten es für sinnvoll, auch die formale Darstellung dieser Anfragesprachen zu verzögern, da diese Instantiierungen für konkretere Substrat-Spezialisierungen als die hier dargestellten definiert sind, und wir diese erst im nächsten Kapitel vorstellen.

4.3. Zusammenfassung

Wir haben das durch die in Kap. 3 durchgeführte Anforderungsanalyse motivierte semistrukturierte graphbasierte Substrat-Datenmodell nun formal definiert, und auch einen formalen Rahmen für Anfragesprachen für dieses Datenmodell bereitgestellt. Das Modell bietet Abstraktionen sowohl für die intensionalen, die extensionalen, als auch die Anfragekomponenten ontologiebasierter ISe. Aufgrund der Generizität können auch DL-Systeme (oder einfach nur ABoxen) in diesem Rahmen gedeutet werden.

Die Flexibilität des Substrat-Datenmodells wird durch auswechselbare Knoten- und Kanten-Beschreibungssprachen erreicht. Die Substrat-Anfragesprache trägt dieser Generizität ebenfalls Rechnung, indem für spezielle Instantiierungen dedizierte Knoten- und Kanten-Atome definiert werden können.

Im nächsten Kapitel wird die softwaretechnische Umsetzung des formalen Rahmens in ein Rahmenwerk (Framework) dargestellt. Aufgrund des formalen Rahmens sind die vom Rahmenwerk zu erwartenden Dienste und Leistungen klar definiert. Wir stellen im nächsten Kapitel auch Ausprägungen des Rahmenwerks vor, welche als prototypische ontologiebasierte ISe in den Leitdomänen angesehen werden können. Diese Prototypen werden ontologiebasierte Anfragebeantwortung ermöglichen und sind somit als *Proof of Concept*-Instantiierungen des formalen und dann auch softwaretechnischen Rahmens zu begreifen.

5. Softwaretechnische Realisierung des formalen Rahmenwerks

Inhaltsangabe

5.1	Realisierung des Substrat-Datenmodells	235
5.1.1	Formale vs. softwaretechnische Anforderungen	235
5.1.2	Objektorientierter Entwurf	236
5.1.3	Substrate für die Leitdomänen	269
5.1.4	Objektorientierte Implementation	274
5.2	Realisierung der Substrat-Anfragebeantwortungsmaschine	278
5.2.1	Softwaretechnische Anforderungen	278
5.2.2	Objektorientierter Entwurf	279
5.2.3	Objektorientierte Implementation	313
5.3	Vier konkrete Substrat-Anfragesprachen	318
5.3.1	Eine Anfragesprache für das Daten-Substrat – Die DSA	319
5.3.2	Eine generische ABox-Anfragesprache – Die BAA	322
5.3.3	(Leitdomäne 2) Der SWIR-Dienst und NRQL	334
5.3.4	(Leitdomäne 1) Das DLMAPS-System und die DISKQL	380
5.4	Zusammenfassung	398

In diesem Kapitel beschreiben wir die softwaretechnische Realisierung des in Kap. 4 aufgespannten Substrat-Datenmodells und der für dieses Datenmodell definierten (Klasse von) Anfragesprachen. Ziel des hier vorzunehmenden Software-Entwurfs soll die Erstellung eines wiederverwendbaren *Software-Rahmenwerks* des formalen Rahmens sein. Wir verwenden die Definition aus [Gamma u. a. 1996, S. 31] zur Charakterisierung eines Rahmenwerks (*Hinzufügungen des Autors in Kursivschrift*):

Eine Rahmenwerk besteht aus einer Menge von zusammenarbeitenden (*softwaretechnischen*) Klassen, die einen wiederverwendbaren Entwurf für eine bestimmte Klasse von Software darstellen.

Die *Klasse von Software* ist in unserem Falle durch *Software zur Erstellung (prototypischer) ontologiebasierter Informationssysteme in COMMON LISP zur Realisierung ontologiebasierter Anfragebeantwortung unter Verwendung des Substrat-Datenmodells und der Substrat-Anfragesprache(n)* gegeben.

Für die in Kap. 3 diskutierten Leitdomänen werden wir zwei prototypische

„Proof of Concept“-Implementierungen von ISen angeben, welche das Rahmenwerk nutzen. Diese prototypischen ISe sind daher Instanzen bzw. Spezialisierungen des Rahmenwerks.

- Als Instanz in der GIS- bzw. DISK-Domäne werden wir das DLMAPS-System entwerfen, wobei es sich um ein prototypisches ontologiebasiertes GIS für die DISK handelt (DLMAPS steht für *Description Logic Maps*). Das DLMAPS-System nutzt für die extensionale Komponente das Substrat-Datenmodell. Die Anfragesprache für das DLMAPS-System nennen wir DISKQL; es handelt es sich um eine Instantiierung bzw. Spezialisierung der generischen (hybriden) Substrat-Anfragesprache.
- Die SEMANTIC WEB-Domäne bzw. der SWIR-Dienst auf RACERPRO-Basis benötigt eine ausdrucksstarke ABox-Anfragesprache – die *new RacerPro Query Language* (NRQL) bietet die in Kap. 3.2.4 identifizierte benötigte Funktionalität (Inkrementalität, Ressourcenkonfigurierbarkeit, erweiterte Filter-Prädikate, usw.). NRQL nutzt für die extensionale Komponente das Substrat-Datenmodell. Wie auch DISKQL ist NRQL eine Instantiierung bzw. Spezialisierung der generischen Substrat-Anfragesprache.

Das prototypische SWIR-Szenario eines Universitäts-ISs im SEMANTIC WEB auf LUBM-Basis (s. Kap. 3.2.4.1) wurde von Studierenden der Technischen Universität Hamburg-Harburg (Matthias Flege, Thomas Hirsch, Torben Rebhan, und Kamil Sokolski) im Rahmen eines Projektseminars realisiert. Die Studierenden haben auch eine endnutzergerechte Weboberfläche für diesen SWIR-Dienst geschaffen – den sog. OWL TREE.

Es sei erneut betont, dass es in dieser Arbeit nicht primär um diese beiden Instantiierungen des Rahmenwerks geht. Der wichtigere Forschungsbeitrag ist in den im Rahmenwerk verkörperten Abstraktionen zu sehen. Die softwaretechnische Realisierung des Rahmenwerks wurde durch den formalen Rahmen bzw. Entwurf in Kap. 4 gut vorbereitet, sodass wir ein klares Bild bzgl. der benötigten Klassen und Funktionalitäten vor Augen haben. Allerdings werden sowohl die initialen Analysen aus Kap. 3 als auch die formalen Entwürfe aus Kap. 4 verfeinert werden müssen. Wir werden die durch die Generizität des formalen Rahmens erzielte Flexibilität durch Techniken aus der objektorientierten Programmierung (OOP), wie Nutzung von Vererbung und später bzw. dynamischer Bindung, realisieren.

Die Kommunikation komplexer Software-Entwürfe ist ein schwieriges Unterfangen. Dies gilt umso mehr, wenn die betrachtete Domäne inhärent komplex und die Software umfangreich ist. Beides ist hier der Fall. Wir sind daher gezwungen, uns bei der Darstellung auf die wesentlichsten Aspekte der Software zu beschränken. Insbesondere widmen wir der Darstellung des *Entwurfs und der zugrundeliegenden Entwurfsprinzipien* wesentlich mehr Aufmerksamkeit als der Darstellung der Implementierung. Wir hatten betont, dass der primäre

Beitrag dieser Arbeit in den im Rahmenwerk materialisierten Abstraktionen besteht.

Zur Darstellung bzw. Kommunikation von Entwurfsideen verwenden wir hin und wieder die sog. *Entwurfsmuster* [Gamma u. a. 1996], da durch sie eine allgemeinbekannte und weitgehend akzeptierte Kommunikationsbasis „zum Reden über Entwürfe“ gegeben ist. Aus gleichem Grund verwenden wir UML [OMG 2004, Fowler 2004]. UML-Klassendiagramme nutzen wir zur Darstellung bzw. Kommunikation der statischen Aspekte unseres Entwurfs. Zustandsdiagramme verwenden wir gelegentlich, um dynamische/operationale Aspekte zu kommunizieren. Wir verwenden jedoch *keine Sequenz-Diagramme*, da mit ihnen nur ausgewählte Beispiel-Abläufe dargestellt werden können. Stattdessen bevorzugen wir hier eine rein sprachliche Darstellung, da diese mehr Flexibilität und Möglichkeiten zur Generalisierung von Aussagen über Abläufe bietet.

Prinzipiell sind wir der Meinung, dass die softwaretechnische Realisierung des Rahmenwerks aufgrund des in Kap. 3 aufgespannten formalen Rahmens gut vorbereitet ist. Natürlich muss der formale Rahmen nun verfeinert werden. Dabei müssen wir die rein formale Perspektive auf das Datenmodell und die Anfragesprachen verlassen, und einen softwaretechnischen Blickwinkel einnehmen. Dies führt auch zu einer *verfeinerten Analyse*, die insbesondere auch *softwaretechnische (operationale) Anforderungen* explizit machen muss. Unter *Realisierung* verstehen wir im Folgenden die integrierte Darstellung von *verfeinerter Analyse, Entwurf und Implementierung*.

Teilweise ist auch die *Trennung von Entwurf und Implementierung nicht immer strikt einzuhalten bzw. wünschenswert* („Implementierung ist Entwurf im Feinen“). Dies liegt u.a. auch an den schon im Entwurf zu berücksichtigenden *Bottom Up-Einflüssen*, also an Basis-Entwurfsentscheidungen wie verwendete Programmiersprache, verwendetes DL-System, etc. Ein guter Entwurf wird diese Bottom Up-Einflüsse berücksichtigen, um innere und somit letztlich auch äußere Qualitätsfaktoren der Software sicherzustellen. Speziell im Fall der hier verwendeten Programmiersprache COMMON LISP sind viele „gute“ Entwurfsentscheidungen (sog. „Best Practices“) wohlbekannt und sollten daher berücksichtigt werden [Norvig 1992, Graham 1996; 1994].

Die Implementationen werden in COMMON LISP [Steele 1990] und ihrer dynamischen objektorientierten Erweiterung CLOS [Kiczales u. a. 1991, Keene 1989] vorgenommen. Die Wahl von COMMON LISP ist durch folgende Sprachmerkmale motiviert: COMMON LISP bietet Mehrfachvererbung, Multimethoden (welche mehr als ein Argument zur Laufzeit-Auswahl der Methode einer generischen Funktion verwenden), automatische Speicherverwaltung, bietet komplexe Symbolverarbeitung und ist ideal zur Realisierung eingebetteter domänenspezifischer Sprachen mittels Makros. *Funktionen* sind in COMMON LISP Datenobjekte erster Ordnung; sie erlauben insbesondere auch den Kon-

trollfluss datengesteuert und daher flexibel zu gestalten, z.B. durch Verwendung der Programmier Technik der *Fortsetzungen (Downward Continuations)*. COMMON LISP bietet damit die für diese Arbeit erforderliche Flexibilität und Erweiterbarkeit:

Lisp is a programmable programming language (John Foderaro, Franz Inc.)

Wir beschreiben zunächst die softwaretechnische Realisierung des Substrat-Datenmodells. Dann folgt die Darstellung der Realisierung der *Anfragebeantwortungsmaschine (Query Answering Engine)* für die Substrat-Anfragesprache(n). Schließlich beschreiben wir *vier konkrete Ausprägungen* des Rahmenwerks und ihre Realisierung. Hierzu gehören auch NRQL und DISKQL, und wir stellen die OWL TREE- und die DLMAPS-Anwendungen als zwei prototypische ontologiebasierte ISe vor, die das Rahmenwerk bzw. NRQL und DISKQL nutzen. Am Ende des Kapitels folgt eine Zusammenfassung. Wir möchten darauf hinweisen, dass wir im Folgenden mit „Nutzer“ entweder einen menschlichen Nutzer oder eine Client-Anwendung meinen (die das API nutzt).

5.1. Realisierung des Substrat-Datenmodells

Unter Realisierung verstehen wir im Folgenden die integrierte Darstellung der verfeinerten softwaretechnischen Analyse, des Entwurfs, und der Implementierung.

Wir beschreiben nun zunächst die verfeinerten softwaretechnischen Anforderungen an ein Substrat-Repräsentationsrahmenwerk. Es folgt dann der objektorientierte Entwurf. In diesem Entwurf konkretisieren wir auch die in Kap. 4 skizzierten spezialisierten Substrat-Arten bzw. -Typen als softwaretechnische Klassen, die die extensionale Repräsentationsbasis des Rahmenwerks darstellen. Einige dieser Instantiierungen bzw. Spezialisierungen des Substrat-Datenmodells sind mehr oder weniger anwendungsübergreifend und somit als Standard- bzw. Mehrzweck-Substrate begreif- und nutzbar, während andere Substratklassen speziell für die extensionalen Komponenten der Beispiel-ISE in den Leitdomänen entworfen werden. Schließlich widmen wir der Darstellung der Implementation einige Zeilen.

5.1.1. Formale vs. softwaretechnische Anforderungen

Als softwaretechnische Anforderungen bezeichnen wir im Folgenden alle Anforderungen an die zu entwerfende Software, welche nicht bereits bei der Darstellung des formalen Substrat-Datenmodells (explizit oder implizit) formuliert worden sind. Während das formale Substrat-Datenmodell primär als *konzeptuelles oder logisches Datenmodell* angesehen werden muss, so muss eine Implementierung eines solchen Datenmodells natürlich auch andere Aspekte berücksichtigen, z.B. Aspekte der *physischen Repräsentation bzw. Darstellung* (welche Speicherstrukturen werden zur Repräsentation verwendet?), die aus rein konzeptueller Perspektive irrelevant sind. Offensichtlich muss ein *geometrisches Substrat* ganz andere Speicherstrukturen als beispielsweise ein *RCC-Substrat* bereitstellen, und effiziente Anfragebeantwortung mit der Substrate-Anfragesprache kann nur realisiert werden, wenn *substratspezifische Indexstrukturen* bereitgestellt werden. Derartige Aspekte gilt es im folgenden Entwurf zu berücksichtigen; sie vervollständigen und komplementieren die formalen Aspekte des Datenmodells. Das Substrat-Datenmodell ist daher sowohl ein *konzeptuelles* als auch ein *physisches Datenmodell*.

Zudem ist durch das formale Substrat-Datenmodell auch die Perspektive gegeben, dass letztlich *überhaupt keine lokale Speicherrepräsentation vorhanden sein muss*, sondern dass es sich lediglich um eine auf Anfrage durch eine Funktion bzw. Methode berechnete *virtuelle Sicht* auf eine andere Repräsentation handelt, beispielsweise auf eine ABox eines *externen (evtl. entfernten)* DL-Systems; in diesem Fall ist das API durch eine Menge entfernter Prozeduren bzw.

5.1 Realisierung des Substrat-Datenmodells

Methoden gegeben. Wir nennen ein solches Substrat *virtuell*. Ein virtuelles Substrat stellt somit lediglich eine *Substrat-Perspektive* bereit und fungiert als eine Art *Vermittler (Mediator)*. Das Substrat nimmt dann die Aufgaben einer *semantischen Zwischenschicht (Semantic Middleware)* wahr, indem es softwaretechnische Abstraktionen zum einheitlichen Zugriff auf die Repräsentations- und Inferenzdienste eines externen DL-Systems bietet. Hierdurch kann lose Kopplung mit verschiedenen DL-Systemen realisiert werden.

Unter einem *externen DL-System* verstehen wir ein DL-System, dessen Repräsentations- und Inferenzdienste vom Rahmenwerk ausschließlich über seine externe dokumentierte Schnittstelle (das API) in Anspruch genommen werden. Das API eines *entfernten DL-Systems* ist über eine Netzwerkschnittstelle erreichbar. Unter Verwendung eines *internen DL-Systems* (wie MiDeLoRA) können Client- und DL-System-Code hingegen gemeinsame Software- und Indexstrukturen nutzen (Offenheit und Erweiterbarkeit auf Software-Ebene).

Sowohl die konzeptuellen als auch die physischen Aspekte des Substrat-Datenmodells können wir durch Nutzung der durch die OOP bereitgestellten Flexibilität realisieren. Insbesondere wird Vererbung eine wichtige Rolle spielen.

5.1.2. Objektorientierter Entwurf

Die Basis-Idee des Entwurfs ist, die konzeptuelle Perspektive auf das Substrat-Datenmodell („Substrat als beschrifteter Graph“) durch eine abstrakte (nicht instantiierbare) Basis-Klasse Substrat (Substrate) zu realisieren, welche eine Schnittstelle (Menge von Methoden bzw. Methoden-Signaturen) zur Inanspruchnahme der Repräsentations- und Inferenzdienste des Substrates bietet. Einen Klassenbezeichner notieren wir im Folgenden mittels Klassenname (Class Name), da in den UML-Diagrammen englischsprachige Bezeichner verwendet werden (Class Name). Die Schnittstelle stellt Methoden zur Konstruktion von Knoten und Kanten, Methoden zum Auffinden und Löschen von Knoten und Kanten, Methoden zur Inanspruchnahme von Inferenzdiensten usw. zur Verfügung. Wir konkretisieren diese Schnittstelle weiter unten (s. Abb. 5.2). Es handelt sich bei der Substrat (Substrate)-Klasse nicht um eine reine Schnittstelle (Interface), da hier bereits generisches Standard-Verhalten implementiert ist.

Von dieser Basis-Klasse können dann *spezialisierte, anwendungsspezifische Unterklassen* abgeleitet werden, die instantiierbar (also konkret) sind. In den Implementationen dieser spezialisierten Substratklassen können dann die *physischen und softwaretechnischen Aspekte* der Instantiierung des Substrat-Datenmodells adäquat berücksichtigt und realisiert werden. Die abstrakte Substrat-Schnittstelle wird dann substratspezifisch ausimplementiert, und kann auch erweitert werden, sodass *substratspezifische Zusatzdienste* geboten

werden können. Zur Implementation der Dienste können dann beispielsweise substratspezifisch bereitgestellte Indexstrukturen genutzt werden.

Während wir ein Substrat bisher also in gewisser Weise als einen *abstrakten Datentypen (ADT)* begriffen haben (die konzeptuelle Perspektive), so begreifen wir nun im Rahmen des objektorientierten Entwurfs diesen ADT als eine Wurzel-Klasse Substrat (Substrate) für ein formal fundiertes, softwaretechnisches Rahmenwerk. Ein Substrat ist im Folgenden daher stets eine Instanz einer Unterklasse der Klasse Substrat (Substrate). Unter *einer Instanz einer Klasse* verstehen wir im Folgenden stets *eine Instanz der Klasse oder eine Instanz einer ihrer Unterklassen*; andere Autoren verwenden die Begriffe *Instanz* und *Element* zur Unterscheidung.

Der objektorientierte Entwurf als Klasse ermöglicht die *Instantiierung des Substrat-Datenmodells* durch Spezialisierung von Substratklassen. Bereits in Kap. 4 wurden bestimmte Instantiierungen des Substrat-Datenmodells als Spezialisierungen beschrieben, und auch Mehrfachvererbung wurde hierfür bereits sprachlich genutzt. Beispielsweise hatten wir ein qualitatives geometrisches RCC-Substrat als ein qualitatives geometrisches Substrat *und* ein RCC-Substrat definiert. Da CLOS Mehrfachvererbung bietet, können wir dies direkt softwaretechnisch umsetzen.

Der Entwurf wird zunächst eher als eine *Klassenbibliothek* [Gamma u. a. 1996, S. 30] denn als ein Rahmenwerk wahrgenommen werden. Ein Rahmenwerk unterscheidet sich von einer Klassenbibliothek darin, dass im Rahmenwerk (für die betrachtete Klasse von Software relevantes) *herausfaktorisierendes Standard-Verhalten* durch sog. *Schablonenmethoden* in Form von *Standard-Implementierungen* bereitgestellt wird. Eine solche Schablonenmethode ruft zur Dienstleistung ihrerseits wieder Methoden auf, welche vom Nutzer überschrieben werden können, wodurch das Verhalten der Standardimplementierung unterklassenspezifisch verfeinert, erweitert, angepasst oder parametrisiert werden kann.

Schablonenmethoden setzen das sog. *Hollywood-Prinzip* („Don't call us, we call you!“) um, welches oft zur *Charakterisierung* von Rahmenwerken verwendet wird [Gamma u. a. 1996, S. 31] (*Hinzufügungen des Autors in Kursivschrift*):

Die Wiederverwendung... führt zu einer Umkehrung der Steuerung zwischen der Anwendung und der Software, auf der sie basiert. ... Wenn Sie ein Framework verwenden, ... schreiben (*Sie*) den Code, der vom Framework gerufen wird. . .

Wir werden insbesondere in der Substrat-Anfragebeantwortungsmaschine umfangreiche Standard-Implementierungen bereitstellen.

Ein Rahmenwerk muss stets auch ein gewisses Mindestmaß an implementierter Grund- bzw. Basis-Funktionalität anbieten („muss ein Angebot bereitstel-

5.1 Realisierung des Substrat-Datenmodells

len“). In unserem Falle ist dieses Angebot durch eine Reihe konkreter Standardklassen gegeben, die unmittelbar für Anwendungen genutzt werden können. Diese werden hier ebenfalls entworfen.

5.1.2.1. Die Basis-Klassen und -Methoden des Entwurfs

Die Basis des Entwurfs ist durch eine Menge abstrakter Klassen (es handelt sich nicht um reine Schnittstellen) gegeben, die die Wurzeln der Klassenhierarchie bilden. Wie in Kap. 4 dargestellt, ist ein Substrat ein knoten- und kantenbeschrifteter gerichteter Graph. Einem konsequenten objektorientierten Entwurf folgend, repräsentieren wir diesen Graphen als *Objektgraphen* (also als Graphen von CLOS-Instanzen). Wir sehen daher die abstrakten Basis-Klassen Substrat (Substrate), Knoten (Node) und Kante (Edge) vor. Nicht nur die Bildung von Substrat-Unterklassen ist (wie bereits erläutert) sinnvoll, sondern auch die Bildung spezialisierter Knoten- und Kanten-Klassen; beispielsweise wird im geometrischen Substrat ein solcher Knoten auch eine Instanz eines räumlichen Datentyps (beispielsweise eines Polygons) sein.

Wir weichen hier in einem wichtigen Detail von dem konzeptuellen Substrat-Datenmodell ab: Wir erlauben hier auch mehr als eine Kante zwischen einem Paar von Knoten. Im formalen Modell müsste die Kanten-Menge E dann zu einer Multimenge (einem „Bag“) werden. Semantisch ist dieser Unterschied irrelevant, da wir eine Multimenge von Kanten $\{(i, j), (i, j)\}$ mit unterschiedlichen Labeln $\{R, S\}$ stets als eine einzelne Kante mit konjunktivem Label „ $R \wedge S$ “ interpretieren können (sofern $\mathcal{L}_E$ Konjunktionen bietet).

Wenn das Substrat lediglich virtuell ist, muss die Graphstruktur nicht notwendigerweise als Objektgraph repräsentiert werden. Stattdessen kann der Graph beispielsweise lediglich relational in einer Datenbank gespeichert sein, oder aber durch die ABox eines (externen und evtl. sogar entfernten) DL-Systems gegeben sein. Der Graph wird somit *extern* repräsentiert. Wir müssen daher die Schnittstellen-Methoden so gestalten, dass es möglichst immer ausreichend ist, für ihre Argumente *Namen bzw. IDs für Objekte* statt (*Referenzen auf*) *Objekte selbst* zu übergeben. Um beispielsweise einen Knoten aus einem Substrat zu entfernen, soll die `delete-node`-Methode als Argument sowohl eine Referenz auf den zu löschenden Knoten als auch den Namen bzw. eine ID für diesen als mögliche Argumente akzeptieren.

Andere Methoden benötigen hingegen notwendigerweise CLOS-Instanzen als Argumente bzw. geben notwendigerweise CLOS-Instanzen zurück. In diesem Falle können *lokale Proxy-Objekte* der entfernten Substrat-Objekte erzeugt werden. Diese können entweder dynamisch, also bei Bedarf auf Anforderung von den Methoden erzeugt und zurückgegeben werden. Ist die Objektidentität dieser Objekte nicht relevant bzw. muss im Laufe der Zeit nicht erhalten blei-

ben, so können speicherplatzsparende Muster wie das *Fliegengewicht-Muster* und das *Objekt-Pool-Muster*¹ verwendet werden. Alternativ kann auch bereits beim Anlegen des Substrates eine vollständige *lokale Spiegel-Repräsentation* des entfernten Substrat-Graphen als Objektgraph (in einer Initialisierungsphase) erzeugt werden. Dieser Spiegel muss dann jedoch mit der entfernten Repräsentation konsistent gehalten werden.

Da verschiedene Substrat-Spezialisierung auch unterschiedliche Knoten- und Kanten-Beschriftungssprachen $\mathcal{L}_V$ und $\mathcal{L}_E$ verwenden, nehmen wir ebenfalls einen objektorientierten Entwurf von *Beschreibungsklassen* (auch: *Deskriptorklassen*) vor. Ein Wort aus $\mathcal{L}_V$ bzw. $\mathcal{L}_E$ wird durch eine Instanz einer solchen Beschreibungsklasse dargestellt; wir nennen diese Instanz auch *Beschreibungsinstanz*. Die Wurzel der Hierarchie der Beschreibungsklassen ist durch die abstrakte Klasse Beschreibung (Description) gegeben. Prinzipiell unterscheiden wir zwischen unären und binären Beschreibungen (Klassen Unäre Beschreibung (Unary-Description) und Binäre Beschreibung (Binary-Description)). Knoten-Beschreibungen sind spezielle unäre Beschreibungen, und Kanten-Beschreibungen sind spezielle binäre Beschreibungen (Klassen Knoten-Beschreibung (Node-Description) und Kanten-Beschreibung (Edge-Description)).

Oftmals sollen Beschreibungsinstanzen anhand einer textuellen Darstellung konstruiert werden. In diesem Fall wird ein *Parser* benötigt. Wir gehen hier wie folgt vor. Zunächst wird durch Analyse des Wortes w die entspr. Beschreibungsklasse identifiziert und instantiiert.² Die entsprechende Klassifikationsmethode wird als *substratlokal* angenommen. Beim Anlegen der Beschreibungsinstanz wird das originale Wort w als Attributwert der Instanz gespeichert. Mittels einer automatisch nach Erzeugung der Beschreibungsinstanz aufgerufenen Initialisierungsmethode kann diese *textuelle Darstellung* w dann verwendet werden, um die für die Beschreibungsinstanz benötigte interne Repräsentation anhand von w zu erzeugen. Speziell im Falle eines *geometrischen Substrates* ist dies nicht trivial. Die Beschreibungsinstanzen haben also die Eigenschaft, *selbstübersetzend* zu sein, nachdem sie erzeugt wurden. Die eigentlichen Parser sind daher als beschreibungsklassenlokale Methoden ausgelegt; der substratlokale Parser löst lediglich das Klassifikationsproblem.

Oftmals wird ein Substrat auch gänzlich auf eine solche Klassifikationsmethode verzichten können, und es ist dann ausreichend, lediglich gewisse Knoten- und Kanten-Beschreibungsklassen als *Standard-Beschreibungsklassen* dieser Substratklasse auszuzeichnen. Dies entspricht auch der Intuition, dass gewisse spezialisierte Substrat-Arten bzw. -Klassen lediglich bestimmte Spra-

¹Das Muster sieht vor, dass Objekte „vorweg“ erzeugt und in einem „Objektpool“ vorgehalten werden. Eine Instanz muss dann nicht mehr zur Laufzeit erzeugt werden, sondern wird lediglich aus dem Pool angefordert und initialisiert [Shalloway u. Trott 2006].

²In CLOS kann der Objekterzeugungsfunktion `make-instance` eine Variable übergeben werden, die an einen Klassennamen gebunden ist.

5.1 Realisierung des Substrat-Datenmodells

chen für $\mathcal{L}_V$ und $\mathcal{L}_E$ zulassen. Die im Folgenden vorgestellten Substratklassen zeichnen zwar Standard-Beschreibungsklassen aus – da es sich jedoch um recht allgemeine Substrate handelt, ist die Assoziation von Substrat und Standard-Beschreibungsklassen für diese Wurzelklassen nicht sehr streng. Die entspricht der Vorstellung, dass die Knoten und Kanten solch allgemeiner Substrate letztlich für Wörter mehr oder weniger beliebiger Beschreibungssprachen als Informationsträger dienen sollen. Erst für spezialisiertere Substrate wird es zur Erbringung ihrer Dienste wesentlich sein, dass lediglich bestimmte Beschreibungsinstanzen zugelassen werden (beispielsweise für das RCC-Substrat, s.u.). Es obliegt daher der spezialisierten Substratklasse, diese Einschränkungen zu erzwingen. Dementsprechend werden auch die Basis-Beschreibungsklassen recht generell ausfallen.

Zur *Population* eines Substrates müssen Knoten, Kanten, und Beschreibungsinstanzen zur Annotation dieser konstruiert werden. Wir sehen hier zusätzlich zu den vorhandenen Methoden in der Schnittstelle auch eine *Endnutzersyntax* (eine COMMON LISP-Makrosprache) vor, die es ermöglicht, Substrate literal zu konstruieren, z.B.

```
(in-substrate test :type 'simple-descriptions-substrate)

(node michael (node-description person))
(node margrit (node-description mother))
(edge margrit michael (edge-description has-son))
```

Die Sprache ist z.B. vergleichbar mit der KRSS-ABox-„Tell“-Sprache zur Erzeugung von Konzept- und Rollen-Assertionen in einer ABox, trägt hier aber notwendigerweise der erforderlichen Variabilität bzw. Flexibilität Rechnung. Hier wird zunächst eine Substrat-Instanz namens *test* der Klasse Einfache-Beschreibungen-Substrat (Simple-Descriptions-Substrate) angelegt. Dieses Substrat wird zum *aktuellen Substrat* und stellt damit den Kontext bereit, auf den sich die nachfolgenden Konstruktionsoperationen beziehen. Es werden dann die beiden Knoten *margrit* und *michael* konstruiert, und eine Kante wird zwischen diesen beiden etabliert. Der Ausdruck `(node-description person)` konstruiert eine Knoten-Beschreibungsinstanz, und `(edge-description has-son)` eine Kanten-Beschreibungsinstanz.

Es stellt sich die Frage, Instanzen welcher Klassen die Konstruktoren hier erzeugen – offensichtlich möchte man in einem Rahmenwerk, welches die Ableitung spezialisierter Knoten-, Kanten- und Beschreibungsklassen ermöglicht, das sog. „*Make isn't generic*“-Problem lösen [Kiczales 1993], also von Konstruktoren wie `node` und `node-description` fordern, dass sie Instanzen der *anwendungsspezifischen Unterklassen* erzeugen mögen. Die Klassennamen der zu erzeugenden Instanzen dürfen also nicht literal im Quellcode genannt werden;

eine Indirektion ist erforderlich. In der Literatur sind hier diverse Vorgehensweisen bekannt, z.B. das *Fabrik-Muster* aus [Gamma u. a. 1996].

Wir verwenden hier die gleiche Lösung wie bereits für die Beschreibungsinstanzen skizziert. Es werden von einer Substratklasse einfach *Standardklassen* ausgezeichnet. Ein Substrat zeichnet daher nicht nur seine Standard-Beschreibungsklassen, sondern auch seine Standard-Knoten- und Standard-Kanten-Klassen aus. Anhand des durch das aktuelle Substrat gegebenen *Kontextes* werden dann Standardklassen für alle intern vom Rahmenwerk zu erzeugenden Objekte ausgezeichnet.³ Natürlich sollte dieses Verhalten durch explizite Angabe eines Klassennamens auch unterdrückt werden können; sowohl die Methoden- als auch die Makro-Schnittstelle erlauben daher die optionale explizite Übergabe eines `class-` bzw. `type-`Argumentes (z.B. `(node-description michael :type 'simple-description)`).

Das Rahmenwerk soll mehr als ein Substrat verwalten können. Ein Substrat hat einen eindeutigen Namen. Der sog. Substrat-Verwalter (Substrate-Manager) ist verantwortlich für die Konstruktion und *namentliche Lokalisierung von Substraten* (Ermittlung einer Referenz auf das Substrat anhand des Namens). Da nur ein Substrat-Manager benötigt wird, kann diese Klasse konzeptuell als *Singleton* entworfen werden. Das *aktuelle Substrat* wird vom Manager etabliert und stellt den *Kontext* für Knoten- und Kanten-Konstruktionsoperationen zur Population des Substrates bereit. Der Kontext wird wie beschrieben zur Ermittlung der Standardklassen etc. verwendet. Der Manager stellt zudem die Endnutzersyntax bereit und delegiert die Aufrufe an das entspr. Substrat.

Die Schnittstellen der Substrat-, Knoten- und Kanten-Klassen als auch der Beschreibungsklassen sehen *Inferenzmethoden* zum Test auf *Erfüllbarkeit*, *Unerfüllbarkeit* und *Gültigkeit* vor (Methoden `consistent-p`, `inconsistent-p`, `tautological-p`). Wir sehen hier explizit *drei verschiedene* Methoden vor, da wechselseitige Reduktion (von beispielsweise Unerfüllbarkeit auf Gültigkeit) durch eine Schablonenmethode nicht immer möglich ist. Zumal möchte man die Freiheit haben, *spezialisierte Inferenzverfahren* zur Lösung dieser Probleme in das Rahmenwerk integrieren zu können, beispielsweise unvollständige Verfahren. Auch in diesem Falle ist eine automatisch durchgeführte Reduktion nicht immer ohne weiteres möglich.

Dennoch sind Schablonenmethoden vorgesehen. Offensichtlich muss Erfüllbarkeit einer Beschreibungsinstanz zunächst sprachspezifisch implementiert werden. Die Erfüllbarkeit eines Knotens ist in einfachen Substraten nun einfach durch die Erfüllbarkeit seiner Knoten-Beschreibungsinstanz gegeben, und analog für die Kanten. Ein einfacher (nicht notwendigerweise vollständiger) Substrat-Erfüllbarkeitstest ist durch eine einfache Schablonenmethode ge-

³Statt `(make-instance 'node)` steht im Code daher beispielsweise `(make-instance (get-standard-node-class *current-substrate*))`.

5.1 Realisierung des Substrat-Datenmodells

geben, die isoliert die Erfüllbarkeit aller Knoten und Kanten testet. Ähnlich können Schablonenmethoden zur Prüfung auf Unerfüllbarkeit angegeben werden.

Während diese Inferenzmethoden als unäre boolesche Methoden in der Schnittstelle bereitgestellt werden, z.B. $\text{consistent-p} : X \mapsto \{\text{TRUE}, \text{FALSE}\}$, so sind hier insbesondere auch *binäre* boolesche Inferenzmethoden erforderlich, z.B. zur Prüfung auf *Folgerbarkeit* mittels $\text{implies-p} : X \times X \mapsto \{\text{TRUE}, \text{FALSE}\}$, wobei X eine Substratklasse, eine Beschreibungs-kategorie, usw. sei. Folgerbarkeit bzw. logische Implikation ist offensichtlich für *Paare* von Beschreibungsinstanzen, aber auch für Paare aus Substrat und Beschreibungsinstanz etc. zu definieren. Offensichtlich sollten bei der dynamischen Bindung die Typen beider Argument-Instanzen berücksichtigt werden; so können beispielsweise auch optimierte Beweisverfahren automatisch ausgewählt werden. Die bereits erwähnten CLOS-Multimethoden sind für diesen Zweck also ideal. Eine Schablonenmethode implementiert dann auch noch equivalent-p durch wechselseitige Reduktion auf zwei Implikationstests.

5.1.2.2. Transiente, persistente, virtuelle Substrate und Nutzung von Mehrfachvererbung

Die Wurzel der Substrat-Klassenhierarchie ist durch die abstrakte Klasse *Substrat* (*Substrate*) gegeben. Da diese Klasse abstrakt ist, kann sie nicht instantiiert werden; sie stellt also keine Datenstrukturen zur Repräsentation des Objektgraphen bereit.

Zur Berücksichtigung der *physischen Darstellungsaspekte*, also zur Festlegung, wie ein Substrat-Graph durch Datenstrukturen repräsentiert bzw. dargestellt wird, leiten wir zwei weitere abstrakte Klassen ab:

- Die Klasse *HS-Substrat* (*MM-Substrate*) speichert den Substrat-Graphen als Objektgraphen im Hauptspeicher (Main Memory). Das Substrat ist transient.
- Die Klasse *DB-Substrat* (*DB-Substrate*) speichert den Substrat-Graphen in einer relationalen Datenbank. Das Substrat ist persistent.

Die Klassen sind zwar abstrakt, stellen jedoch Standard-Implementationen zur Verwaltung dieser Speicherstrukturen in Form von Schablonenmethoden bereit. So führt der Aufruf von *create-node* der Klasse *HS-Substrat* (*MM-Substrate*) zum Anlegen einer CLOS-Instanz, während der selbe Aufruf für *DB-Substrat* (*DB-Substrate*) zur Erzeugung eines Datenbank-Eintrages führt. Wie oben diskutiert, können bei Bedarf lokale CLOS-Proxy-Objekte (statisch oder dynamisch) angelegt und konsistent mitverwaltet werden.

Die physische Darstellung einer konkreten *anwendungsspezifischen* Substrat-

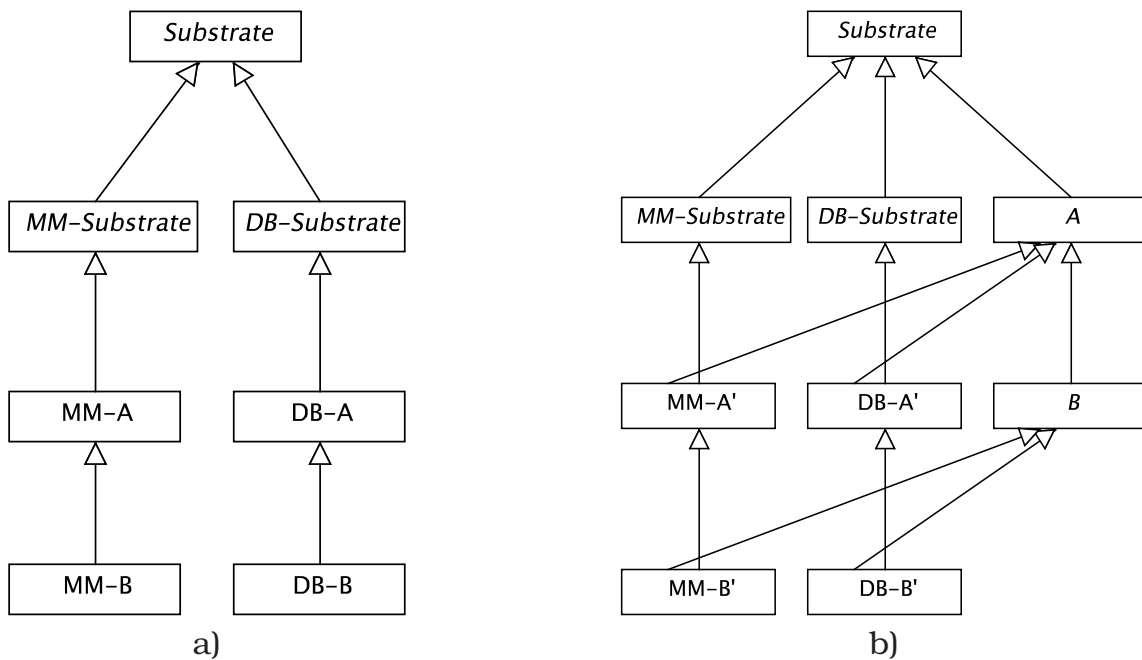


Abbildung 5.1.: Nutzung von Mehrfachvererbung zur Vermeidung von Code-Replikation. Die Klassen mit '-Suffix im Namen in b) sind implementationslos.

klasse, sagen wir A, kann nun durch Ableitung von DB-Substrat (DB-Substrate) oder von HS-Substrat (MM-Substrate) ererbt und somit festgelegt werden. Für viele anwendungsspezifische Klassen werden die in HS-Substrat (MM-Substrate) bzw. DB-Substrat (DB-Substrate) voreingestellten Standard-Implementationen ausreichend sein, sodass bei der Implementation von A keine physischen Darstellungsaspekte berücksichtigt werden müssen. Die Situation ändert sich, wenn z.B. anwendungsklassenspezifische Indexstrukturen erforderlich werden.

Prinzipiell ist der Diskriminator „transient vs. persistent“ eine Entwurfsdimension, die *orthogonal* zu und somit unabhängig von der *anwendungsspezifischen Funktionalität des Substrates* ist. Es handelt sich primär um eine nichtfunktionale Eigenschaft. Somit sind für A prinzipiell beide physischen Repräsentationsmöglichkeiten wünschenswert; offensichtlich könnten nun zwei Unterklassen DB-A und MM-A gebildet werden, s. Abb. 5.1a). Da die Implementation von A nun i.d.R. komplex sein wird, führt ein solches Vorgehen zu Code-Replikation und einem schlecht wartbaren Entwurf.

Wir nutzen daher Mehrfachvererbung wie in Abb. 5.1b) gezeigt, indem eine *funktionale* und eine *nichtfunktionale* Klassenhierarchie getrennt gepflegt werden.⁴ Wir konzentrieren uns im Folgenden fast ausschließlich auf die Darstel-

⁴Oftmals werden derartige Repräsentationsspezifika auch durch unterschiedliche *Meta-*

5.1 Realisierung des Substrat-Datenmodells

lung der *funktionalen Aspekte* bzw. die *funktionale Klassenhierarchie* der Substratklassen. Durch Mehrfachvererbung werden nun *Produktklassen* (MM-A', DB-A') abgeleitet, deren *gesamte Funktionalität ererbt* werden kann. Hierdurch erreichen wir eine gute *Wartbarkeit* und *Modularität* für den Code. Dabei nehmen wir die durch CLOS bereitgestellte Semantik für Mehrfachvererbung an. Definieren zwei Oberklassen einer Klasse ein gleichnamiges Attribut bzw. einen gleichnamigen „Slot“, so wird dieser Slot in der Unterklasse weder dupliziert noch muss der Vererbungspfad zur Disambiguierung festgelegt werden. Dies ist nicht erforderlich, da COMMON LISP dynamisch typisiert ist. Die *lokale Klassenpräzedenzliste* einer Klasse stellen wir (wenn relevant) in den UML-Diagrammen durch eine *links-nach-rechts-Schreibweise* der Enden der Generalisierungsbeziehungen auf der Seite der Unterklasse dar. Im obigen Diagramm Abb. 5.1b) hätten also die für MM-A' durch A bereitgestellten Methoden *Präzedenz* gegenüber gleichnamigen durch MM-Substrate bereitgestellten Methoden.

Diese Nutzung von Mehrfachvererbung kann nur dann funktionieren, wenn A so implementiert wird, dass keine (oder nur wenige, einfach zu redigierende) Annahmen bzgl. der physischen Repräsentation gemacht werden. *Physische Darstellungsunabhängigkeit* muss daher erreicht werden; wir beschreiben im nächsten Abschnitte, wie diese mittels Iteratoren prinzipiell erreicht werden kann. Das zugrundeliegende Prinzip „Programmiere gegen eine Schnittstelle, nicht gegen eine Implementation bzw. Ausprägung!“ ist wohlbekannt [Gamma u. a. 1996, S. 21].

Prinzipiell könnte A auch *direkt* von der abstrakten Klasse Substrat (Substrate) erben. In diesem Falle ist A selbst für eine „Realisierung der Speicherdarstellung“ verantwortlich. Insbesondere kann der Substrat-Graph dann auch rein virtuell bleiben. Wie erwähnt, kann so beispielsweise eine Substrat-Perspektive auf eine RACERPRO-ABox hergestellt werden. Das Anlegen eines Knotens mittels create-node wird dann an RACERPRO delegiert, sodass ein ABox-Individuum erzeugt wird. Wiederum ist die Perspektive der dynamisch und/oder statisch erzeugten Proxy-Objekte gegeben, wie erläutert.

Nutzt A die Dienste von HS-Substrat (MM-Substrate) bzw. DB-Substrat (DB-Substrate) zur physischen Repräsentation, so wird A i.d.R. abstrakt bleiben, und erst die Produktklassen (MM-A', DB-A') sind instantiierbar. Dementsprechend ist A in Abb. 5.1b) als abstrakte Klasse (in kursiver Schrift) dargestellt.

Klassen manifestiert; aus dieser Perspektive wären HS-Substrat (MM-Substrate) und DB-Substrat (DB-Substrate) dann Meta-Klassen.

5.1.2.3. Physische Darstellungsunabhängigkeit durch Iteratoren

Damit die Implementation der Klasse A aus Abb. 5.1b) nun so geschrieben werden kann, dass sie ohne wesentliche Änderungen sowohl die Implementation des Objektgraphen durch HS-Substrat (MM-Substrate) als auch die Implementation des Objektgraphen durch DB-Substrat (DB-Substrate) in Anspruch nehmen kann, dürfen im A implementierenden Code keine Annahmen bzgl. der physischen Speicherdarstellung gemacht werden. Zum Zugriff auf den Substrat-Graphen dürfen lediglich Methoden aus der Schnittstelle der Klasse Substrat (Substrate) aufgerufen werden. In [Gamma u. a. 1996, S. 166] werden verschiedene Techniken diskutiert, die es erlauben

... eine Abstraktion von ihrer *Implementierung* zu entkoppeln.

Viele der anwendungsspezifisch für A definierten Operationen werden auf die Knoten-und-Kanten-Struktur des Substrates zugreifen müssen. Offensichtlich können wir Methoden wie *get-nodes*, *get-edges* in die Schnittstelle der Klasse Substrat (Substrate) aufnehmen, die dann für HS-Substrat (MM-Substrate) und DB-Substrat (DB-Substrate) entsprechend ausimplementiert werden, und die prinzipiell eine *Liste* von Objekten zurückliefern, sodass von der physischen Speicherdarstellung im Code von A abstrahiert werden kann.

Diese *mengen- bzw. listenwertigen Schnittstellen* sind jedoch oftmals suboptimal. Wir hatten bereits in Kap. 3 identifiziert, dass beispielsweise „echte“ inkrementelle Anfragebeantwortung nur mit inkrementellen Schnittstellen realisiert werden kann. Viele Operationen werden zudem nur einen Bruchteil der zurückgelieferten Objekte betrachten. Für große Substrate wird auch die Konstruktion der Ergebnislisten speicheraufwendig sein (und für ein DB-Substrat (DB-Substrate) u.U. mehr Hauptspeicher als vorhanden benötigen).

Wir sehen daher *Iteratoren* vor, die einen inkrementellen und darstellungstransparenten iterativen Zugriff auf die Knoten- und Kanten eines Substrates ermöglichen. Anders als in der Java-Welt sind Iteratoren hier keine Instanzen einer Klasse Node-Iterator mit den Methoden *get-next*, *has-next-p*, sondern einfache Methoden. Die Klasse Substrat (Substrate) stellt die Methoden *loop-over-nodes* und *loop-over-edges* bereit. Wie auch *get-nodes*, *get-edges* können diese dann unterklassenspezifisch redefiniert werden. Den Iterator-Methoden kann ein *Funktionsobjekt-Argument* übergeben werden, welches den Schleifenkörper darstellt. Gleichzeitig stellen wir auch eine Makro-Schnittstelle für diese Iteratoren bereit, sodass die Iteratoren wie normale COMMON LISP-Schleifenkonstrukte genutzt werden können. Die Makros erzeugen intern automatisch die benötigten Funktionsobjekte und delegieren dann an die entspr. Iterator-Methode. Von der Implementierung der Iteratoren kann dann abstrahiert werden; gleichzeitig wird dynamisches Binden ermöglicht. Hier zeigt sich die Generizität von COMMON LISP. *Iterator-Methode und Iterator-*

5.1 Realisierung des Substrat-Datenmodells

Makro sind in der folgenden Darstellung daher stets als ein Paar zu betrachten (wir verwenden aus Gründen der Einfachheit den gleichen Namen für Methode und Makro).

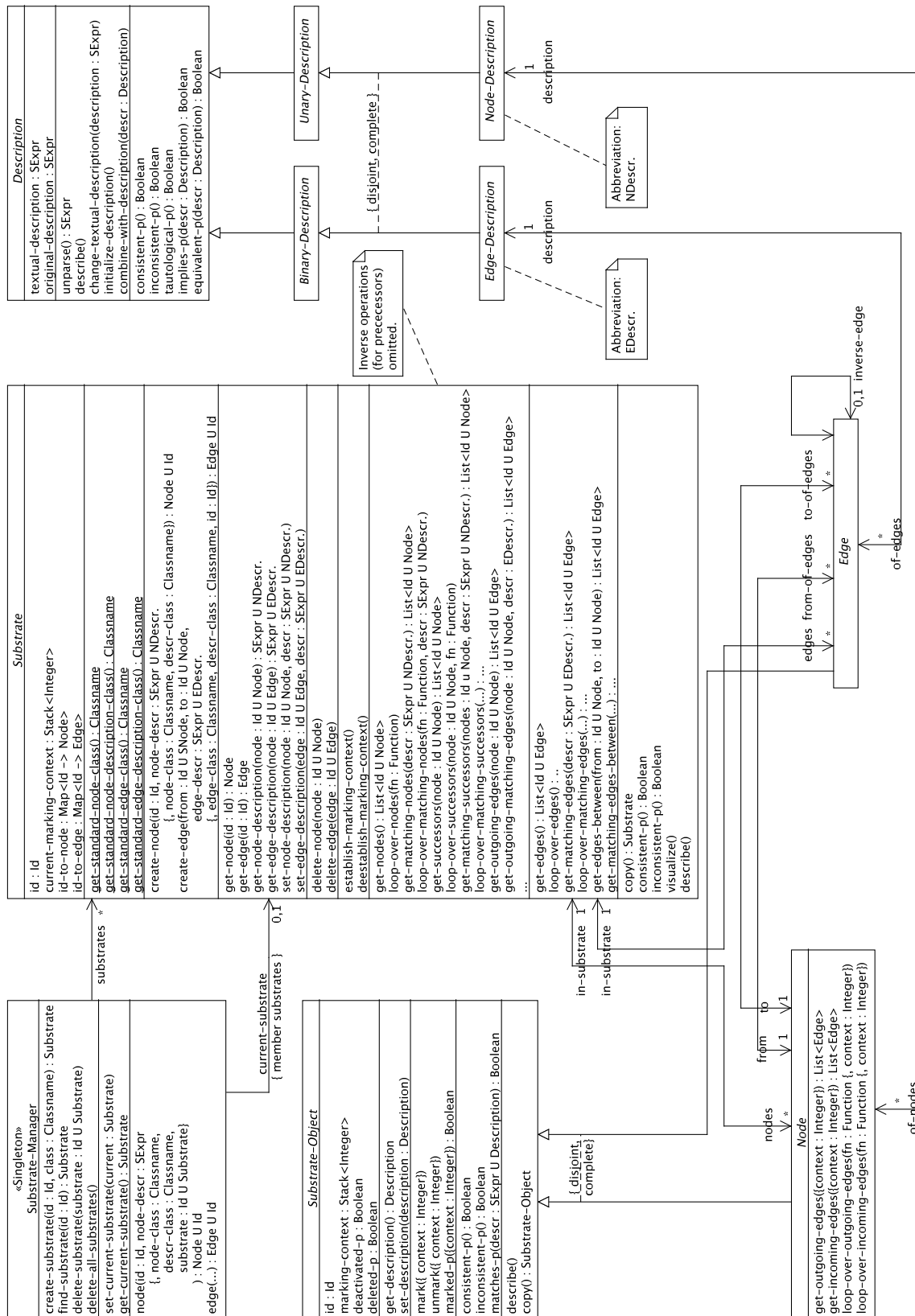
Oftmals sollen bestimmte Operationen lediglich *Subgraphen* des Substrat-Graphen betrachten. Die Iteratoren können daher auf Subgraphen fokussiert werden, indem die entspr. Knoten und Kanten *markiert* werden. Die Iteratoren iterieren dann lediglich über markierte Knoten und Kanten. Dabei werden lediglich solche Knoten und Kanten betrachtet, deren Markierung dem aktuellen *Markierungskontext* entspricht. Solche Markierungskontexte können auch geschachtelt werden, falls für einen Algorithmus mehrere Markierungskontexte aufgesetzt werden müssen (z.B. für rekursive Graphalgorithmen).

Wir sehen noch weitere Markierungs-Attribute für Knoten und Kanten vor. Beispielsweise können Knoten und Kanten als inaktiv oder als gelöscht markiert werden (es ist manchmal sinnvoller, ein Objekt lediglich als gelöscht zu markieren, als es wirklich physisch zu löschen).

5.1.2.4. Klassendiagramm des Basis-Entwurfs

Der vorgenommene Entwurf ist in Abb. 5.2 als UML-Klassendiagramm dargestellt. Wir haben die in [Fowler 2004] angegebenen Richtlinien zur Erstellung von UML-Diagrammen angewendet. Wir wollen mit dem Diagramm die wesentlichsten Charakteristika unseres Entwurfs kommunizieren und dokumentieren.

Wie üblich werden abstrakte Klassen kursiv dargestellt. Klassenattribute und -operationen werden unterstrichen dargestellt. Wir verzichten bewusst auf die Angabe von Sichtbarkeitsmodifizierern (privat, öffentlich, etc.); ebenso zeichnen wir Methoden (im Gegensatz zu Klassen) nicht als abstrakt aus. Zudem werden eigene Basis-Datentypen verwendet, die selbsterklärend sein sollten und nicht unbedingt dem UML-Standard entsprechen. Mit dem Typ *T* parametrierter Kollektionstypen („Generics“) bezeichnen wir wie folgt: *List<T>*, *Set<T>*, *Bag<T>*, etc. Kollektionstypen werden ausschließlich für Operationen verwendet; für kollektionstypwertige Attribute verwenden wir stattdessen Assoziationen mit entspr. Multiplizitätsangaben [Fowler 2004, S. 38]. Der Typ *Pair<T1,T2>* bezeichnet das Kreuzprodukt aus *T1* und *T2*; *Pair<T>* steht für *Pair<T,T>*. *Funktionen* mit Urbildtyp *T1* und Bildtyp *T2* notieren wir mittels *Map(T1 -> T2)*; meist werden diese als Hashtabellen implementiert. Der Typ *Classname* wird zur Referenzierung einer Klasse verwendet (in COMMON LISP werden hierfür Symbole genutzt); hier erlauben wir auch *NIL* als *Classname*. Als IDs können Strings oder Symbole verwendet werden. Mittels *Enum(T1, ..., Tn)* wird ein Aufzählungstyp beschrieben (beispielsweise *ampelfarbe : Enum(Rot, Gelb, Grün)*). *Funktionsobjekte (Closures)* wer-



5.1 Realisierung des Substrat-Datenmodells

den mittels Function oder auch Predicate notiert, falls es sich um eine Funktion mit Rückgabetyt Boolean handelt. Der Typ SExpr notiert COMMON LISP *S-Expressions*. Aus Platzgründen wird eine abkürzende Schreibweise für *überladene Operationen* verwendet; eine Operation mit Signatur $op(i : X \cup Y)$ steht in Wirklichkeit für 2 Operationen $op(i : X)$, $op(i : Y)$.⁵ Das Zeichen „U“ soll Vereinigung andeuten. Die in COMMON LISP-Funktionen bzw. -Methoden (genauer: generischen Funktionen) verwendbaren *Keyword-Argumente* notieren wir im UML-Diagramm mittels geschwungener Klammern „{...}“ (es handelt sich nicht um offizielles UML). Wir verzichten auf die Angabe einer Operations-Signatur und notieren nur „...“, falls diese offensichtlich identisch zur Signatur einer unmittelbar zuvor aufgelisteten Operation ist. Geben wir keinen Rückgabetyt an, so ist dieser irrelevant (die Methode wird nur aufgrund ihrer Seiteneffekte aufgerufen). Die meisten dargestellten Operationen bzw. Methoden sind Multimethoden. Diese sind prinzipiell nicht einer einzigen Klasse zuordbar. Im UML-Klassendiagramm ordnen wir eine Multimethode jedoch ihrer ersten Argument-Klasse zu.

Der Basis-Entwurf sollte aufgrund der erfolgten Erläuterungen weitgehend selbsterklärend sein. Wir erläutern einige zentrale Punkte:

- Der Substrat-Verwalter (Substrate-Manager) stellt die Endnutzersyntax bereit und ist für die Erzeugung, Lokalisierung und Verwaltung von Substraten verantwortlich. Er etabliert ein *aktuelles Substrat*, welches den Kontext für die Konstruktoren in Endnutzersyntax (`node`, `edge`) aufspannt. Diese Operationen werden vom Manager an das entspr. Substrat weiterdelegiert und beziehen sich standardmäßig auf das aktuelle Substrat. Mittels des `substrate`-Argumentes können jedoch stets beliebige Substrate referenziert werden; hier sind sowohl IDs von Substraten als auch Instanzen der Klasse `Substrat` (`Substrate`) als Argument zulässig. Analog gilt auch für die Operationen, die Knoten und Kanten als Argumente benötigen, dass sie sowohl Objekte als auch IDs von Objekten akzeptieren.
- Die Klasse `Substrat` (`Substrate`) stellt mittels der Klassenmethoden (s. `get-standard-node-class`, `get-standard-node-description-class`, ...) die verwendeten Standardklassen klassenglobal ein, sofern das Substrat nicht virtuell ist – in diesem Fall wird `NIL` zurückgegeben. Wie oben erwähnt, akzeptieren wir `NIL` als gültigen Classname. Den Konstruktoren können jedoch stets `class`-Argumente zur Abweichung von den Standardklassen übergeben werden; die Argumente sind daher optional. Dies gilt auch für die Konstruktoren, die der Substrat-Verwalter

⁵Tatsächlich handelt es sich hier nicht um einfaches Überladen, sondern um Überschreiben, da `X` und `Y` Klassen sein können und somit auch bzgl. `i` spät gebunden wird. Eine Operation $op(i : X \cup Y)$ der Klasse `A` entspricht daher konzeptionell zwei CLOS-Multimethoden: `(defmethod op ((instance A) (i X)) ...)` und `(defmethod op ((instance A) (i Y)) ...)`.

(Substrate-Manager) bereitstellt.

Da ein Substrat virtuell sein kann und somit evtl. gar keine Objektgraph-Repräsentation des Substrat-Graphen vorhanden ist (also keine Instanzen von Knoten (Node), Kante (Edge), Knoten-Beschreibung (Node-Description), Kanten-Beschreibung (Edge-Description) existieren), müssen die Operationen der Klasse Substrat (Substrate) (wie bereits erwähnt) als Argumente wann immer möglich auch symbolische IDs statt Objekte akzeptieren. Daher wird beispielsweise die Methode `get-outgoing-edges`, die primär der Klasse Knoten (Node) zuzuordnen ist, auch in der Schnittstelle von Substrat (Substrate) angeboten, und dort akzeptiert sie `from`-Knoten-Argumente vom Typ `Id U Node`. Bekommt sie ein `from`-Argument vom Typ `node`, so delegiert sie an die entspr. Methode `get-outgoing-edges` der Klasse Knoten (Node). Dieses Delegationsmuster gibt uns die benötigte Flexibilität. Wir haben diese „Operations-Migration in die Klasse Substrat (Substrate)“ nicht für alle Operationen im Klassendiagramm aufgezeigt. Die Idee sollte klar geworden sein.

Zu jeder mengen- bzw. listenwertigen Operation gibt es einen entsprechenden Iterator. Wir haben nicht alle Operationen bzw. Iteratoren aufgeführt; insbesondere gibt es noch Knoten-Operationen, die sich auf die Vorgänger und ausgehenden Kanten beziehen, wie `get-predecessors`, `loop-over-predecessors`, `get-outgoing-edges`, `loop-over-outgoing-edges`, etc. Die Signaturen dieser Operationen sind identisch zu den Signaturen ihrer inversen Operationen. Wir möchten hier noch auf die Operationen mit `matching` im Namen hinweisen, die insbesondere für die Realisierung der Substrat-Anfragesprachen eine wichtige Rolle spielen werden (s.u.). Es handelt sich hier um *informierte* Operationen.

Bzgl. der Beschreibungsinstanzen sind die Operationen der Klasse Substrat (Substrate) zudem so ausgelegt, dass sie stets auch mit „unübersetzten“ textuellen Darstellungen bzw. S-Expressions statt mit Beschreibungsinstanzen der Klasse Knoten-Beschreibung (Node-Description) bzw. Kanten-Beschreibung (Edge-Description) arbeiten können.

Zudem sind die Operationen der Klasse Substrat (Substrate) nahezu vollständig über den Substrat-Verwalter (Substrate-Manager) erreichbar (ein Substrat (Substrate) hat ebenfalls eine ID); hier wird das gleiche Delegationsmuster verwendet.

- Die Klassen zur Repräsentation des Objektgraphen sind selbsterklärend. Kanten sind gerichtet und referenzieren ihren Ursprungsknoten (`from`) und Zielknoten (`to`), sowie ihre (evtl. vorhandene) *inverse* Kante (`inverse-edge`). Knoten (Klasse Knoten (Node)) referenzieren ihre eingehenden und ausgehenden Kanten (`incoming-edges`, `outgoing-edges`). Knoten und Kanten sind mit Beschreibungsobjekten (Instanzen von Knoten-

5.1 Realisierung des Substrat-Datenmodells

Beschreibung (Node-Description) und Kanten-Beschreibung (Edge-Description)) annotiert bzw. referenzieren diese mittels *description*. Sowohl Knoten als auch Kanten sind Substrat-Objekte (Klasse Substrat-Objekt (Substrate-Object)); diese Klasse stellt herausfaktorierte Gemeinsamkeiten bereit, u.a. die besprochene Markierungsfunktionalität. Analog stellt die Klasse Beschreibung (Description) herausfaktorierte Gemeinsamkeiten von Knoten- und Kanten-Beschreibungen und Methoden bereit (Test auf Erfüllbarkeit etc.).

Man mag an dem Entwurf kritisieren, dass er doch recht *schwergewichtig* erscheint. Die konsequente Verwendung der OOP führt selbst für relativ kleine Substrate-Graphen (sofern es sich nicht um „rein virtuelle Substrate“ handelt) schon zu Objektgraphen mit mehreren hundert Instanzen. Die Erzeugung einer CLOS-Instanz ist als laufzeit- und speicheraufwendige Operation bekannt. Durch Verwendung des *Objekt-Pool-Musters* kann der Laufzeit- und Speicher-aufwand reduziert werden.

5.1.2.5. Standard-Beschreibungsklassen

Die bereitgestellten Standard-Beschreibungsklassen sind in Abb. 5.3 dargestellt.

Die abstrakte Klasse Einfache-Beschreibung (Simple-Description) stellt *einfache Beschreibungen* bereit; hierbei handelt es sich um eingeschränkte COMMON LISP-S-Expressions. Als Literale in diesen S-Expressions sind Symbole, Zahlen, Zeichen und Zeichenkette erlaubt; diese können beliebig in COMMON LISP-Listen bzw. -Bäumen geschachtelt werden. Mit dieser Definition ist beispielsweise "Michael" eine einfache Beschreibung, aber auch ((("Michael") "Achim" ("Wessel")) (34 35)) oder das KRSS-Konzept (and (some r c) d). Eine einfache Beschreibung sagt nichts über deren Semantik aus. Einfache Beschreibungen sind daher Datenstrukturen, die interpretiert werden müssen. Dementsprechend ist die Klasse Einfache-Beschreibung (Simple-Description) abstrakt; viele Substratklassen werden dennoch Beschreibungsinstanzen vom Typ Einfache-Beschreibung (Simple-Description) beispielsweise für Argumente akzeptieren, womit dann Beschreibungsinstanzen konkreter Unterklassen von Einfache-Beschreibung (Simple-Description) gemeint sind. Die folgenden konkreten Unterklassen stellen nun eine Interpretation dieser einfachen Beschreibungsinstanzen bereit:

- Die konkrete Unterklasse Einfache-Konjunktive-Beschreibung (Simple-Conjunctive-Description) interpretiert eine COMMON LISP-Liste von Literalen (oder ein einzelnes Literal) einer einfachen Beschreibung als eine *positive Konjunktion*. Die Konjunkte dieser Konjunktion haben für die Klasse selbst keine weitere Struktur; die Struktur der Konjunkte kann daher



5.1 Realisierung des Substrat-Datenmodells

unterklassenspezifisch interpretiert werden. Die Konjunkte werden aus der Perspektive der Klasse daher als einfache positive Propositionen wie in der Aussagenlogik behandelt. Negative Propositionen sind nicht möglich. Die Konjunkte für obiges Beispiel wären also ("Michael"), "Achim", ("Wessel") und (34 35). Obwohl das Konjunkt ("Wessel") eine komplexe Struktur hat, wird es semantisch als einfache (atomare) Proposition behandelt.

- Die konkrete Unterklasse Einfache-Disjunktive-Beschreibung (Simple-Disjunctive-Description) interpretiert die Liste von Literalen einer einfachen Beschreibung als *Disjunktion*. Wiederum ist die *Struktur der Disjunkte* aus der Perspektive der Klasse uninteressant; die Struktur kann unterklassenspezifisch interpretiert werden.
- Die konkrete Unterklasse Einfache-KNF-Beschreibung (Simple-CNF-Description) der Klasse Einfache-Konjunktive-Beschreibung (Simple-Conjunctive-Description) interpretiert Konjunkte, die Listen sind, wiederum als Disjunktionen. Die Struktur der anderen (literalen) Konjunkte bleibt uninterpretiert. Wir sprechen von *einfachen Beschreibungen in konjunktiver Normalform*. Somit würden die Literale (34 35) und ("Wessel") im obigen Beispiel als Disjunktionen (von positiven Disjunkten bzw. Literalen) gedeutet. Als textuelle Beschreibung einer solchen Beschreibungsinstanz könnte man den Ausdruck ((("Michael") "Achim" ("Wessel") (34 35))) also als Aussage auffassen, dass Michael Achim Wessel entweder 34 oder 35 Jahre alt ist.

Zudem wird die Syntax der einfachen Beschreibungen so eingeschränkt, dass die Disjunktionen lediglich atomare Literale enthalten dürfen:

Definition 92 Die Menge der einfachen KNF-Beschreibungen, $\mathcal{L}_{SKNF}$ mit $w \in \mathcal{L}_{SKNF}$, gehorcht folgender Grammatik (in EBNF):

w	::=	<i>konjunktion</i>	
<i>konjunktion</i>	::=	<i>literal</i>	
		(<i>konjunkt</i> ⁺)	
<i>konjunkt</i>	::=	<i>literal</i>	
		(<i>literal</i> ⁺)	
<i>literal</i>	::=	<i>symbol_literal</i>	
		<i>zeichen_literal</i>	
		<i>zeichenkette_literal</i>	
		<i>numerisches_literal</i>	

$symbol\_literal ::=$ ein COMMON LISP-Symbol, z.B. `symbol`
 $zeichen\_literal ::=$ ein COMMON LISP-Zeichen, z.B. `#\a`
 $zeichenkette\_literal ::=$ eine COMMON LISP-Zeichenkette, z.B. `"string"`
 $numerisches\_literal ::=$ eine COMMON LISP-Zahl, z.B. `123`, `3/2`, `123.23`

■

Somit ist $((\text{"Michael"}) \text{"Achim"} (\text{"Wessel"}) (34 \ 35))$ ein Wort aus $\mathcal{L}_{SKNF}$ und daher als Notation für die aussagenlogische Formel in KNF $\text{"Michael"} \wedge \text{"Achim"} \wedge \text{"Wessel"} \wedge (34 \vee 35)$ zu deuten; die Struktur bzw. Domäne der Literale ist hier wie gesagt irrelevant.

- Wir lassen auch das Analogon zu KNF-Beschreibungen zu – die Klasse Einfache-DNF-Beschreibung (Simple-DNF-Description) stellt DNF-Beschreibungen bereit. Es handelt sich um einen Unterklasse von Einfache-Disjunktive-Beschreibung (Simple-Disjunctive-Description). Listenwertige Disjunkte werden als Konjunktionen interpretiert. Für $\mathcal{L}_{SDNF}$ gilt die gleiche Grammatik wie für $\mathcal{L}_{SKNF}$, doch wird *konjunkt* gegen *disjunkt* ersetzt.

Wie angedeutet existiert auch ein Parser, der die benötigte Beschreibungs-klasse für ein Wort anhand des Wortes ermitteln kann. Offensichtlich werden dann jedoch zusätzliche Schlüsselwörter wie `:and`, `:or` zur Disambiguierung benötigt.

Die einfachen KNF- und DNF-Beschreibungen sind für Anwendungen interessant. Zum einen sind sie stets erfüllbar (da keine negativen Propositionen verwendet werden können); zum anderen erlauben sie einfache Disjunktionen und können somit Indefinitheit ausdrücken. *Folgerbarkeit* kann zudem rein syntaktisch (mit polynomiellen Methoden) entschieden und daher effizient implementiert werden.

Da die „Propositionen“ in diesen Beschreibungen zudem nicht nur einfache logische Symbole sind, sondern COMMON LISP-Literale mit einer assoziierten „reichhaltigen“ Hintergrundtheorie (z.B. definierten Operationen auf diesen), bietet sich hier auch die Möglichkeit, *Prädikate aus der COMMON LISP-Wirtssprache* z.B. für eine Anfragesprache zu nutzen. Wir kommen auf diese Idee in Kap. 5.3.1 zurück.

Zusätzlich werden noch *DL-Beschreibungen* bzw. Beschreibungsinstanzen der Klasse DL-Beschreibung (DL-Description) geboten, wobei es sich nicht um einfache Beschreibungen handelt. Es handelt sich hier um Konzeptterme (Klasse Konzept-Beschreibung (Concept-Description)) oder Rollenterme (Klasse Rollen-Beschreibung (Role-Description)) einer DL. Zur Nutzung konkreter DL-Systeme existieren Unterklassen (etwa RACERPRO-Beschreibung (RACERPRO-Description)).

Die Inferenzmethoden zur Prüfung von Erfüllbarkeit und Folgerbarkeit von Be-

5.1 Realisierung des Substrat-Datenmodells

schreibungsinstanzen einer konkreten Unterklasse der Klasse Konzept-Beschreibung (Concept-Description) können stets an das verwendete DL-System delegieren. Beispielsweise kann implies-p bzw. Folgerbarkeit für Paare $C \times D$ vom Typ Konzept-Beschreibung (Concept-Description) vom DL-System durch einen Subsumptionstest $C \sqsubseteq D$ entschieden werden. Analoges gilt für Paare $R \times S$ vom Typ Rollen-Beschreibung (Role-Description): Hier gilt $R \sqsubseteq S$ gdw. $\exists R.T \sqsubseteq \exists S.T$.

Um nun explizit zwischen Knoten- und Kanten-Beschreibungen auf *Typebene* unterscheiden zu können, wird eine weitere Basis-Hierarchie für Knoten- und Kanten-Beschreibungen etabliert (Klasse Knoten-Beschreibung (Node-Description) und Klasse Kanten-Beschreibung (Edge-Description)), und es werden wieder Produktklassen (diesmal mit den Beschreibungsklassen) gebildet. Es sei im Folgenden angenommen, dass alle Produktklassen existieren, auch wenn nicht alle im obigen UML-Diagramm dargestellt sind (so existiert beispielsweise Einfache-KNF-Knoten-Beschreibung (Simple-CNF-Node-Description) als nicht dargestellte gemeinsame Unterklassen der Klassen Einfache-KNF-Beschreibung (Simple-CNF-Description) und Knoten-Beschreibung (Node-Description), etc.).

5.1.2.6. Standard-Substratklassen

Im Folgenden stellen wir die bereitgestellten Standard-Substratklassen vor.

Die meisten sind als Unterklassen von HS-Substrat (MM-Substrate) ausgeführt. Für alle Substrat-Arten ist jedoch die Perspektive der DB-Substrat (DB-Substrate)-Realisierung gegeben, wodurch Persistenz ermöglicht wird. Wir hatten beschrieben, wie diese physische Darstellungsunabhängigkeit erreicht werden kann.

5.1.2.6.1. Das einfache Substrat Die einfachsten Substrate werden als Instanzen der Klasse Einfache-Beschreibungen-Substrat (Simple-Descriptions-Substrate) bereitgestellt, s. Abb. 5.5. Für $\mathcal{L}_V$ verwenden wir Instanzen der Klasse Einfache-Knoten-Beschreibung (Simple-Node-Description), und für $\mathcal{L}_E$ Instanzen der Klasse Einfache-Kanten-Beschreibung (Simple-Edge-Description).

5.1.2.6.2. Das Daten-Substrat Das Daten-Substrat (Data-Substrate) wird von der Klasse Einfache-Beschreibungen-Substrat (Simple-Descriptions-Substrate) abgeleitet, s. Abb. 5.5. Zudem werden die Beschreibungen auf einfache KNF-Beschreibungen eingeschränkt. Somit wird $\mathcal{L}_{SKNF}$ für $\mathcal{L}_V$ und $\mathcal{L}_E$ verwendet, bzw. als Standard-Beschreibungsklassen werden die Klassen Einfache-KNF-Knoten-Beschreibung (Simple-CNF-Node-Description) und Einfache-KNF-Kanten-Beschreibung (Simple-CNF-Edge-Description) ausgezeichnet.

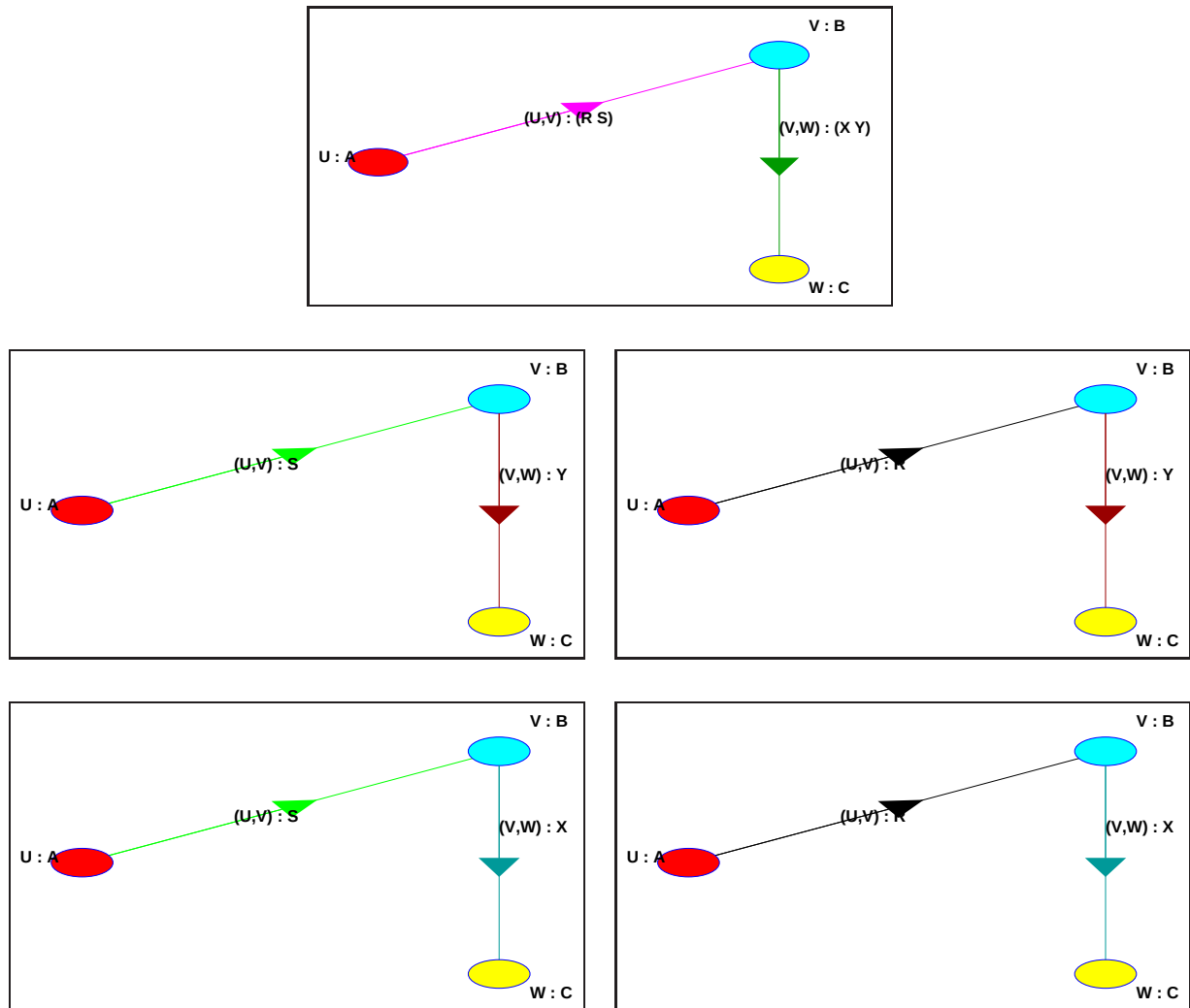


Abbildung 5.4.: Ein disjunktives Kanten-Substrat und seine Konfigurationen.

Dieses Substrat ist für viele Repräsentationsaufgaben gut geeignet. Beispielsweise können RDF-Graphen repräsentiert werden, wenn die XML-Schema-Literale auf COMMON LISP-Literale abgebildet werden. Im Gegensatz zu RDF werden auch Disjunktionen geboten.

5.1.2.6.3. Das disjunktive Kanten-Substrat Wir hatten in Def. 62, S. 197 das RCC-Substrat definiert. Dieses verwendet disjunktive Kanten-Beschreibungen, nämlich Mengen von RCC-Basis-Relationen zur Kanten-Annotation. Wir können also Beschreibungsinstanzen der Klasse Einfache-Disjunktive-Beschreibung (Simple-Disjunctive-Description) für diesen Zweck verwenden. Disjunktive Kanten werden nicht nur vom RCC-Substrat benötigt, sondern beispielsweise auch zur Implementation von ABoxen mit disjunktiven Rollen-Assertionen und Bewei-

5.1 Realisierung des Substrat-Datenmodells

sen für die $\mathcal{ALCI}_{RCC}$ -Familie (s. Kap. 6). Da die *Struktur der Disjunkte* einer Beschreibungsinstanz der Klasse Einfache-Disjunktive-Beschreibung (Simple-Disjunctive-Description) nicht interpretiert wird, können Disjunkte auch durch *Rollen* (der $\mathcal{ALCI}_{RCC}$ -DL) gegeben sein. Dies motiviert die Ableitung der Klasse Disjunktives-Kanten-Substrat (Disjunctive-Edges-Substrate) von der Klasse Einfache-Beschreibungen-Substrat (Simple-Descriptions-Substrate), s. Abb. 5.5, in der wir entsprechende Funktionalität herausfaktorisieren. Als Standardklasse für Kanten-Beschreibungen zeichnen wir die Klasse Einfache-Disjunktive-Kanten-Beschreibung (Simple-Disjunctive-Edge-Description) aus.

Die Klasse Disjunktives-Kanten-Substrat (Disjunctive-Edges-Substrate) soll zudem *zusätzliche Funktionalität* bieten. Wir wollen die *Konfigurationen* eines solchen Substrates enumerieren können. Eine Konfiguration ist analog wie für RCC-Netze in Def. 54, S. 141 definiert; Abb. 5.4 illustriert die möglichen Konfigurationen eines Substrates mit disjunktiven Kanten. Offensichtlich kann diese Funktionalität zur Implementation eines RCC-Konsistenzprüfers genutzt werden. Die Funktionalität wird auch für einen ABox-Erfüllbarkeitstester mit disjunktiven Rollen-Assertionen benötigt.

Beim Entwurf der Enumerations-Funktionalität ist zu berücksichtigen, dass die Anzahl der Konfigurationen exponentiell ist. Eine Methode, die alle Konfigurationen zurückgeben würde, müsste also zunächst eine exponentielle Anzahl neuer Substrate konstruieren und diese dann in einem Container (beispielsweise einer Liste) zurückliefern. Der Speicherbedarf wäre sehr hoch.

Wir sehen daher zusätzlich zur Methode `get-all-configurations` (die wie soeben beschrieben vorgeht) auch noch eine iteratorbasierte Methode zur *inkrementellen Enumeration von Konfigurationen* vor. Hier wird eine Konfiguration durch destruktive Modifikation des aktuellen Zustands des Substrates erzeugt. Mittels `get-next-configuration` kann anhand der aktuellen Konfiguration die nächste Konfiguration angefordert werden, sofern diese existiert, was mittels `has-next-configuration-p` geprüft werden kann. Somit werden dann keine Kopien des Substrat-Graphen notwendig. Prinzipiell kann von der aktuellen Konfiguration natürlich jederzeit eine tiefe Kopie erzeugt werden (s. Methode `copy` in Abb. 5.2), wodurch die aktuelle Konfiguration dauerhaft gemacht würde.

Die destruktive Modifikation von Kantenbeschriftungen von Kanten wird durch die `set-description`-Methode ermöglicht (s. Abb. 5.2). Das verwendete Entwurfsmuster hat Ähnlichkeit mit dem *Fliegengewicht-Muster* [Gamma u. a. 1996].

Um anhand der aktuellen Konfiguration die nächste noch offene Konfiguration herzustellen, muss im Substrat der aktuelle Enumerationszustand bzw. die Position im Oder-Baum, der die Wahlmöglichkeiten für die Disjunkte repräsentiert, verwaltet werden (s. Attribut `enumeration-state`).

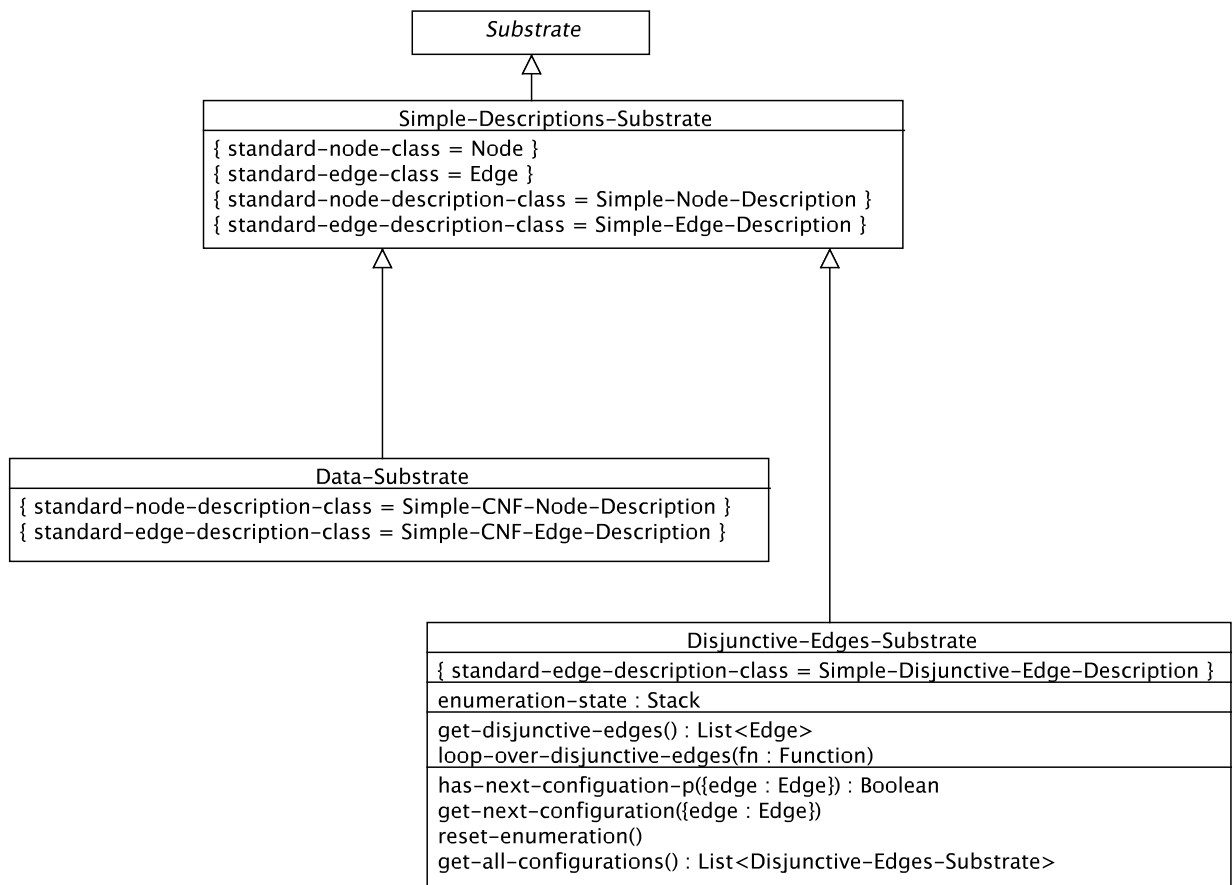


Abbildung 5.5.: Das einfache Substrat, das disjunktive Kanten-Substrat, und das Daten-Substrat.

Oftmals soll nicht eine beliebige nächste Konfiguration erzeugt werden, sondern eine, in der eine vorgegebene Kante ihre Beschreibung wechselt bzw. ihr nächstes Disjunkt annimmt. Hierdurch kann die *Enumerationsreihenfolge der Konfigurationen* beeinflusst werden. Vorweggreifend erwähnen wir hier, dass genau diese Funktionalität zur Implementation des *dependenzgesteuerten Rücksetzens* (*Dependency Directed Backtrackings*) für ABoxen mit disjunktiven Rollen-Assertionen benötigt wird (s. Kap. 6.4.4). Den Methoden `get-next-configuration`, `has-next-configuration-p` können daher optionale Kanten-Argumente übergeben werden (s. optionales Argument `edge` in `get-next-configuration`).

5.1.2.6.4. Das RBox-Substrat Das RBox-Substrat (RBox-Substrate), s. Abb. 5.7, ist ein spezielles Disjunktives-Kanten-Substrat (Disjunctive-Edges-Substrate) und stellt herausfaktorierte Basis-Funktionalität für eine Reihe spezialisierterer Substrate bereit.

5.1 Realisierung des Substrat-Datenmodells

Als Kanten-Beschreibungssprache werden hier spezielle Beschreibungsinstanzen der Klasse Einfache-DNF-Beschreibung (Simple-DNF-Description) verwendet, in denen lediglich COMMON LISP-Symbole als atomare Literale erlaubt sind. Die *literal*-Regel in Def. 92 wird daher gegen $literal ::= symbol\_literal$ ersetzt. Diese Symbole werden als binäre Relationsnamen interpretiert; wir sparen uns die Nennung der entsprechenden Beschreibungsklasse. Es können daher Disjunktionen von (Konjunktionen von) Relationen (bzw. „Rollen“) formuliert werden.

Ein RBox-Substrat (RBox-Substrate) verfügt zudem über eine assoziierte sog. *Relationsbox* (RBox) (s. Klasse RBox), die es ermöglicht, die Eigenschaften dieser binären Relationen zu definieren. Relationen können als *reflexiv*, *symmetrisch*, *transitiv* und als *zueinander invers* deklariert werden. Die RBox enthält also entspr. „Relations- bzw. Rollen-Axiome“ zur Definition dieser Eigenschaften und Beziehungen. Die RBox ermöglicht auch die Spezifikation von Kompositionensaxiomen der Form $S \circ T \sqsubseteq R_1 \sqcup \dots \sqcup R_n$, die u.a. für das qualitative räumliche Schließen und zur Implementation der $\mathcal{ALCT}_{\mathcal{RCC}}$ -Familie benötigt werden (s. Kap. 3.1.5).

Zur Definition von RBoxen wird eine COMMON LISP-Makrosprache bereitgestellt. Das Makro `with-rbox` erlaubt die Definition einer RBox und etabliert einen Kontext für ein zu erzeugendes RBox-Substrat (RBox-Substrate), zum Beispiel:

```
(with-rbox (test-rbox
            :relations (r s t)
            :reflexive-relations (t)
            :inverse-relations ((r s) (s r))
            :axioms
              ((r r (s))
               (r s (s))))

(with-substrate (test :type 'rbox-substrate)
  (node a (node-description a))
  (node b (node-description b))
  (node c (node-description c))
  (node d (node-description d))
  (edge a b (edge-description r))
  (edge b c (edge-description r))
  (edge c d (edge-description r)))
```

Hier wird eine Relationsbox namens `test-rbox` mit den Relationen R, S, T definiert. T ist reflexiv, R und S sind zueinander invers, und es gelten die Kompositionensaxiome $R \circ R \sqsubseteq S$ und $R \circ S \sqsubseteq S$. Alsdann wird ein RBox-Substrat namens `test` im Kontext der frisch aufgesetzten RBox `test-rbox` erzeugt. Diese RBox ist daher die *assoziierte RBox* von `test` (so wie auch eine ABox eine assoziierte TBox hat).

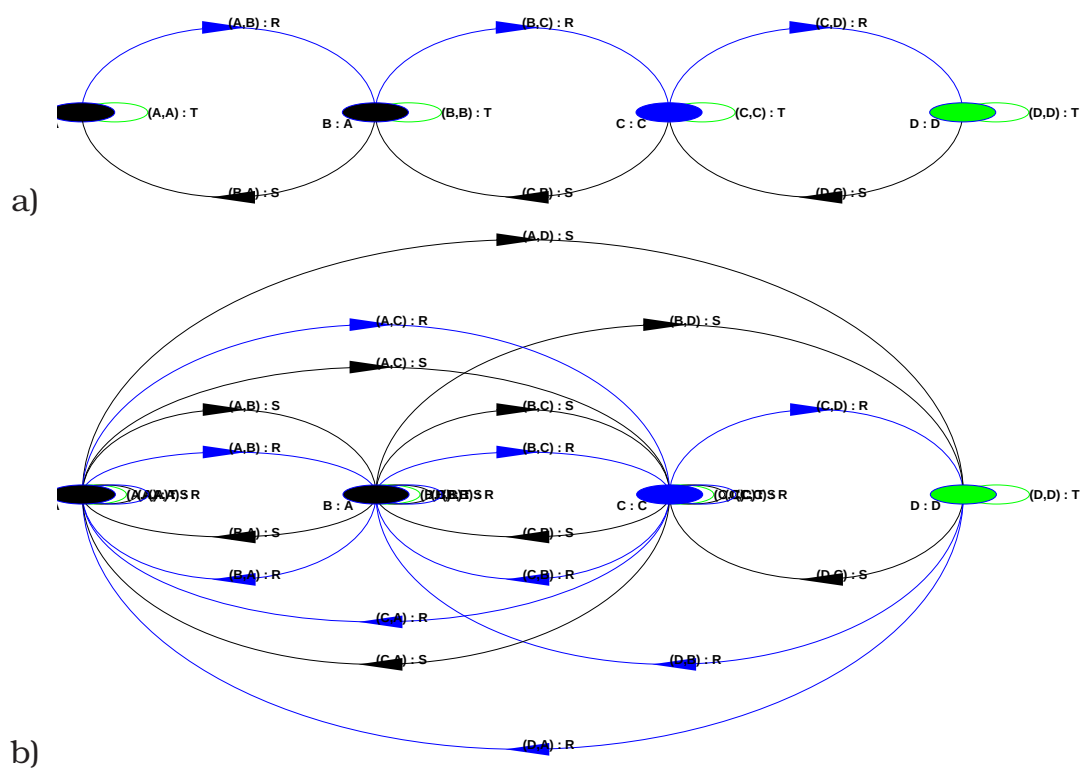


Abbildung 5.6.: Ein RBox-Substrat a) und seine Vervollständigung b).

5.1 Realisierung des Substrat-Datenmodells

Ein RBox-Substrat (RBox-Substrate) bietet eine Menge von Zusatzdiensten. Insbesondere können die implizit durch die Relationsaxiome erzwungen bzw. implizierten Kanten mittels der Methode `complete-substrate` explizit gemacht werden. Abb. 5.6b) zeigt die mittels `complete-substrate` erzeugte *Vervollständigung* des RBox-Substrates aus Abb. 5.6a). Wir sehen eine Methode `complete-p` vor, die überprüft, ob die anhand der RBox-Axiom erzwungenen Kanten bereits explizit vorhanden sind.

Es ist offensichtlich, wie ein solcher Vervollständigungsalgorithmus arbeitet (durch Konstruktion eines Fixpunktes mittels einer Iteration). Sei ein RBox-Substrat mit einer einfachen R -Kante $R(a, b)$ gegeben. Ist R nun als reflexiv definiert, und gilt auch noch $R \circ R \sqsubseteq S$, so wird der Algorithmus zunächst $R(a, a)$ und $R(b, b)$ sicherstellen, und schließlich auch $S(a, b)$ aufgrund von $R(a, a) \circ R(a, b) \sqsubseteq S(a, b)$.

Speziell zur Ermittlung der Kanten, welche an einer Komposition beteiligt sind, wird eine Methode `get-support` bereitgestellt; für das Beispiel liefert `get-support(a, b) = \{(R(a, a), R(a, b))\}`. Die Methode liefert also die Menge der Kantenpaare, deren Beschreibungen komponiert werden müssen. Pro Kantenpaar $(R(x, y), S(y, z))$ in `get-support(x, z)` wird *eine neue Kante* $T(x, z)$ angelegt, wenn $R \circ S \sqsubseteq T$ in der RBox ist.

Auch hier muss für die vom Vervollständigungsalgorithmus zu erzeugenden neuen Instanzen das *Objekterzeugungsproblem* gelöst werden; die Klassen der zu erzeugenden Instanzen dürfen im Code nicht literal genannt werden. Dies Problem wird ebenfalls durch die vom Substrat ausgezeichneten Standardklassen gelöst. Alternativ ermöglichen wir jedoch auch, dass ein anonymer *Kantenkonstruktor* als Funktionsobjekt als optionales Argument an `complete-substrate` mitübergeben werden kann.

5.1.2.6.5. Das JEPD-Substrat Das JEPD-Substrat (JEPD-Substrate), s. Abb. 5.7, ist ein spezielles RBox-Substrat (RBox-Substrate), in dem die *JEPD-Eigenschaft* (s. Kap. 3.1.1.1.1) erzwungen wird.

Als Konsequenz hieraus kann ein JEPD-Substrat (JEPD-Substrate) nun inkonsistent bzw. unerfüllbar werden (die bisher betrachteten Substrate waren stets erfüllbar). Zudem wird $\mathcal{L}_\varepsilon$ nun noch einmal weiter eingeschränkt, denn es dürfen nun keine Konjunktionen mehr als Disjunkte auftreten. Also werden lediglich einzelne COMMON LISP-Symbole oder Listen von einfachen COMMON LISP-Symbolen zugelassen; diese stellen dann Basis-Relationen bzw. Disjunktionen von Basis-Relationen dar. Wir sparen uns die Nennung der entspr. Beschreibungsklassen.

Wie in Kap. 3.1.1.1.1 beschrieben, muss in einem JEPD-Substrat nun zwischen je zwei Knoten *genau eine* Kante verlaufen. *Vervollständigte und kon-*

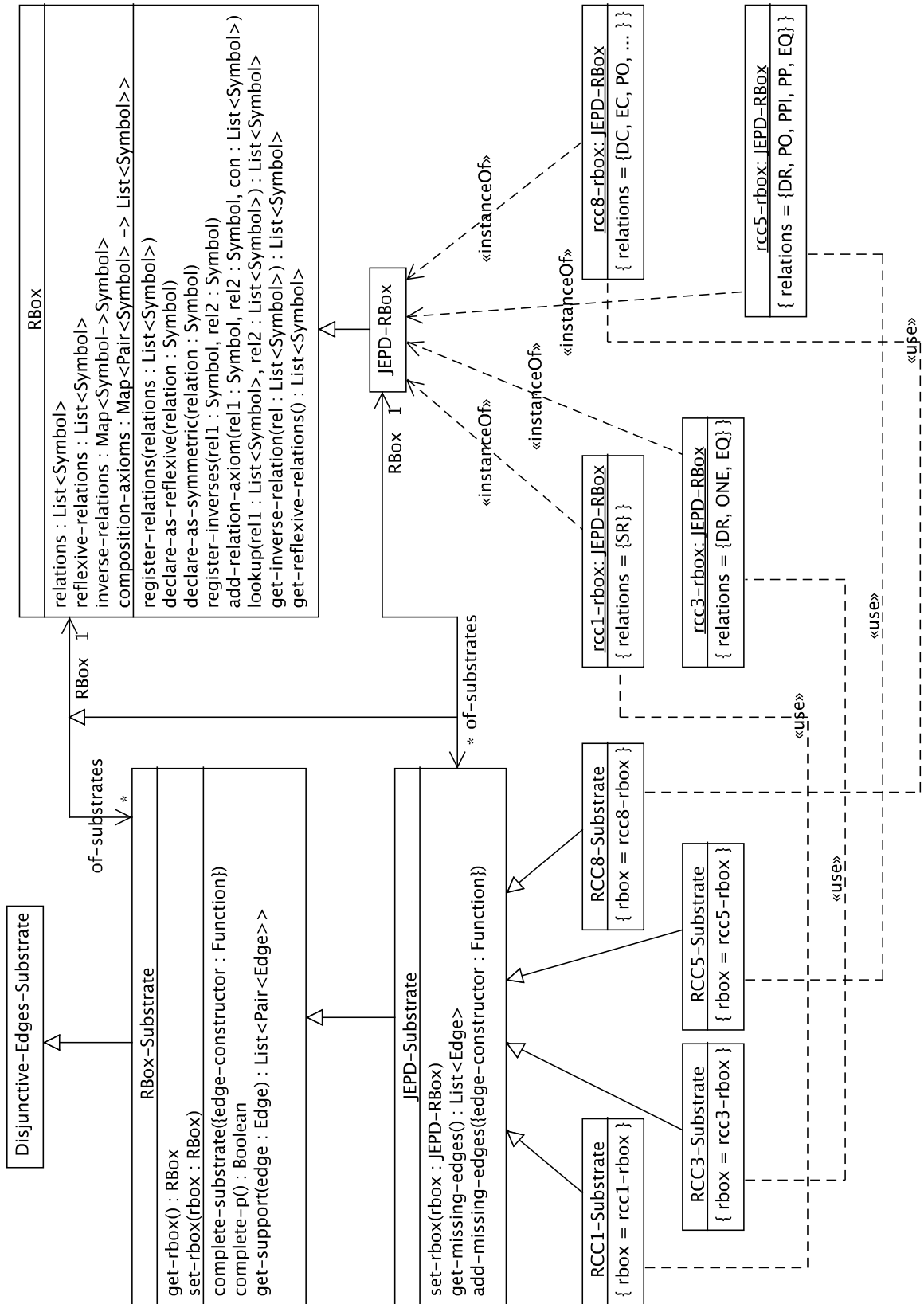


Abbildung 5.7.: Die RBox-Substrate.

5.1 Realisierung des Substrat-Datenmodells

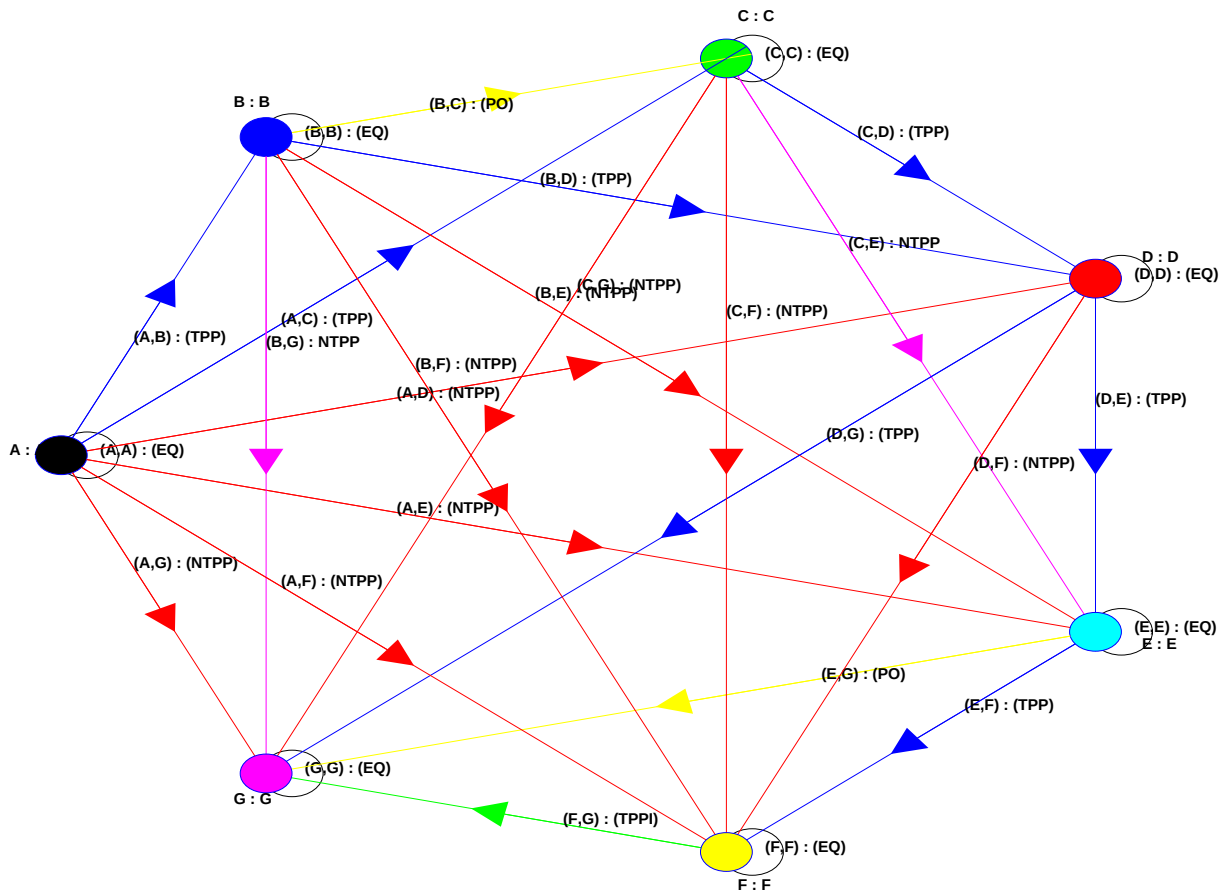


Abbildung 5.8.: Ein RCC8-Substrat.

sistente JEPD-Substrate haben die Form eines K_n -Graphen, s. Abb. 5.8. Aufgrund der JEPD-Eigenschaft ergeben sich bereits für die Rollenbox eines JEPD-Substrates gewisse Einschränkungen. So darf z.B. lediglich eine reflexive Relation vorhanden sein, und für zwei Relationen R, S darf lediglich ein Axiom der Form $R \circ S \sqsubseteq \dots$ existieren, etc. Eine Unterklasse der Klasse RBox namens JEPD-RBox erzwingt die syntaktisch prüfbaren Bedingungen.

Zur Berechnung der Vervollständigung eines JEPD-Substrates ist es hier zweckreich, zunächst die *fehlenden* Kanten zu identifizieren (Methode `get-missing-edges`). Aufgrund der JEPD-Eigenschaft ist eine fehlende Kante äquivalent zu einer vollständig disjunktiven Kante (die Disjunktion aller Relationsnamen). Diese Kanten werden daher in das Substrat eingetragen (Methode `add-missing-edges`). Nun verläuft zwischen je zwei Knoten a und b genau eine Kante.

Die ererbte Vervollständigungsmethode `complete-substrate` kann zur Implementation eines (unvollständigen) Verfahrens zur Prüfung auf relationale

Konsistenz verwendet werden (Methode *consistent-p*). Wir hatten in Def. 53 beschrieben, dass dieser Algorithmus durch Constraintpropagierung, also durch die sukzessive Einschränkung der Kantenbeschriftungen mittels $L_E((x, y)) := L_E((x, y)) \cap (\bigcup_{R \in \mathcal{R}, S \in \mathcal{S}} \text{comp}(R, S))$ in einer Fixpunktiteration arbeitet. Die Iteration terminiert, wenn keine Kantenbeschriftung mehr eingeschränkt werden kann, oder aber eine Kante mit der leeren Menge beschriftet wurde. In letzterem Fall ist das Substrat unerfüllbar. Wir implementieren diesen Algorithmus unter Verwendung von *complete-substrate*, indem wir einen speziellen *Kantenkonstruktor* als Funktionsobjekt übergeben. Dieser „Konstruktor“ erzeugt nun keine neuen Kanten, sondern modifiziert einfach die zwischen x und y bestehende Kanten-Beschreibungsinstanz destruktiv, indem er $L_E((x, y)) := L_E((x, y)) \cap (\bigcup_{R \in \mathcal{R}, S \in \mathcal{S}} \text{comp}(R, S))$ setzt.

5.1.2.6.6. Die RCC-Substrate Diese werden von der Klasse JEPD-Substrat (JEPD-Substrate) abgeleitet, s. Abb. 5.7. Es werden lediglich die assoziierten RBoxen auf die RBoxen für die RCC-Systeme RCC1, RCC3, RCC5 und RCC8 festgelegt.

Die ererbte Funktionalität ermöglicht bereits, RCC-Netze auf relationale Konsistenz zu prüfen, ihre Konfigurationen zu enumerieren, und sie zu vervollständigen. Diese Funktionalität wird später an mehreren Stellen Anwendung finden (u.a. zur Implementierung des $\mathcal{ALCT}_{RCC}$ -Beweisers und zur Entscheidung von *Anfragesubsumption* für DISKQL-Anfragen).

5.1.2.6.7. Beschreibungslogik-Substrate Wir haben das Substrat-Datenmodell nicht zuletzt auch deshalb definiert, um die Dienste existierender (externer) DL-Systeme durch die Bereitstellung einer einheitlichen Schnittstelle für unsere Zwecke nutzbar zu machen (Perspektive der semantischen Middleware und losen Kopplung). Wir definieren daher eine Reihe von *DL-Substraten* in unserem Rahmenwerk:

1. Alle DL-Substrate sind Unterklassen der abstrakten Klasse DL-Substrat (DL-Substrate), welche wiederum von Substrat (Substrate) abgeleitet ist.
2. Einige DL-Systeme bieten ABoxen. Wir leiten eine abstrakte Unterklasse ABox-Substrat (ABox-Substrate) ab. Es wird eine „Substrat-Sicht“ auf eine ABox geboten, vgl. Prop. 2 auf S. 197.
3. Wird die ABox, auf die die Substrat-Sicht hergestellt wird, durch ein *externes DL-System* verwaltet, so müssen die ABox-Repräsentations- und -Inferenzdienste des DL-Systems per Delegation in Anspruch genommen werden. Handelt es sich zudem um ein *entferntes* DL-System, wie beispielsweise einen RACERPRO-Server, so müssen die Dienste per entferntem Prozedur- bzw. Methodenaufruf in Anspruch genommen werden. Der

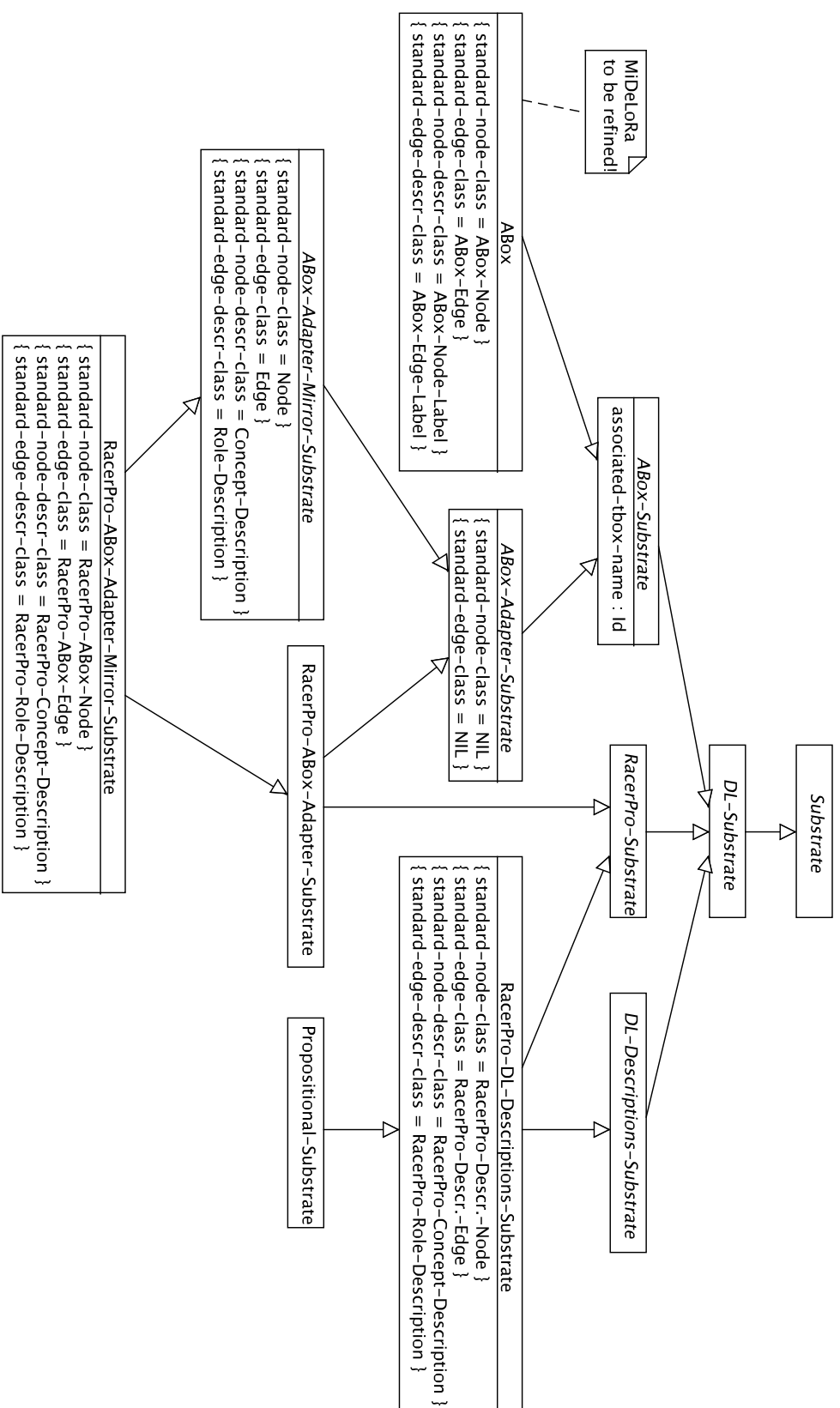


Abbildung 5.9.: Die DL-Substrate.

Substrat-Graph bzw. die ABox wird also primär durch das externe DL-System verwaltet, und der Substrat-Graph ist eine bei Bedarf berechnete Perspektive, die lediglich durch die Schnittstellenmethoden der Klasse Substrat (Substrate) gegeben ist. So führt beispielsweise das Anlegen eines Substrat-Knotens mittels `create-node` zur Erzeugung eines ABox-Individuums im externen DL-System, und der Aufruf `get-nodes` wird an die API-Funktion `get-abox-individuals` des DL-Systems delegiert.

Das ABox-Adapter-Substrat (ABox-Adapter-Substrate) ist als Unterklasse von ABox-Substrat (ABox-Substrate) vorgesehen und ermöglichte diese „virtuelle, berechnete Substrat-Sicht“ auf die externe ABox (s. auch S. 244). Das Substrat ist also virtuell, und dementsprechend ist kein Objektgraph vorhanden. Das hier verwendete Entwurfsmuster kann als *Adapter-Strukturmuster* [Gamma u. a. 1996] klassifiziert werden.⁶

Das ABox-Adapter-Substrat (ABox-Adapter-Substrate) ist eine abstrakte Klasse. Für konkrete DL-Systeme werden instantiierbare Unterklassen vorgesehen (z.B. RACERPRO-ABox-Adapter-Substrat (RACERPRO-ABox-Adapter-Substrate)).

4. Oftmals ist es dennoch wünschenswert, *lokale Proxy-Objekte* der soeben diskutierten virtuellen Knoten- und Kanten des Substrat-Graphen zu erzeugen, wie bereits in Kap. 5.1.2.1 diskutiert. Die abstrakte Unterklasse ABox-Adapter-Spiegel-Substrat (ABox-Adapter-Mirror-Substrate) der Klasse ABox-Adapter-Substrat (ABox-Adapter-Substrate) ermöglicht dies. Eine Substrat-Instanz dieser Klasse stellt also eine Art „Spiegel“ einer externen ABox dar, der initial beim Anlegen des Substrates erzeugt wird. Der Spiegel muss dann mit der externen ABox synchronisiert bzw. konsistent gehalten; entspr. Techniken sind bekannt (u.a. „Write Through“ und „Publish Subscribe“).

Der Objektgraph hat zum einen die Funktion eines lokalen Caches. Im Falle eines entfernten DL-Systems kann Netzwerk-Kommunikation eingespart werden. Zum anderen können die Knoten- und Kanten-Klassen anwendungsspezifisch spezialisiert werden, sodass deren Instanzen beliebige Zusatzinformation mit ABox-Objekten assoziieren können.

Zur RACERPRO-Nutzung wird eine konkrete Unterklasse RACERPRO-ABox-Adapter-Spiegel-Substrat (RACERPRO-ABox-Adapter-Mirror-Substrate) vorgesehen.

5. Einige DL-Systeme bieten keine ABox (z.B. das FACT-System). Für diese können wir eine Art „Ersatz-ABox“ schaffen, indem wir Knoten- und Kanten einfach mit Beschreibungsinstanzen der Klasse DL-Beschreibung (DL-Description) annotieren. Diese abstrakte Substratklasse heißt DL-Beschrei-

⁶ Die Methoden `get-standard-node-class`, `get-standard-edge-class`, `get-standard-nodedescription-class`, `get-standard-edge-description-class` liefern stets NIL.

5.1 Realisierung des Substrat-Datenmodells

bungs-Substrat (DL-Description-Substrate); es handelt sich jedoch nicht um eine Unterklasse von ABox-Substrat (ABox-Substrate).

Obwohl bei RACERPRO-Nutzung primär das RACERPRO-ABox-Adapter-Substrat (RACERPRO-ABox-Adapter-Substrate) Verwendung finden wird (da RACERPRO ein ABox-DL-System ist), sehen wir auch die Klasse RACERPRO-DL-Beschreibungs-Substrat (RACERPRO-DL-Description-Substrate) vor. Diese Klasse kann beispielsweise zur Implementierung eines *Instance Stores* [Horrocks u. a. 2004b] Anwendung finden.

6. Die Klasse Aussagenlogisches-Substrat (Propositional-Substrate) (s. Def. 59, S. 196) kann nun als konkrete Unterklasse der Klasse RACERPRO-ABox-Adapter-Spiegel-Substrat (RACERPRO-ABox-Adapter-Mirror-Substrate) bereitgestellt werden. RACERPRO entscheidet hier die (aussagenlogischen) Inferenzprobleme.
7. Die Klasse ABox (konkrete Unterklasse der Klasse ABox-Substrat (ABox-Substrate)) wird als ABox- bzw. Tableau-Repräsentation für ein eigenes DL-System namens MIDELORA dienen (s. Kap. 6).

Die unterschiedlichen Substratklassen sind im UML-Diagramm in Abb. 5.9 dargestellt. Diese Klassen realisieren eine hohe Flexibilität zur Nutzung der Repräsentations- und Inferenz-Dienste von DL-Systemen für ontologiebasierte ISe für die extensionalen Komponenten. Aufgrund der uniformen Substrat-Schnittstelle können verschiedene Repräsentationsmöglichkeiten für die extensionalen Komponenten verwendet und einfach variiert werden, ohne dass programmtechnische Anpassungen in der Implementation des ISs erforderlich wären (da das IS gegen die Substrat-Schnittstelle implementiert wurde). Wir illustrieren dies später am Beispiel des DLMAPS-Systems in Kap. 5.1.3.

5.1.2.6.8. Hybride Substrate Die Klasse Hybrides-Substrat (Hybrid-Substrate), s. Abb. 5.10, realisiert die Idee des hybriden Substrates. Es handelt sich um eine konkrete Unterklasse der Klasse Substrat (Substrate). Wie in Def. 67, S. 200 dargestellt, aggregiert ein hybrides Substrat letztlich zwei Substrate S_1 und S_2 . Beim Anlegen eines hybriden Substrates werden zwei Komponenten-Substrate S_1 , S_2 der Klassen erzeugt, die durch die Klassenmethoden `get-standard-substrate-class-for-s1`, `get-standard-substrate-class-for-s2` als Standard-Substratklassen für S_1 und S_2 ausgezeichnet werden.

Das hybride Substrat verwaltet selbst *keinen* Substrat-Graphen, sondern überlässt die Verwaltung den Komponenten-Substraten S_1 und S_2 . Dementsprechend werden alle Methoden-Aufrufe für die in der Klasse Substrat (Substrate) definierten Schnittstellen-Operationen vom hybriden Substrat stets an das Substrat S_1 delegiert. Die Schnittstelle kann jedoch auf S_2 „umgeschaltet“ werden, indem S_1 die Rolle von S_2 , und S_2 die Rolle von S_1 annimmt (s.

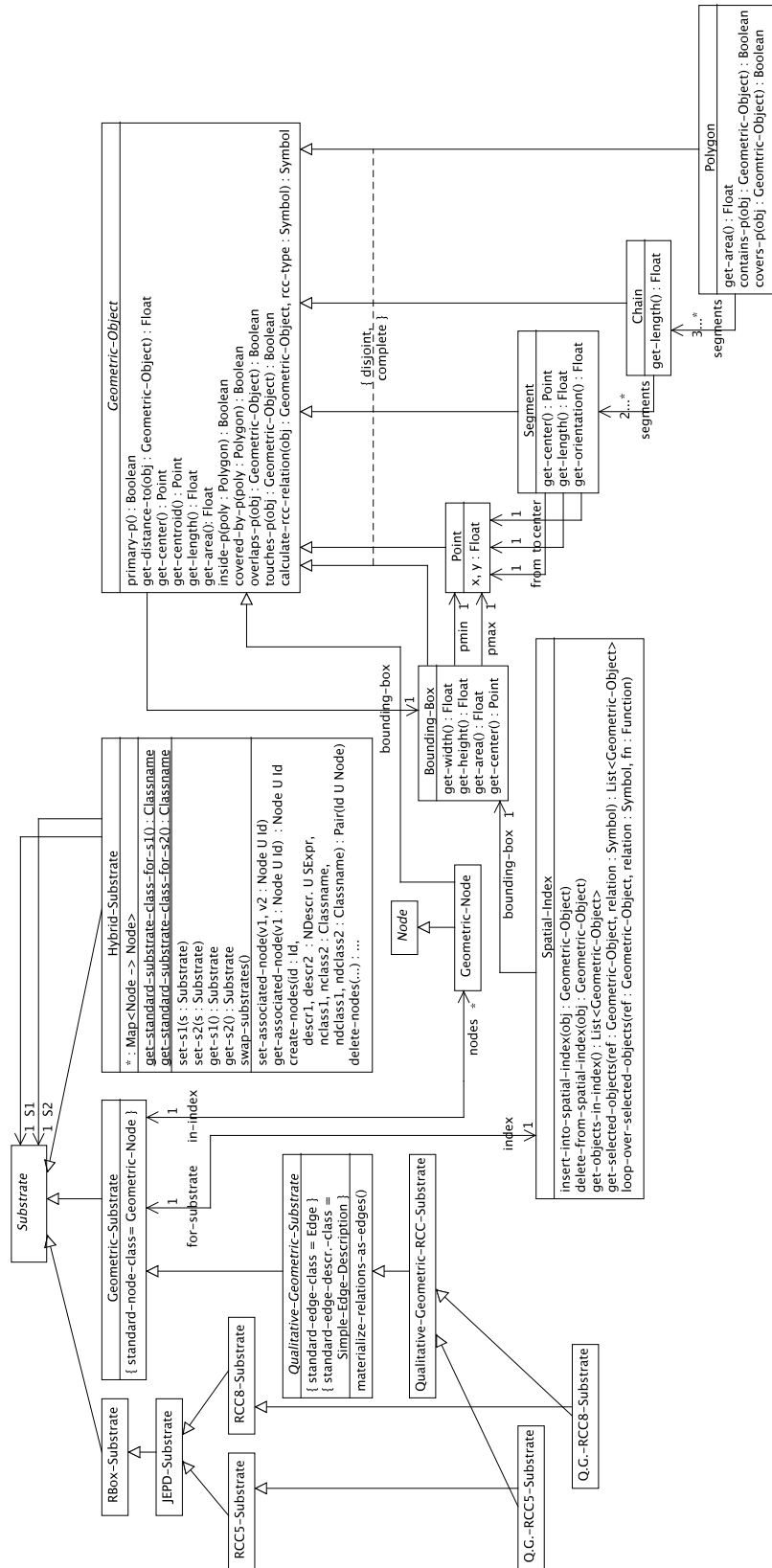


Abbildung 5.10.: Das hybride Substrat und die geometrischen Substrate.

5.1 Realisierung des Substrat-Datenmodells

Methode `swap-substrates`). Ein hybrides Substrat ist damit (je nach Zustand) entweder für S_1 oder für S_2 (typ-)substituierbar. Zur Assoziation eines Knotens in S_1 mit einem Knoten in S_2 über die $*$ -Funktion ist die Methode `set-associated-node` vorgesehen; der assoziierte Knoten eines Knotens in V_1 kann mittels `get-associated-node` erhalten werden.

Das hybride Substrat stellt zudem erweiterte Konstruktoren und Destruktoren zur Verfügung; so kann mittels der `create-nodes`-Methode automatisch ein Paar korrespondierender Knoten gleichen Namens, die mit entspr. Beschreibungsinstanzen annotiert sind, erzeugt werden; `delete-nodes` kann dieses Paar wieder löschen.

Für Anwendungen ist es sinnvoll, anwendungsspezifische Unterklassen der Klasse `Hybrides-Substrat` (`Hybrid-Substrate`) zu bilden, für die die Klassenmethoden `get-standard-substrate-class-for-s1` und `get-standard-substrate-class-for-s2` anwendungsspezifisch zur Festlegung geeigneter Komponenten-Substrate überschrieben werden. Wiederum können hier Produktklassen gebildet werden, für die keine Implementierungsarbeit erforderlich ist. Flexibles und orthogonales Kombinieren wird so ermöglicht.

5.1.2.6.9. Geometrisches Substrat Die konkrete Klasse `Geometrisches-Substrat` (`Geometric-Substrate`), s. Def. 63, S. 198 und Abb. 5.10, bietet eine Knoten-Beschreibungssprache $\mathcal{L}_{V_R}$ zur Beschreibung von Instanzen räumlicher Datentypen (SDTs). Die konkrete Syntax der in Def. 63 auf S. 198 angegebenen Geometrie-Beschreibungsterme sei durch folgendes Beispiel illustriert:

```
(node p1 (node-description (point 0 0)))
(node p2 (node-description (point 5 5)))
(node s1 (node-description (segment p1 (point 10 0))))
(node s2 (node-description (segment (point 10 0) p2)))
(node poly1 (node-description (polygon s1 s2 (segment p2 p1))))
```

Hier wird ein dreieckiges Polygon (sowie seine Komponenten) konstruiert. In dem Namen von bereits konstruierten Objekten in den Geometrie-Beschreibungen referenziert werden (wie `p1`), können bereits konstruierte Knoten im Substrat referenziert werden.

Auch die Komponentenobjekte (z.B. die Punkte eines Segments) werden als Knoten im Substrat repräsentiert. Diese sind jedoch „anonym“, d.h. nicht benannt. Die anonyme Knoten können mittels der `primary-p`-Methode von den benannten Knoten unterschieden werden. Die kompositionale Struktur kann zudem mittels einer `has-part`-Kante im Substrat explizit gemacht werden.

Das geometrische Substrat kann in einem Modus betrieben werden, in dem räumliche bzw. „topologische“ Konsistenzbedingungen erzwungen werden. In

diesem Modus werden geometrische Objekte nicht als individuelle Objekte, sondern als *Werte* betrachtet. Ein Punktkonstruktoraufruf erzeugt dann nicht zwingenderweise einen neuen Punkt-Knoten, sondern gibt u.U. ein bereits mit diesen Koordinaten bestehenden Punkt-Knoten zurück (zu dessen effizienter Ermittlung ein räumlicher Index verwendet wird).

Das geometrische Substrat bietet eine umfangreiche Bibliothek an *geometrischen Basis-Funktionen*, z.B. Methoden zur Berechnung der Fläche eines Polygons, Berechnung der Länge eines Segments oder einer Kette, Berechnung von Winkeln und Abständen, etc. Das Substrat bietet auch *Funktionalität zur Berechnung qualitativer räumlicher Relationen*. Es werden u.a. RCC-Relationen geboten. Zur Auswertung bzw. Berechnung dieser qualitativen räumlichen Beziehungen wird ebenfalls der räumliche Index verwendet. Wir können diese Methoden als *Inspektionsmethoden* (s. Kap. 3.1.1.1, S. 135) nutzen.

Räumliche Selektion wird unter Verwendung des räumlichen Indexes mit Hilfe spezieller Iteratoren ermöglicht, den sog. *räumlichen Iteratoren*. Ein solcher Iterator enumeriert anhand eines vorgegebenen Referenzobjektes und einer vorgegebenen vom geometrischen Substrat implementierten (qualitativen oder metrischen) räumlichen Relation all die Objekte, die mit dem angegebenen Referenzobjekt in der geforderten räumlichen Beziehung stehen.

Wir hatten das qualitative geometrische RCC-Substrat (s. Def. 66, S. 200) als Spezialisierung des qualitativen geometrischen Substrates definiert (s. Def. 65, S. 199), welches wiederum eine Spezialisierung des geometrischen Substrates ist. Wir sehen daher in Abb. 5.10 entspr. Klassen vor (Klassen Qualitatives-Geometrisches-Substrat (Qualitative-Geometric-Substrate) und Qualitatives-Geometrisches-RCC-Substrat (Qualitative-Geometric-RCC-Substrate)).

5.1.3. Substrate für die Leitdomänen

Die bisher vorgestellten Substratklassen können als Standard- oder Mehrzweck-Substrate charakterisiert werden. Sie können für eine Vielzahl von Anwendungsszenarien nützliche Dienste leisten.

Zur Realisierung der prototypischen ontologiebasierten ISe in den Leitdomänen definieren wir nun anwendungsspezifische Unterklassen dieser Standard-Substrate und setzen damit die Diskussion aus Kap. 4.1.4, Kap. 3.1.6 und Kap. 3.2.4 auf softwaretechnischer Ebene fort.

5.1.3.1. (Leitdomäne 1) Die Karten-Substrate

Für die GIS-Domäne hatten wir in Kap. 3.1.6 die prinzipiellen Realisierungsmöglichkeiten der extensionalen Komponente besprochen. Den Objektschlüs-

5.1 Realisierung des Substrat-Datenmodells

selkatalog hatten wir als TBox remodelliert, wodurch die intensionale Komponente gegeben ist (die sog. DISK-Ontologie).

Durch Nutzung der bereitgestellten Standard-Substratklassen bieten sich nun eine Vielzahl von Realisierungsmöglichkeiten für die extensionale Komponente. Dies demonstriert auch die Flexibilität des Ansatzes. Alle Karten-Substrate sind von der Klasse Karten-Substrat (Map-Substrate) abgeleitet. Einige dieser Substrate sind hybrid.

Die (durch Beschreibungsinstanzen gegebenen) thematischen Annotationen der Knoten werden anhand der Objektschlüsselannotationen in den DISK-SQD-Daten erzeugt. Die Klasse Karten-Substrat (Map-Substrate) stellt eine Abbildungsmethode `get_descriptor_for_key` bereit, die durch die genutzte Klasse überschrieben bzw. redefiniert wird, sodass anhand des Objektschlüssels i eine Beschreibungsinstanz aus $\mathcal{L}_V$ mittels `get_descriptor_for_key(i)` geliefert wird.

Zur Repräsentation der Geometrie der DISK verwenden wir Instanzen der Klasse Geometrisches-Substrat (Geometric-Substrate). Die folgenden Repräsentationsmöglichkeiten führen nun zu unterschiedlichen Unterklassen vom Karten-Substrat (Map-Substrate):

- Frage 1: Sollen RCC-Relationen explizit als Kanten im geometrischen Substrat eingetragen werden? Diese können als Index dienen.
 - Antwort auf Frage 1 = Ja: Wir spezialisieren die Klasse Qualitatives-Geometrisches-RCC-Substrat (Qualitative-Geometric-RCC-Substrate) zur abstrakten Klasse Explizite-Relationen-Karten-Substrat (Explicit-Relations-Map-Substrate).
 - Antwort auf Frage 1 = Nein: Wir spezialisieren die Klasse Geometrisches-Substrat (Geometric-Substrate) zur abstrakten Klasse Implizite-Relationen-Karten-Substrat (Implicit-Relations-Map-Substrate).
- Frage 2: Soll ein DL-System für die extensionale Komponente verwendet werden?
 - Antwort auf Frage 2 = Ja: Dann stellt sich
 - * Frage 3: Soll eine ABox verwendet werden (falls das verwendete DL-System ABoxen bietet)?
 1. Antwort auf Frage 3 = Ja:

Zur Nutzung der Dienste eines *externen* / *entfernten* Standard-DL-Systemes können wir ein ABox-Adapter-Substrat (ABox-Adapter-Substrate) oder ein ABox-Adapter-Spiegel-Substrat (ABox-Adapter-Mirror-Substrate) verwenden. Alternativ können wir ein *lokales* DL-System wie MIDELORA verwenden. Es ergeben sich also drei Möglichkeiten:

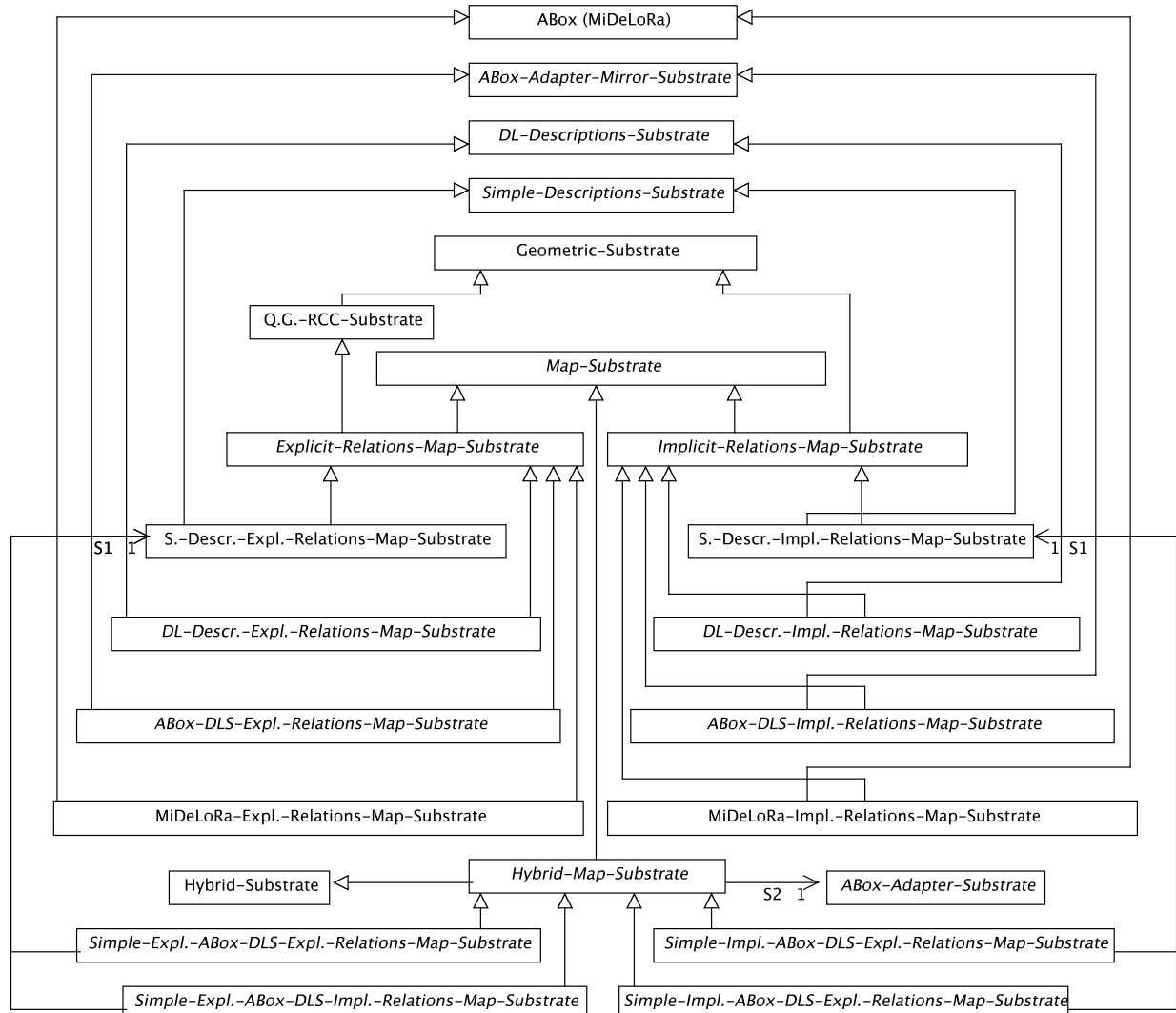


Abbildung 5.11.: Die Karten-Substrate.

- a) Verwenden wir die Klasse ABox-Adapter-Spiegel-Substrat (ABox-Adapter-Mirror-Substrate), so können die lokalen Proxy-Knoten zugleich auch geometrische Objekte sein bzw. mit diesen *identifiziert* werden. Für diesen Zweck bilden wir eine gemeinsame Unterklasse der Klassen Explizite-Relationen-Karten-Substrat (Explicit-Relations-Map-Substrate) (bzw. Implizite-Relationen-Karten-Substrat (Implicit-Relations-Map-Substrate)) und ABox-Adapter-Spiegel-Substrat (ABox-Adapter-Mirror-Substrate) namens ABox-DLS-Explizite-Relationen-Karten-Substrat (ABox-DLS-Explicit-Relations-Map-Substrate) (bzw. ABox-DLS-Implizite-Relationen-Karten-Substrat (ABox-DLS-Implicit-

5.1 Realisierung des Substrat-Datenmodells

Relations-Map-Substrate)).

Als Beschreibungssprachen akzeptieren wir die Vereinigung der Beschreibungssprachen der Oberklassen. Ein Knoten in diesem Substrat ist dann sowohl ein ABox-Individuum, als auch ein geometrisches Objekt. Wir hatten diskutiert, dass eine Instanz der Klasse ABox-Adapter-Spiegel-Substrat (ABox-Adapter-Mirror-Substrate) beliebige Zusatzinformationen mit ABox-Individuen assoziieren kann; in diesem Falle ist diese Zusatzinformation durch die Geometrie gegeben.

- b) Verwenden wir die Klasse ABox-Adapter-Substrat (ABox-Adapter-Substrate), so liegt ein virtuelles Substrat vor, und so können die geometrischen Objekte offensichtlich nicht mit den virtuellen Substrat-Knoten (die ABox-Individuen darstellen) identifiziert werden. Die Assoziation wird dann durch ein hybrides Substrat hergestellt. Wir bilden daher eine entspr. Unterklasse von Hybrides-Substrat (Hybrid-Substrate) namens Hybrides-Karten-Substrat (Hybrid-Map-Substrate), für die wir S_1 auf die Klasse Geometrisches-Substrat (Geometric-Substrate), und S_2 auf die Klasse ABox-Adapter-Substrat (ABox-Adapter-Substrate) festlegen.
- c) Schließlich sehen wir noch vor, das MiDeLoRA-System *lokal* (also integriert) zu nutzen. Wir können für MiDeLoRA eine spezialisierte ABox schaffen, deren Instanzen Knoten enthalten, die sowohl ABox-Individuen als auch geometrische Objekte sind. Wir hatten die Idee dieser „reichhaltigen, räumlichen ABox“ bereits in Kap. 3.1.6 erläutert. Durch Bildung einer gemeinsamen Unterklasse der Klassen ABox und Explizite-Relationen-Karten-Substrat (Explicit-Relations-Map-Substrate) (bzw. Implizite-Relationen-Karten-Substrat (Implicit-Relations-Map-Substrate)) können wir diese Idee realisieren. Wir erhalten so die konkrete Klasse MiDeLoRA-Explizite-Relationen-Karten-Substrat (MiDeLoRA-Explicit-Relations-Map-Substrate) (bzw. MiDeLoRA-Implizite-Relationen-Karten-Substrat (MiDeLoRA-Implicit-Relations-Map-Substrate)).

Wir erreichen hiermit prinzipiell den gleichen Effekt wie unter a), doch kann zum einen auf Netzwerkkommunikation verzichtet werden. Zum anderen hat diese Repräsentation eine Reihe anderer operationaler Vorteile, da MiDeLoRA sich der Räumlichkeit einer solchen ABox „bewusst“ ist und diese nutzen kann. Wir kommen hierauf in Kap. 6 zurück.

- 2. Antwort auf Frage 3 = Nein: Wir verwenden statt einer ABox

eine „Ersatz-ABox“; die Semantik einer solchen Ersatz-ABox kann mit einem aussagenlogischen Substrat verglichen werden, doch sind die erlaubten Knoten- und Kantenbeschriftungen hier ausdrucksstärker, da nicht nur Aussagenlogik, sondern die vom DL-System verwendeten Konzept- und Rollenterme zur Annotation verwendet werden können. Im Vergleich zu einer ABox sind die Inferenzen i.Allg. natürlich unvollständig (aber vollständig, wenn hier lediglich bestimmte Konzept- und Rollenterme zugelassen werden).

Wir können die Knoten im geometrischen Substrat einfach mit Beschreibungsinstanzen der Klasse DL-Beschreibung (DL-Description) annotieren. Dementsprechend bilden wir eine gemeinsame Unterklassen der Klassen DL-Beschreibungs-Substrat (DL-Description-Substrate) und Explizite-Relationen-Karten-Substrat (Explicit-Relations-Map-Substrate) (bzw. Implizite-Relationen-Karten-Substrat (Implicit-Relations-Map-Substrate)). Hieraus resultiert die konkrete Klasse DL-Beschreibungen-Explizite-Relationen-Karten-Substrat (DL-Descriptions-Explicit-Relations-Map-Substrate) (bzw. DL-Beschreibungen-Implizite-Relationen-Karten-Substrat (DL-Descriptions-Implicit-Relations-Map-Substrate)).

- Antwort auf Frage 2 = Nein: Wir verwenden also kein DL-System für die extensionale Komponente.

Wir können dann Beschreibungsinstanzen der Klasse Einfache-Beschreibung (Simple-Description) zur Thematik-Annotation nutzen. Dies führt uns zu den Klassen Einfache-Beschreibungen-Explizite-Relationen-Karten-Substrat (Simple-Descriptions-Explicit-Relations-Map-Substrate) und Einfache-Beschreibungen-Implizite-Relationen-Karten-Substrat (Simple-Descriptions-Implicit-Relations-Map-Substrate).

Eine schwache Form ontologiebasierter Anfragebeantwortung ist immer noch möglich. Hierzu wird die klassifizierte TBox (der remodellierte Objektschlüsselkatalog, die DISK-Ontologie) zur erweiterten Thematik-Annotation der Knoten genutzt. Wir stellen für einen mit dem thematischen Konzept C annotierten Knoten i in diesem Substrat einfach sicher, dass auch die Oberkonzepte und Synonyme von C explizit als Konjunkte in $L_V(i)$ aufgenommen werden. Somit ist eine „Wiese“ nicht nur als „Wiese“, sondern auch als explizite Instanz von „Grünfläche“ und „Fläche“ etc. bekannt. Dies ermöglicht eine schwächere Form der ontologiebasierten Anfragebeantwortung, die wir als thesaurusbasierte oder taxonomische Anfragebeantwortung bezeichnen können.

Von der Klasse Hybrides-Karten-Substrat (Hybrid-Map-Substrate) leiten wir noch wei-

5.1 Realisierung des Substrat-Datenmodells

tere Unterklassen ab, wobei der Diskriminator nun durch die für S_1 und S_2 voreingestellten Substratklassen gegeben ist. Da S_2 stets ein ABox-Adapter-Substrat (ABox-Adapter-Substrate) ist, lassen wir für S_1 nur Unterklassen von Einfache-Beschreibungen-Substrat (Simple-Descriptions-Substrate) zu (eine „hybride ABox“ ist hier nicht sinnvoll).

Als Konsequenz dieser Repräsentationsmöglichkeiten ergeben sich die in Abb. 5.11 dargestellten Klassen. Die meisten Klassen sind abstrakt; es sind also noch konkrete Unterklassen für konkrete DL-Systeme wie RACERPRO etc. abzuleiten. Wir verzichten auf eine Darstellung.

Dem Einwand einer kombinatorischen Explosion in der Anzahl der Klassen sei mit dem Argument begegnet, dass diese „Kreuzprodukt-Klassen“ nicht gewartet werden müssen, da ihre Implementationen nahezu vollständig von den genutzten Oberklassen ererbt werden können. Speziell der Diskriminator „explizite vs. implizite Kanten“ mag aus der Perspektive eines „Lehrbuch-OO-Entwurfs“ kritisch erscheinen, vgl. auch [Oestereich 2005, Kap. 3.2.14]. Wir verteidigen unseren Entwurf mit dem Argument, dass diese Meta-Information über die Kanten in unserem Rahmenwerk unterschiedliches Verhalten bedingt, und daher als Kriterium zur Implementation von Verhaltensvariabilität durch dynamisches Binden genutzt werden muss (u.a. benötigen die Implementationen der Iteratoren diese Information auf *Typ- und somit Klassen-Ebene*).

5.1.3.2. (Leitdomäne 2) Substrate für einen SWIR-Dienst auf RACERPRO-Basis

Als Basis-Substrat für den SWIR-Dienst unter Verwendung von RACERPRO als OWL-Inferenzmaschine wählen wir das RACERPRO-ABox-Adapter-Substrat (RACERPRO-ABox-Adapter-Substrate). Bei Bedarf kann dieses Substrat mit einem Daten-Substrat (Data-Substrate) assoziiert und in einem entspr. hybriden Substrat aggregiert werden (s. Kap. 3.2.4). Wir kommen hierauf in Kap. 5.3.3.2 zurück.

5.1.4. Objektorientierte Implementation

Die Implementation ist als direkte Umsetzung des Entwurfs gestaltet. Wir beschreiben hier nur die wesentlichsten Punkte.

Die benötigten Iteratoren wurden im Entwurf als Methoden `loop-over-nodes`, `loop-over-edges` ausgeführt. Zusätzlich zu dieser funktionalen Schnittstelle existiert auch noch eine Makro-Schnittstelle. Beispielsweise kann ein Knoten-Iterator wie folgt aufgesetzt werden:

```
(loop-over-nodes (node substrate)
  (write node))
```

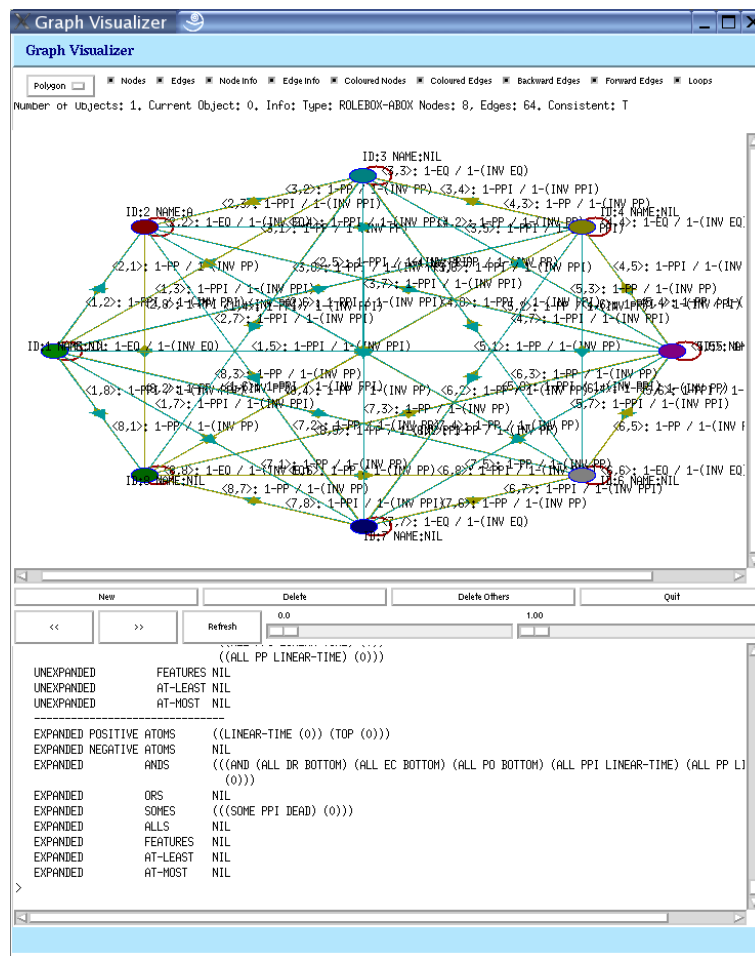


Abbildung 5.12.: Ein Substrat-Visualisierungswerkzeug - Der SUBSTRATE INSPECTOR.

Zusätzlich sind die Makros `loop-over-edges`, `loop-over-successors`, `loop-over-predecessors` vorhanden. Alle Iterator-Makros sind so ausgelegt, dass sie, wie auch die Methoden, dynamisches Binden anhand der Klasse von `substrate` und `node` unterstützen (die Makros delegieren an die entspr. Methoden).

Ähnlich sehen auch die *räumlichen Iteratoren zur Realisierung der räumlichen Selektion* im geometrischen Substrat aus; `ppi` ist eine RCC5-Relation:

```
(with-selected-spatial-objects (node reference-node t :ppi)
  (format t "Objekt ~A enthält Objekt ~A%" reference-node node))
```

Das HS-Substrat (MM-Substrate) verwendet Hashtabellen, um effizienten Zugriff auf die Knoten und Kanten eines Substrates zu gewährleisten. Die eingehenden und ausgehenden Kanten eines Knotens werden zudem in listenwertigen

zen in der Datenbank sind daher nicht indiziert; dies müsste für anwendungsspezifische Unterklassen von DB-Substrat (DB-Substrate) implementiert werden (die Beschreibungsinstanzen sind hier zu generisch, als das eine generelle Indizierung möglich wäre).

Die Klasse Geometrisches-Substrat (Geometric-Substrate) ist momentan nur als Hauptspeicherimplementation verfügbar. Prinzipiell könnte es unter Verwendung einer räumlichen Datenbank implementiert werden. Als räumlicher Index wird eine einfache *Bucket-Struktur* [Bartelme 1995, Kap. 8.4], [Sedgewick 1988, Kap. 27] verwendet, die für unsere Zwecke gute und ausreichend performante Dienste leistet. Die umfangreiche Bibliothek geometrischer Basis-Funktionen wurde vom Autor über viele Jahre hinweg selbst entwickelt und weiterentwickelt [Wessel 1996; 1998].

Es wurden auch einige *Inspektions-Werkzeuge*, wie z.B. der SUBSTRATE INSPECTOR (s. Abb. 5.12) implementiert. Wird dieser zur Inspektion einer Instanz der Klasse RACERPRO-ABox-Adapter-Substrat (RACERPRO-ABox-Adapter-Substrate) verwendet, so erhält man einen ABox-Inspektor für RACERPRO.

Speziell zur Visualisierung der Karten-Substrate und der DISK wurde der MAP VIEWER entwickelt (s. Abb. 5.13). Der MAP VIEWER verwendet die Thematik der Kartenobjekte zur Farbgebung. Der MAP VIEWER ermöglicht auch klassische, layerbasierte Anfragen bzw. Analysen des Datenbestands. Insofern kann er als typische GIS-Präsentations- und -Analysekomponente betrachtet werden, wie sie auch in konventionellen GIS zu finden ist. Links stellt der MAP VIEWER die Liste der Objektschlüssel dar. Einzelne Objektschlüssel können hier selektiert und deselektiert werden. Lediglich Instanzen der selektierten Objektschlüssel werden vom MAP VIEWER dargestellt. Der MAP VIEWER bietet nicht nur Inspektionsfunktionalität bzgl. thematischer, sondern auch bzgl. räumlicher Aspekte. Beispielsweise finden sich im „Spatial Querying“-Menü einige räumliche Basis-Anfragen, die direkt unter Verwendung des räumlichen Indexes evaluiert werden. So können z.B. die in einem selektierten Objekt enthaltenen, oder die das selektierte Objekt überlappenden Objekte visuell hervorgehoben werden.

Alle Oberflächen wurden mit Hilfe des CLIM-Rahmenwerks erstellt [McKay 1991, Franz Inc. 1994].

5.2. Realisierung der Substrat-Anfragebeantwortungsmaschine

Die abstrakte Syntax und auch die Semantik der Substrat-Anfragesprache(n) wurde in Kap. 4.2 definiert. Demnach hat eine Anfrage einen Kopf und einen Rumpf (s. Def. 73, S. 211); der Rumpf wird mittels der Konstruktoren $\wedge, \vee, \setminus, \pi$ aus Teilrumpfen bzw. Atomen zusammengesetzt. Wir hatten (unäre) Knoten-Atome und (binäre) Kanten-Atome unterschieden, s. Def. 68, S. 205. Atome referenzieren Objekte, womit Variablen und Individuen gemeint sind. Ein unäres Atom referenziert ein Objekt, ein binäres Atom zwei Objekte. Die Extension einer Anfrage ergibt sich kompositional anhand der Extensionen der Teilrumpfe anhand der Semantik der Rumpfkonstruktoren und der Semantik der Atome, s. Kap. 4.2.10. Wesentlich für die Semantik-Definition (s. Def. 76, S. 213) ist die instantiierungsspezifisch zu redefinierende Implikations- bzw. Folgerbarkeitsbeziehung, „ $\models$ “, bzw. aus softwaretechnischer Perspektive die Multimethode `implies-p`. Für einige „einfache“ Arten von Substraten vereinfacht sich die Semantik, s. Kap. 4.2.3.

Wir beschreiben im Folgenden die Realisierung (softwaretechnischer Entwurf und Implementation) einer Anfragebeantwortungsmaschine für die Substrat-Anfragesprache und setzen somit den formalen Entwurf aus Kap. 4.2 fort. Die Maschine ist wiederum generisch zu gestalten, da $\mathcal{L}_{QV}$ und $\mathcal{L}_{QE}$ instantiierungsspezifisch sind. Dies beschränkt sich nicht nur auf eine instantiierungsspezifische `implies-p`-Methode, sondern es müssen auch die unterschiedlichen konkreten Syntaxen dieser Sprachen berücksichtigt werden, etc. Die Maschine ist also wieder als Rahmenwerk zu realisieren; ihre Funktionalität wird durch Schablonenmethoden realisiert. Die in den Schablonenmethoden gerufenen Methoden können für spezielle Instantiierungen redefiniert bzw. überschrieben werden. Die Schablonenmethoden können jedoch auch komplett überschrieben werden.

Wir möchten betonen, dass wir hier nur die Realisierung des generischen Anteils dieser Maschine beschreiben, also des Rahmenwerks. Die konkreten Instantiierungen dieses Rahmenwerks (u.a. NRQL und DISKQL) werden erst in Kap. 5.3 dargestellt.

5.2.1. Softwaretechnische Anforderungen

Die durch die Leitdomänen gestellten (überwiegend funktionalen) Anforderungen hatten wir in Kap. 3 identifiziert und im formalen Entwurf berücksichtigt. Es wurden in Kap. 3 jedoch auch eine Reihe pragmatischer bzw. operationaler Anforderungen an die Anfragebeantwortungsmaschine identifiziert, die

natürlich nicht die Syntax oder Semantik der Anfragesprache betreffen, aber die Funktionalität der zu entwerfenden Maschine. Wir listen daher erneut stichpunktartig die operationalen und pragmatischen Anforderungen auf:

- Flexibilität bzgl. der konkreten Syntax und Semantik der Sprachen $\mathcal{L}_{QV}$ und $\mathcal{L}_{QE}$,
- inkrementelle Anfragebeantwortung,
- Mehrbenutzerfähigkeit (Anfragen sollen nebenläufig abgearbeitet werden können),
- Lebenszyklus-Verwaltung von Anfragen (Anfragen sollen als Objekte vorliegen und mittels API-Funktionen zugreifbar und manipulierbar sein),
- ressourcenkonfigurierbare Antwortberechnungsmodi:
 - proaktive oder bedarfsgerechte (inkrementelle) Berechnung von Tupeln,
 - unvollständige Modi,
 - Organisation der Tupelberechnung in Phasen (zunächst Berechnung „einfacher“ Tupel in Phase 1 mittels unvollständiger Verfahren, dann Berechnung der restlichen „teuren“ Tupel in Phase 2 mittels vollständiger Verfahren bei Bedarf);
- lose Kopplung an verwendetes externes DL-System,
- feste Kopplung bzw. hohe Kohäsion mit MIDELORE,
- Nutzung instantiierungsspezifischer Indexstrukturen,
- Unterstützung hybrider Anfragesprachen,
- Bereitstellung eines funktionsfähigen „Standardangebots“ bzw. eines Repertoires von Standard-Atomen zur unmittelbaren Nutzung zur Anfragebeantwortung auf Instanzen der bereitgestellten Standard-Substratklassen.

5.2.2. Objektorientierter Entwurf

Die Basis-Entwurfsidee für die Maschine ist, eine sog. *Anfragebearbeitungskette* als Schablonenmethode(n) bereitzustellen, s. Abb. 5.14. Diese 7-stufige Kette wird für bzw. von Anfragen wie folgt durchlaufen:

1. Die Klasse des Substrates (an das die Anfrage gestellt wird) wird verwendet, um einen für dieses Substrat geeigneten Parser zu bestimmen. Der Parser kennt die konkrete Syntax der für diesen Typ von Substrat definierten Sprachen $\mathcal{L}_{QV}$ und $\mathcal{L}_{QE}$.

5.2 Realisierung der Substrat-Anfragebeantwortungsmaschine

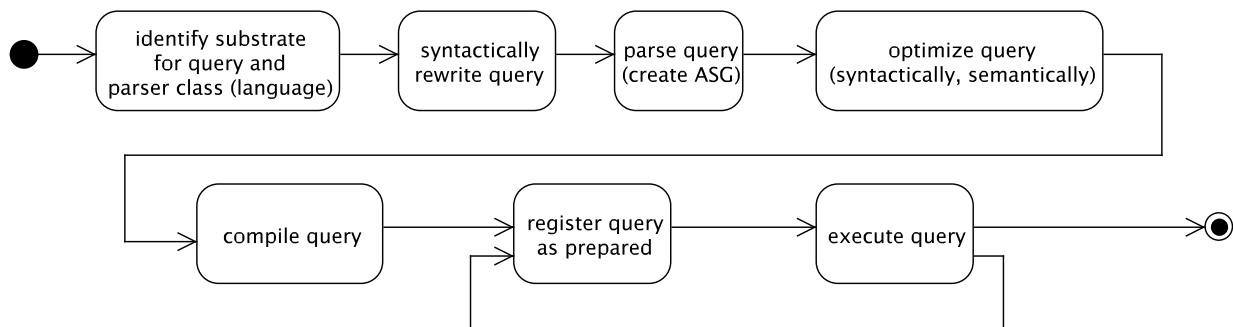


Abbildung 5.14.: Die Anfragebearbeitungskette.

2. Die originale Anfrage wird vom so bestimmten Parser zunächst syntaktisch umgeschrieben. Im Rahmen des Umschreibens wird *syntaktischer Zucker aufgelöst* und die *Anfrage normalisiert*.
3. Die Anfrage in konkreter Syntax wird vom Parser nun in einen *abstrakten Syntaxgraphen (ASG)* überführt, wobei es sich um die Hauptaufgabe des Parsers handelt. Der ASG ist ein Objektgraph; die Objekte dieses Graphen nennen wir auch ASG-Knoten (vgl. Abb. 5.16).
4. Der ASG wird als Grundlage für eine (*syntaktische und semantische*) *Anfrageoptimierung* verwendet. Das Resultat ist ein *Anfragebearbeitungsplan (ABP)*, welcher eine Serialisierung des ASGs und somit die Bearbeitungsreihenfolge darstellt.
5. Der Anfragebearbeitungsplan wird verwendet, um die *Anfragebearbeitungsfunktion (ABF)* für den Rumpf (und jeden Teilrumpf) der Anfrage zu erstellen und dem entsprechenden ASG-Knoten zuzuweisen.

Die ABF ist eine COMMON LISP-Funktion, die für die Berechnung der Extension des Teilrumpfes zuständig ist. Die ABF kann auf zwei Arten realisiert werden:

- a) Durch *Interpretation* des ABP (also des serialisierten ASGs) zur Anfrage-Laufzeit (wir verwenden dann ein Muster, das dem *Interpreter-Muster* aus [Gamma u. a. 1996] ähnelt).
- b) Durch *Übersetzung* des ABP in ein COMMON LISP-Suchprogramm. Hierdurch wird Interpretationsaufwand zur Laufzeit vermieden und in Übersetzungsaufwand umgewandelt. Eine so in eine *compilierte COMMON LISP-Funktion übersetzte Anfrage* wird i.d.R. effizienter ablaufen.

Prinzipiell unterscheiden sich die beiden Realisierungsmöglichkeiten nicht wesentlich, wenn entsprechende Abstraktionen bereitgestellt werden. In COMMON LISP ist der Compiler Teil der Sprache. Daher können beide Va-

rianten ohne großen Mehraufwand von der Maschine unterstützt werden.

6. Die Vorbereitungsphase der Anfrage ist nun abgeschlossen. Die Anfrage wird als *vorbereitet* registriert und ist bereit für die (evtl. spätere) Ausführung.
7. Auf Anforderung wird die Anfrage schließlich ausgeführt, wozu die ABF des Rumpfes gerufen wird. Zur *Anfragebeantwortungszeit* sehen wir dann noch vor, dass *Notifikationsmethoden* („Hooks“) gerufen werden – so kann ein Nutzer beispielsweise benachrichtigt werden, wenn ein neues Anfrageergebnis (Tupel) gefunden wurde.

Die einzelnen Schritte dieser Bearbeitungskette werden als Schablonenmethoden (i.d.R. in der Klasse *Parser* oder *Anfrage* (*Query*)) vorgesehen. Wir beschreiben nun jeden Schritt in der Bearbeitungskette detaillierter.

5.2.2.1. Schritt 1 - 3: Syntaktisches Umschreiben und Parsen

Zunächst muss anhand der Klasse des zu befragenden Substrates ein für dieses Substrat geeigneter Parser ausgewählt bzw. erzeugt werden, der die konkrete Syntax der für dieses Substrat definierten Sprachen $\mathcal{L}_{QV}$ und $\mathcal{L}_{EQ}$ kennt. Parser sind Instanzen einer Unterklasse der abstrakten Klasse *Parser*. Ein Substrat wird einen *Standardparser* auszeichnen, indem eine Methode *get-parser-class-for-substrate* definiert wird, die den Klassennamen des Standardparsers dieser Substratklasse zurückliefert.

Der Parser muss die konkrete Syntax der Sprachen $\mathcal{L}_{QV}$, $\mathcal{L}_{QE}$ kennen. Für gegebenes Wort w in konkreter Syntax muss der Parser $\mathcal{L}_{QV}$ bzw. $\mathcal{L}_{QE}$ identifizieren können, sodass $w \in \mathcal{L}_{QV}$ bzw. $w \in \mathcal{L}_{QE}$ gilt. Wir sagen, dass der Parser eine *Klassifikation des Wortes* durchführen muss.

Bevor die Anfrage von der erzeugten Parser-Instanz übersetzt wird, wird die Anfrage jedoch syntaktisch (semantikerhaltend) umgeschrieben. Zum einen ist diese Umschreibung notwendig, da die nachfolgenden Bearbeitungsschritte lediglich für Anfragen einer gewissen syntaktischen Struktur bzw. für Anfragen in einer „Normalform“ (s.u.) definiert sind. Zum anderen kann hier *syntaktischer Zucker*, z.B. in Form von Anfrage-Idiomen (s. Kap. 4.2.9), aufgelöst werden.

Wir sehen hierzu die Methode *syntactically-rewrite-query* in der Klasse *Parser* vor. Das Umschreiben geschieht durch die Technik des rekursiven Abstiegs. Beim Absteigen wird für jeden Teilausdruck mittels *get-expression-type* vom Parser eine Klassifikation des Ausdrucks angefordert. Je nach Klassifikationsergebnis wird dann entweder *rewrite-complex-query-body* oder *rewrite-query-atom* (mit Argument zur genaueren Kennzeichnung des Typs des Ausdrucks) aufgerufen; als Ergebnis wird der umgeschriebene Ausdruck

5.2 Realisierung der Substrat-Anfragebeantwortungsmaschine

zurückgegeben.

Nach erfolgter syntaktischer Umschreibung wird die Anfrage q *normalisiert*:

- Der *Optimierer* (s.u.) verlangt, dass die Anfrage q π -frei (Def. 73, S. 211) und in DNF (Def. 79, S. 215) ist.
- Die *Anfrageauswertungsmaschine* (*Methode execute-query*) verlangt hingegen lediglich, dass die Anfrage q π -frei und in NNF (Def. 78, S. 215) ist; zusätzlich wird verlangt, dass die Disjunkte einer Disjunktion die gleiche Stelligkeit haben (Def. 81, S. 218).

In einem weiteren Schritt wird daher zunächst die Menge $\mathcal{M}(q)$ (Def. 80, S. 217) berechnet. Die Rümpfe in $\mathcal{M}(q)$ seien nun in DNF oder NNF, wie gewünscht; offensichtlich können wir in Def. 80, S. 217 einfach DNF gegen NNF abschwächen. Dann wird der Operator *normalisiere_disjunkte* (Def. 81, s. 218) auf die Elemente in $\mathcal{M}(q)$ angewendet. Individuen wurden zudem durch Anwendung des Operators *ersetze_individuen* (Def. 82, S. 219) gegen repräsentative Variablen und $\boxminus$ -Konjunkte ersetzt (diese Ersetzung erleichtert die Implementation). Die Normalform einer Anfrage ist daher durch folgende Menge $\mathcal{N}(q)$ gegeben, deren Elemente die Bedingungen zur weiteren Verarbeitung durch die Bearbeitungskette erfüllen:

Definition 93 (Normalform $\mathcal{N}(q)$ einer Anfrage q) Die Normalform einer Anfrage q ist wie folgt definiert:

$$\mathcal{N}(q) =_{def} \{ \text{normalisiere\_disjunkte}(q') \mid q' \in \mathcal{M}(\text{ersetze\_individuen}(q)) \}.$$

■

Die Elemente in $\mathcal{N}(q)$ werden nun durch Aufruf der *parse-query*-Methode in ASGs überführt. Ein ASG ist ein Objektgraph, s. Abb. 5.16. Dementsprechend müssen wir ASG- bzw. Anfrage-Klassen zur Repräsentation von Anfrage-Bestandteilen vorsehen (Klassen für Variablen und Individuen, Atome, komplexe Rümpfe, etc.). Das Klassendiagramm in Abb. 5.15 zeigt die verschiedenen ASG-Klassen. Prinzipiell gibt es einen Strang für atomare Anfragen (Atomare-Anfrage (Atomic-Query)), und einen Strang für komplexe Anfragen (Komplexe-Anfrage (Complex-Query)), sowie Klassen zur Repräsentation von Variablen (Anfrage-Variable (Query-Variable)) und Individuen bzw. Knoten (Anfrage-Individuum (Query-Individual)). Der Kopf der Anfrage wird lediglich als Attributwert *query-head* in Anfrage (Query)-Instanzen repräsentiert; Rümpfe werden also als ASG-Instanzen mit leerem *query-head* Attributwert repräsentiert.

Atomare Anfragen werden in unäre und binäre Atome spezialisiert, s. Unäres-Atom (Unary-Query) und Binäres-Atom (Binary-Query). Da stets $\{\top, \perp\} \subseteq \mathcal{L}_{\text{QV}}$ gefordert wird, sehen wir entsprechende Klassen *Top-Atom* (Top-Query) und *Bottom-Atom* (Bottom-Query) als Unterklassen von Unäres-Atom (Unary-Query) vor. Zudem findet sich hier das $\boxminus$ -Atom als Klasse *Same-As-Atom* (Same-As-Query). Es

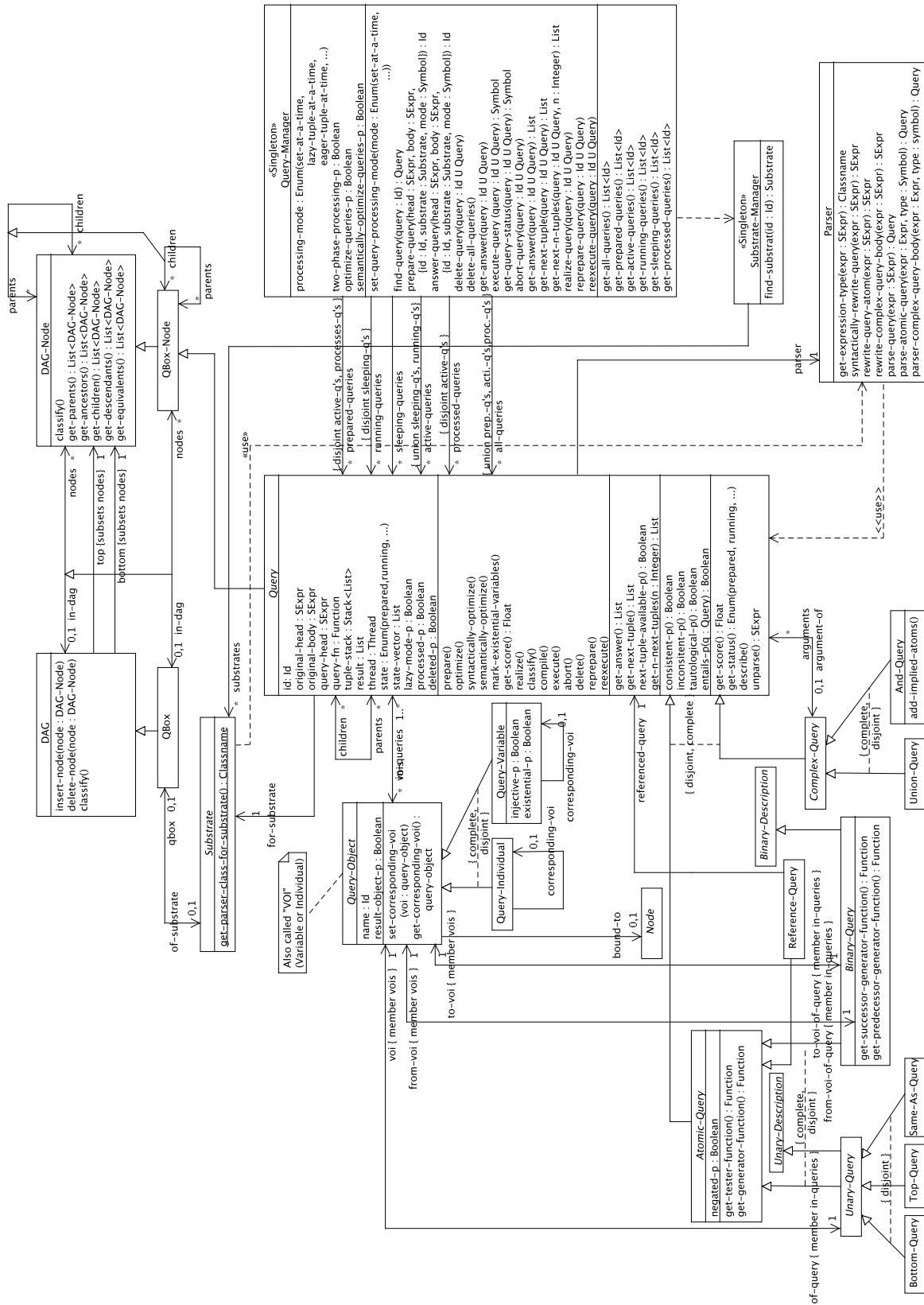


Abbildung 5.15.: Der Basis-Entwurf der Anfragebeantwortungsmaschine.

5.2 Realisierung der Substrat-Anfragebeantwortungsmaschine

handelt sich um ein unäres Atom. Das Referenz-Atom gibt Anlass zur Klasse Referenz-Atom (Query-Reference).

Die Klassen Unäres-Atom (Unary-Query) und Binäres-Atom (Binary-Query) sind abstrakt. Wir werden weiter unten konkrete Unterklassen dieser angeben. Wie in Def. 68, S. 205 dargestellt, haben Knoten-Atome die Form (a, v) und Kanten-Atome die Form (a, b, w) ; dabei ist $v \in \mathcal{L}_{QV}$ ein Wort der *Knoten-Anfragesprache*, und $w \in \mathcal{L}_{QE}$ ein Wort der *Kanten-Anfragesprache*. In unserem Entwurf werden diese Sprachen durch die Klassen der Atome festgelegt. Zudem sind diese Klassen Unterklassen von Beschreibungsklassen (s. Unäre Beschreibung (Unary-Description) und Binäre Beschreibung (Binary-Description)). Wir hatten erläutert, dass Beschreibungsinstanzen (und daher auch Atome) *selbstübersetzend* sind, s. S. 239. Zum Parsen eines Atoms muss daher vom Parser lediglich die korrekte ASG-Klasse mittels `get-expression-type` identifiziert werden. Beim Initialisieren der ASG-Instanz für das Atom wird das Parsing von v bzw. w dann mittels der Methode `initialize-description` vom Atom selbst durchgeführt.

Komplexe Anfragen hatten wir durch Bekanntgabe der möglichen Rumpfkonstruktoren $\{\wedge, \vee, \backslash, \pi\}$ fixiert (Def. 73, S. 211). Wir sehen (mit Ausnahme von π , welches ja durch Transformation entfernt wird, und mit Ausnahme von $\backslash$, s.u.) entsprechende ASG-Klassen für diese vor, also die Klassen Und-Rumpf (And-Query) und Oder-Rumpf (Or-Query). Diese Klassen sind instantiierbar; es wird i.d.R. keinen Grund geben, anwendungsspezifische Spezialisierungen dieser Klassen zu erzeugen. Da die Anfragebeantwortungsmaschine verlangt, dass Anfragen in NNF gebracht werden, kann $\backslash$ nur vor Atomen auftreten. Für ein Atom vermerkt das Attribut `negated-p` (s. Klasse Atomare-Anfrage (Atomic-Query)) daher einfach, ob es NAF-negiert ist oder nicht.

Der ASG eines Elements aus $\mathcal{N}(q)$ wird durch rekursiven Abstieg konstruiert. Mittels `get-expression-type` wird für jeden Teilrumpf eine Klassifikation angefordert; der Parser konstruiert dann einen ASG-Knoten des zurückgegebenen Klassennamens. Das verwendete Muster ähnelt also dem *Erbauer-Muster* [Gamma u. a. 1996]. Für komplexe Rümpfe (Konjunktion, Disjunktion) werden zunächst rekursiv die ASGs der Argumente konstruiert, die dann zum ASG des Rumpfes zusammengesetzt werden.

5.2.2.2. Schritt 4: Optimierung von Anfragen in Normalform

Nach erfolgter syntaktischer Vorverarbeitung liegt die originale Anfrage q nun also als eine *Liste von ASGs* $\mathcal{N}(q)$ vor, s. Def. 93, S. 282. Die hier beschriebenen Optimierungstechniken sind nur für π -freie Anfragen in DNF anwendbar. Die Elemente in $\mathcal{N}(q)$ müssen also nur dann in DNF sein, wenn Optimierung

erwünscht ist.⁷

Für eine Anfrage aus $\mathcal{N}(q)$ sehen wir dann

1. semantische Optimierung,
2. Bestimmung existentieller Variablen, und
3. syntaktische bzw. heuristische Optimierung

vor (in dieser Reihenfolge). Alle drei Optimierungsarten sind optional und werden durch die Methode `optimize-query` angestoßen.

Die semantische Optimierung (Methode `semantically-optimize-query`) muss vor der heuristischen Optimierung erfolgen, da hier der ASG u.U. weitgehend verändert wird (es können neue Knoten hinzukommen, entfernt oder verschmolzen werden, s.u.). Die semantische Optimierung ist eine Anfrageumschreibung, nun jedoch auf ASG-Ebene; prinzipiell könnte diese Umschreibung jedoch auch von der Methode `rewrite-complex-query-body` vorgenommen werden. Natürlich muss das Ergebnis der Umschreibung noch die Voraussetzungen für die Weiterverarbeitung der Anfrage durch die Kette erfüllen (also weiterhin in NNF und π -frei etc. sein, s. Def. 93).

Die heuristische Optimierung (Methode `syntactically-optimize-query`) beschränkt sich hingegen auf das *Umordnen* der Kinderknoten eines ASG-Knotens, der einen konjunktiven oder disjunktiven Rumpf darstellt. Der bereits erwähnte *Anfragebearbeitungsplan (ABP)* ist durch die Reihenfolge dieser Kinderknoten gegeben. Insbesondere sind die Kinderknoten Atome, da die Anfrage ja in DNF ist.

Wir beschreiben nun jede Optimierungstechnik detaillierter.

5.2.2.2.1. Semantische Optimierung Die *semantische Optimierung* verwendet zwei Techniken:

1. Wir hatten in Def. 90, S. 227 die sog. *QBox (Query Box)* als GAG definiert, welcher bereits gestellte Anfragen inklusive ihrer gespeicherten Antworten in einem Subsumptionsverband hält.

Wie dargestellt, wird ein Test auf Anfrage-Subsumption der Art $q_2 \sqsubseteq q_1$ benötigt, der für die entspr. ASG-Klassen zu implementieren ist. Wir sehen wieder eine Multimethode `entails-p` für diesen Zweck vor, die mit der `implies-p`-Methode vergleichbar ist, jedoch für Paare Anfrage (Query) $\times$ Anfrage (Query) definiert ist.

Wir stellen weiter unten einen (unvollständigen) Algorithmus zum Test auf Anfragesubsumption für eine ABox-Anfragesprache vor. Da wir die *QBox*

⁷Die DNF einer Anfrage kann exponentiell viel größer als die originale Anfrage sein.

nur zu Optimierungszwecken nutzen, ist hier auch ein unvollständiger Algorithmus bzw. ein Algorithmus, der nur für bestimmte Arten von Anfragen korrekt und vollständig ist, tolerierbar (die Anfragen, für die der Algorithmus unvollständig ist, werden nicht in die QBox eingetragen).

Der *Anfrage-Klassifikationsprozess* (s. S. 227) wird wieder als Schablonenmethode *classify-query* implementiert. Hier sind zudem die in [Baader u. a. 1994] beschriebenen (und ursprünglich zur Beschleunigung der TBox-Taxonomieberechnung definierten) Optimierungen anwendbar.

Sollte sich eine Anfrage (durch Klassifikation) als äquivalent zur $\top$ -Anfrage in der QBox herausstellen, so wird eine Warnung ausgegeben – die Anfrage ist dann *tautologisch*. Eine solche Anfrage ist i.d.R. nicht gewünscht bzw. nicht sinnvoll, da sie über keine Selektivität verfügt. Bei Bedarf kann das Ergebnis dennoch angefordert werden. Stellt sich eine Anfrage als äquivalent zur $\perp$ -Anfrage der QBox heraus, so ist die Anfrage *unerfüllbar* bzw. *inkonsistent*. Es wird ebenfalls eine Warnung ausgegeben und (bei Ausführung) dann einfach die leere Menge zurückgegeben. Speziell im Falle von inkonsistenten Anfragen kann also vergeblich investierte Anfragebeantwortungszeit eingespart werden.

2. Eine weitere Technik nennen wir *Anfrage-Realisierung*. Hierbei handelt es sich um eine (nach Wissen des Autors zuvor nicht untersuchte) Technik, die als Analogon zur *ABox-Realisierung* (s. S. 79) beschrieben werden kann.

Die Basis-Idee ist, dass eine konjunktive Anfrage um *implizierte Atome* erweitert wird. Die benötigte Implikationsbeziehung ist vergleichbar mit der für die Anfragesubsumption benötigten Methode *entails-p*, jedoch wird hier die Verschärfung auf Variablenvektoren gleicher Stelligkeit aufgehoben (diese Verschärfung war aus nichtlogischen Gründen erforderlich, s. S. 227). Es sollen hier stets solche implizierten Atome hinzugefügt werden, die möglichst *speziell* sind. Hierdurch soll die *Informiertheit* und somit *Zielgerichtetheit* des für die Anfragebearbeitung verwendeten Suchverfahrens erhöht werden.

Zur Illustration dieser Technik betrachten wir eine Beispiel-Anfrage, die *ABox-Atome* verwendet:

$$(x, \forall R.D) \wedge (x, C) \wedge (x, y, R)$$

Die sog. Basis-ABox-Anfragesprache wird weiter unten formal eingeführt; wir setzen hier intuitives Verständnis voraus, s. auch Kap. 4.2.2, S. 208. Offensichtlich muss eine *y*-Bindung nun auch *D* erfüllen, also *D*-Instanz sein, da $(x, y, R) \wedge (x, \forall R.D)$ gefordert ist. Es ist jedoch kein Knoten-Atom (y, D) vorhanden, welches diese Forderung explizit machen würde.

Die Anfragebearbeitungsfunktion, welche das Anfrageergebnis berechnet,

würde daher *schlecht informiert* sein. Angenommen, die Anfragebearbeitungsfunktion sucht in einem ersten Schritt nach Bindungsmöglichkeiten für y . Offensichtlich wäre ein *Generator zur Etablierung von y -Bindungen* besser beraten, die Menge der D -Instanzen als Kandidaten zu enumerieren, als die Menge der $\top$ -Instanzen. Dieses Wissen steht dem Generator jedoch nicht zur Verfügung, da es nicht explizit ist. Zudem bestehen nur solche y -Kandidaten die weiteren Tests bzw. Anfragebedingungen, die zusätzlich R -Nachfolger eines Knotens sind, für den dann auch noch $C \sqcap \forall R.D$ gilt.

Werden diese Forderungen gleich beim Generieren von Kandidaten für y berücksichtigt, so wird fruchtloses Durchprobieren von Bindungsmöglichkeiten für y im Suchprozess minimiert und der *Verzweigungsfaktor (Branching Factor)* im Suchbaum reduziert. Wird z.B. eine Instanz von $\neg D$ als Kandidat für eine y -Bindung gewählt, so steht bereits fest, dass keine erfüllende x -Bindung mehr gefunden werden kann. Dennoch würden in diesem Falle mangels weiteren Wissens alle R -Vorgänger von y als Kandidaten für x betrachtet. Dies wird in der *Constraint Solving*-Forschungsgemeinde als *Thrashing* bezeichnet [Russel u. Norvig 2003, Kap. 5], [Kumar 1992].

Es sind diverse Techniken zur Vermeidung bzw. Minimierung von Thrashing bekannt. Begreifen wir die Berechnung von Anfrageergebnissen als Suche von Lösungen eines endlichen Constraintsystems, so wird durch das Hinzufügen der implizierten Konjunkte (y, D) und $(y, \exists R^{-1}.C)$ eine Art von *Forward Checking* [Kumar 1992] bzw. Kantenkonsistenz erzwungen. Allerdings ist nun auch das Enumerieren von y -Kandidaten aufwendiger, da die Bestimmung der y -Kandidaten als Instanzen von $D \sqcap \exists R^{-1}.C$ wesentlich teurer ist, als die Enumerierung von $\top$ -Instanzen. Hier muss ein guter Kompromiss zwischen Aufwendigkeit der Kandidatenberechnung vs. Thrashing-Potential gefunden werden.

Es stellt sich die Frage, *wie speziell* die hinzugefügten implizierten Atome sein sollen. Zum einen gibt es oftmals keine „speziellste Beschreibung“, da die Menge der implizierten Atome für eine Anfrage u.U. unendlich ist. Letztlich müssen (einige ausgewählte) implizierte Atome ja berechnet werden. Sicherlich spielt hier auch die Ausdrucksmächtigkeit der Sprachen $\mathcal{L}_{QV}$ und $\mathcal{L}_{QE}$ eine Rolle. Zum anderen kann das Hinzufügen von sehr speziellen implizierten Atomen, wie diskutiert, auch kontraproduktiv sein, da die Kandidatengenerierung dann berechnungsaufwendiger wird. Eine vielversprechende Idee ist, nur solche Atome hinzuzufügen, für deren Evaluierung dann Indexstrukturen genutzt werden können. Die Existenz von unterstützenden Indexstrukturen wäre dann das wesentliche Kriterium zur Entscheidung, ob die Berücksichtigung des zusätzlichen (implizierten)

5.2 Realisierung der Substrat-Anfragebeantwortungsmaschine

Atomes lohnenswert ist oder nicht. Prinzipiell können nicht nur implizierte Knoten-Atome hinzugefügt werden, sondern auch Kanten-Atome.

Eine sehr einfache Art der Realisierung besteht z.B. darin, *Konjunktionen von Knoten-Atomen* zusammen zu ziehen, sofern $\mathcal{L}_{QV}$ Konjunktion bietet: $(x, C) \wedge (x, D)$ wird gegen $(x, C \sqcap D)$ ersetzt. Es sei bemerkt, dass folgende Ersetzungen *nicht* gültig sind: $(x, C) \vee (x, D) \rightarrow (x, C \sqcup D)$, $\neg(x, C) \rightarrow (x, \neg C)$ (mit $\{\neg, \sqcup\} \subseteq \mathcal{L}_{QV}$).

Eine komplexere Anfragerealisierung erfordert hingegen einen $\mathcal{L}_{QV}$ - und $\mathcal{L}_{QE}$ -spezifischen Algorithmus, der implizierte Atome berechnen kann. Im Falle der ABox-Anfragesprache können wir den *ABox-Realisierungsdienst* für eine einfache konjunktive Anfrage nutzen. Die vorgesehene Schablonenmethode *realize-query* ist für eine konjunktive ABox-Anfrage q definiert und funktioniert wie folgt:

- In einem ersten Schritt werden implizierte Atome als neue explizite Konjunkte zu q hinzugefügt. Hierzu wird für eine konjunktive Anfrage die Methode *add-implied-atoms* aufgerufen. Die resultierende Anfrage heißt dann q' ; sie ist offensichtlich äquivalent zu q .
- In einem zweiten Schritt werden für ein Anfrage-Objekt a lediglich die *speziellsten* Atome (a, v) beibehalten; alle Konjunkte (a, w) mit $v \models w$ und $w \not\models v$ werden entfernt. Analog verfahren wir für die Kanten-Atome. Eine Anfrage kann mehrere Realisierungen haben.

Betrachten wir wieder die Anfrage

$$(x, C) \wedge (x, \forall R.D) \wedge (x, y, R),$$

so könnte das Ergebnis nach Schritt 1 z.B.

$$\begin{aligned} & (x, C) \wedge (x, \forall R.D) \wedge (x, C \sqcap (\forall R.D) \sqcap (\exists R.D)) \wedge \\ & (y, D \sqcap \exists R^{-1}.(C \sqcap (\forall R.D) \sqcap (\exists R.D))) \wedge \\ & (x, y, R) \wedge (y, x, R^{-1}) \end{aligned}$$

sein. Schritt 2 würde die implizierten (redundanten) Atome (x, C) und $(x, \forall R.D)$ entfernen, und

$$(x, C \sqcap (\forall R.D) \sqcap (\exists R.D)) \wedge (y, D \sqcap \exists R^{-1}.(C \sqcap (\forall R.D) \sqcap (\exists R.D))) \wedge (x, y, R)$$

oder

$$(x, C \sqcap (\forall R.D) \sqcap (\exists R.D)) \wedge (y, D \sqcap \exists R^{-1}.(C \sqcap (\forall R.D) \sqcap (\exists R.D))) \wedge (y, x, R^{-1})$$

herstellen, wobei es sich um „minimale“ konjunktive Anfragen handelt, die äquivalent zur originalen Anfrage sind.⁸ „Minimal“ bedeutet hier, dass durch Entfernung eines Atoms die Anfrage nicht mehr äquivalent zur originalen Anfrage wäre. Eine *vereinfachte Version des Verfahrens* würde in Schritt 1 lediglich solche Konzept-Atome hinzufügen, die atomare Kon-

⁸Tatsächlich ist der Unterschied (y, x, R^{-1}) vs. (x, y, R) irrelevant für die Anfragebeantwortungsmaschine, s.u.

zepte verwenden (hier also lediglich (y, D)). Die entstandene Menge von Anfragen wird durch die nachfolgenden Optimierungsmethoden weiterbearbeitet.

5.2.2.2.2. Bestimmung existentieller Variablen In einem Rumpf wie $\pi(x)((x, C) \wedge (x, y, R))$ oder einer (hierzu äquivalenten) Anfrage $\text{antwort}(x) \leftarrow (x, C) \wedge (x, y, R)$ wird die Bindung von y niemals im Anfrageergebnis auftauchen. Tatsächlich ist es ausreichend, für jede an x gebundene C -Instanz zu überprüfen, ob ein an y zu bindender R -Nachfolger existiert. Da y von keinen anderen Atomen referenziert wird, müssen auch keine weiteren Einschränkungen für die generierte y -Bindung überprüft werden. Es ist daher ausreichend, für jeweils eine x -Bindung jeweils eine y -Bindung zu generieren. Wir bezeichnen y daher als *existentielle Variable*.

Variablen werden daher in diesem Optimierungsschritt mittels der Methode `mark-existential-variables` als *existentiell* gekennzeichnet. Die Generatoren (s.u.) berücksichtigen beim Enumerieren der Kandidatenbindungen für eine Variable, ob diese als existentiell markiert ist. Ist dies der Fall, so darf der Generator das Enumerieren nach der ersten Bindungsmöglichkeit einstellen. Offensichtlich kann eine Variable nicht existentiell sein, wenn sie von mehr als einem Atom referenziert wird, oder in der Projektionsliste eines π -Operators auftaucht, oder im Kopf der Anfrage. Insbesondere die Blätter von Baum-Anfragen können so als existentiell gekennzeichnet werden.

5.2.2.2.3. Heuristische, syntaktische Optimierung Der *heuristischen, syntaktischen Optimierung* von Anfragen liegt die Idee zugrunde, dass unterschiedliche Bindungsreihenfolgen der Variablen in einer *disjunktions-* und π -freien Anfrage zu wesentlich unterschiedlich performanten *Anfragebearbeitungsplänen* (ABPs) führen können. Wir verwenden hier *disjunktionsfrei* im Sinne von Def. 73, S. 211 um deutlich zu machen, dass auch NAF-negierte Atome in einer solchen Anfrage als Konjunkte auftreten dürfen (wir hatten eine *konjunktive Anfrage* in Def. 73, S. 211, als eine Anfrage definiert, deren Rumpf weder $\setminus$ noch $\vee$ enthält).

Ein ABP für eine disjunktions- und π -freie Anfrage ist durch die Bearbeitungsreihenfolge ihrer atomaren Konjunkte definiert. Wir notieren einen Plan als Vektor, z.B. $\vec{p} = \langle a_1, \dots, a_n \rangle$, wobei a_i die Atome des Planes sind.

Zur Illustration sei die konjunktive Anfrage $(x, C) \wedge (y, D) \wedge (x, y, R)$ betrachtet. Da diese Anfrage drei Konjunkte hat, sind $3! = 6$ ABPs möglich. Zwei mögliche ABPs sind $\langle (x, C), (y, D), (x, y, R) \rangle$ und $\langle (x, C), (x, y, R), (y, D) \rangle$.

Die Position eines Atoms in einem Plan bestimmt seine *Rolle*:

- Ein unäres Atom, wie (x, C) , kann entweder als *Kandidatengenerator* (kurz:

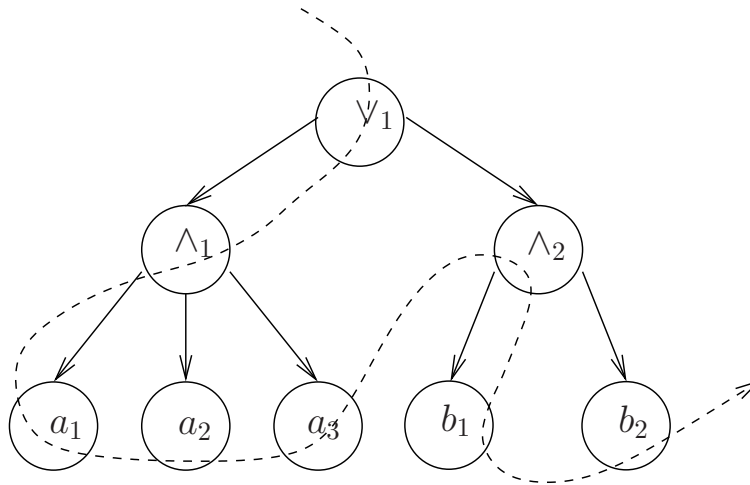


Abbildung 5.16.: ASG der Anfrage $(a_1 \wedge a_2 \wedge a_3) \vee (b_1 \wedge b_2)$ als Objektgraph und durch Prä-Order-Traversierung (gestrichelte Linie) festgelegter ABP $\langle \vee_1, \wedge_1, a_1, a_2, a_3, \wedge_2, b_1, b_2 \rangle$.

Generator) für Variablenbindungen verwendet werden, oder aber als *Tester* (auch: *Filter*) einer bereits etablierten Variablenbindung. Gilt $x \in V$ (ist x also ein Individuum), so ist (x, C) ebenfalls ein Tester.

- Ein binäres Atom, wie (x, y, R) , kann dementsprechend vier Rollen einnehmen, je nachdem, ob für keines, eines oder zwei seiner Objekte bereits Bindungen zum Bearbeitungszeitpunkt bestehen.

Dementsprechend klassifizieren wir die Rolle eines binären Atoms als

- *Generator*,
- *Nachfolger-Generator*,
- *Vorgänger-Generator*, oder
- *Tester (Filter)*.

Betrachten wir z.B. den Plan $\langle (x, C), (x, y, R), (y, D) \rangle$, so ist die Rolle von (x, C) die eines Generators, die Rolle von (x, y, R) die eines Nachfolger-Generators, und die Rolle von (y, D) die eines Testers. Letztlich muss natürlich für jede Variable ein Generator aufgesetzt werden. Oftmals gibt es jedoch mehr als eine Wahlmöglichkeit für einen Generator einer Variable. Es ist Aufgabe des heuristischen Optimierers, diese Wahlmöglichkeiten gegeneinander zu bewerten und einen vorteilhaften (performanten) Plan auszuwählen.

Betrachten wir den Plan $\langle (x, C), (y, D), (x, y, R) \rangle$, so nehmen sowohl (x, C) als auch (y, D) die Generator-Rolle an, was zur Berechnung eines Kreuzproduktes der Größe $|(x, C)^\varepsilon| \times |(y, D)^\varepsilon|$ führen wird. Das Atom (x, y, R) hat die Tester-Rolle.

Im Plan $\langle (x, C), (x, y, R), (y, D) \rangle$ hat (x, C) die Generator-Rolle, und (x, y, R) ist Nachfolger-Generator für y . Unter der Annahme, dass ein Knoten in $(x, C)^\varepsilon$ durchschnittlich n verschiedene R -Nachfolger hat, müssen dann lediglich $n|(x, C)^\varepsilon|$ Kandidaten daraufhin überprüft werden, ob sie in $(y, D)^\varepsilon$ enthalten sind; das Atom (y, D) hat die Rolle eines Testers. In der Regel wird n sehr viel kleiner als $|(y, D)^\varepsilon|$ sein. Daher ist dieser Plan dem Plan $\langle (x, C), (y, D), (x, y, R) \rangle$ vorzuziehen. Wir werden zudem in Kap. 5.3.2.1.1 zeigen, dass dieses „navigierende Auswertungsverfahren“ insbesondere für eine ABox-Anfragesprache vorteilhaft ist. Zudem können entsprechende *Statistik-Funktionen* über die Kardinalitäten, sofern bekannt, Auskunft geben und diese somit in die Bewertung der einzelnen Pläne einfließen.

Der syntaktische, kostenbasierte Optimierer wird eine heuristische Suche verwenden, um einen „guten“ Plan zu finden. Offensichtlich ist die Anzahl der Pläne für eine Konjunktion mit n Konjunkten $n!$, und so muss der Optimierer *heuristische, unvollständige Suchverfahren* zur Suche nach einem guten Plan verwenden, um eine kombinatorische Explosion zu vermeiden (er kann nicht alle $n!$ Pläne erzeugen und gegeneinander bewerten).

Der Plan wird daher schrittweise mit einem heuristischen Suchverfahren konstruiert. Wir definieren eine Bewertungsfunktion h (entspricht der Methode `get-score`) auf Plänen. Die Bewertung eines (Teil)Plans $\vec{p} = \langle a_1, \dots, a_n \rangle$ ergibt sich anhand der Bewertungen der Einzelatome $h(a_i)$, und $h(a_i)$ ergibt sich in der Standard-Implementierung anhand der Rolle von a_i im bisher konstruierten Teilplan. Für einen (Teil)plan p setzen wir in der Standard-Implementierung der Methode

$$h(p) = \sum_{1 \leq i \leq n} \frac{h(a_i)}{i}.$$

Aufgrund der Wichtung der Bewertung eines Atoms mit $\frac{1}{i}$ werden daher früh vorzunehmende Bindungsentscheidungen als wichtiger bewertet, als spät vorzunehmende Bindungsentscheidungen. Die Idee ist, dass der Verzweigungsfaktor im Suchbaum durch Wahl hochinformierter bzw. gut bewerteter Atome so früh wie möglich reduziert werden soll.

Wir verwenden das sog. *Strahlsuchverfahren (Beam Search)* [Winston 1993, S. 75] zur heuristischen Suche nach Plänen. Das Strahlsuchverfahren hält die jeweils k besten Pläne auf der Such-Agenda und ist suboptimal, kommt aber mit linearem Speicherbedarf und Rechenzeit aus, was ein großer Vorteil ist (sicherlich ist ein exponentiell viel Zeit- und Speicherplatz verbrauchender Optimierer nicht wünschenswert). Eine hinreichend große Wahl von k wird zudem das Problem der lokalen Maxima, wie es bei reinen *Greedy Verfahren* wie z.B. dem *Hill Climbing*-Suchverfahren auftritt, lindern.

5.2 Realisierung der Substrat-Anfragebeantwortungsmaschine

Die Bewertungsfunktion h wird als Methode `get - score` vorgesehen. Wir arrangieren es so, dass `get - score` für Atome einen Wert zwischen 0 und 99 liefern muss, wobei höhere Werte bessere Bewertungen bedeuten.

Für unäre und binäre Atome sind bereits Standard-Implementationen von `get - score` vorgesehen, die, mangels weiteren Wissens, lediglich die Rolle des Atomes im bisherigen Teilplan zur Bewertung verwenden. Wir werden später zeigen, wie durch Überschreiben dieser Bewertungsmethoden für spezielle Instanziierungen auch domänenspezifisches Wissen berücksichtigt werden kann. Prinzipiell sind jedoch Tester höher zu bewerten als Generatoren; im Falle binärer Atome werden Nachfolger- und Vorgänger-Generatoren höher bewertet als Generatoren. Diese einfache Heuristik ist schon ausreichend, um Pläne zu forcieren, die Generatoren für Variablen anhand der Vorgänger und Nachfolger bevorzugen, und somit Bindungen für Variablen durch *Navigieren durch die relationale Struktur eines Substrates herstellen* und somit Kreuzprodukte vermeiden.

Die Standard-Implementation von `get - score` bewertet Instanzen der Klasse `Top-Atom` (`Top-Query`) in der Rolle des Generators mit 0, in der Rolle des Testers hingegen mit 99. Instanzen von `Bottom-Atom` (`Bottom-Query`) werden stets mit 99 bewertet. Zudem wird ein unärer Tester („t“) stets mit 99, ein unärer Generator („g“) mit 50, ein binärer Generator („gn“) mit 25, und ein binärer Nachfolger- oder Vorgänger-Generator („gn“, „gr“) mit 75 bewertet.

Dann ergeben sich folgende Kosten für die sechs Pläne der Beispielanfrage:

1. $\langle (x, C), (x, y, R), (y, D) \rangle$, Rollen: $\langle g, gn, t \rangle$, Bewertung: $\langle 50, 75, 99 \rangle$, Summe $50 + \frac{75}{2} + \frac{99}{3} = 120.5$
2. $\langle (x, C), (y, D), (x, y, R) \rangle$, Rollen: $\langle g, g, t \rangle$, Bewertung: $\langle 50, 50, 99 \rangle$, Summe $50 + \frac{50}{2} + \frac{99}{3} = 108$
3. $\langle (x, y, R), (x, C), (y, D) \rangle$, Rollen: $\langle g, t, t \rangle$, Bewertung: $\langle 25, 99, 99 \rangle$, Summe $25 + \frac{99}{2} + \frac{99}{3} = 107.5$
4. $\langle (y, D), (x, y, R), (x, C) \rangle$, wie 1.
5. $\langle (y, D), (x, C), (x, y, R) \rangle$, wie 2.
6. $\langle (x, y, R), (y, D), (x, C) \rangle$, wie 3.

Plan 1 und 4 werden also (wie erwartet) am höchsten bewertet.

Wie erwähnt, kann die `get - score`-Methoden für spezielle ASG-Klassen überschrieben werden. Attraktiv ist z.B. die (bereits angedeutete) Idee, evtl. vorhandenes Wissen über die Kardinalitäten der Extensionen in die Bewertungsfunktionen einfließen zu lassen; wir kommen hierauf in Kap. 5.3.3.6 zurück.

Der Bewertungsfunktion wird zudem Generatoren für Variablen, die als *existentiell* markiert sind, höher bewerten. Die entspr. Variablen wurden ja zuvor

markiert, sodass diese Information zur Verfügung steht.

Der Optimierer sollte bei optional erfolgter QBox-Klassifikation der Anfrage q auch die Extensionen der gefundenen Elternanfragen berücksichtigen. Prinzipiell müssen ja lediglich die Elemente

$$\bigcap_{p \in \text{parents}(q)} p^{\mathcal{E}} \setminus \bigcup_{c \in \text{children}(q)} c^{\mathcal{E}}$$

auf Mitgliedschaft in $q^{\mathcal{E}}$ überprüft werden (s. S. 227). Während die Elemente von $\bigcup_{c \in \text{children}(q)} c^{\mathcal{E}}$ sofort ohne weitere Inspektion als Teilmenge von $q^{\mathcal{E}}$ zurückgegeben werden können, wird die Implementation für die verbleibenden Tupel so vorgehen, dass *eine* ausgewählte Elternanfrage $p \in \text{parents}(q)$ als Generator ausgezeichnet wird. Anhand von $p^{\mathcal{E}}$ werden Variablenbindungen für q etabliert. Diese Bindungen werden auf Mitgliedschaft in $\bigcap_{p' \in \text{parents}(q) \setminus p} p'^{\mathcal{E}}$ getestet (alle Atome degradieren also zu Testern). Somit sollte $|p^{\mathcal{E}}|$ minimal in $\{|p^{\mathcal{E}}| \mid p \in \text{parents}(q)\}$ sein. Wir vermeiden so, dass (aufwendige) Verschneiden von n -stelligen Tupelmengen zu implementieren.

5.2.2.3. Schritt 5 - 7: Anfrageübersetzung und -bearbeitung von Anfragen in Normalform

Wir hatten die Reihenfolge, in der die einzelnen ASG-Knoten einer Anfrage bearbeitet werden, als Anfragebearbeitungsplan (ABP), kurz Plan, bezeichnet. Werden in dem ASG der Anfrage lediglich Knoten betrachtet, die Rümpfe⁹ darstellen, so kann der ASG als Baum bzw. AST (*Abstract Syntax Tree*) angesehen werden. Der ABP kann dann aus dem AST durch eine einfache Prä-Ordnungs-links-nach-rechts-Traversierung gewonnen werden, vgl. Abb. 5.16. Wir nehmen nicht nur die Blätter (also Atome), sondern auch die $\wedge$ - und $\vee$ -Knoten in den so konstruierten ABP auf. Die ASG-Knoten, die im ABP auftreten, nennen wir im Folgenden *ABP-Knoten*. Wir notieren einen Plan weiterhin durch $\langle k_1, \dots, k_n \rangle$; nun ist k_i nicht mehr notwendigerweise ein Atom. Die in Abb. 5.16 eingezeichnete Traversierung entspricht daher dem ABP $\langle \vee_1, \wedge_1, a_1, a_2, a_3, \wedge_2, b_1, b_2 \rangle$. Der APB ist daher letztlich durch die Ordnung bzw. Reihenfolge der Kinder-Knoten der ABP-Knoten gegeben.

Der *Anfrageübersetzer (Query Compiler)* wird nun für jeden ABP-Knoten k_i eine *Anfragebearbeitungsfunktion (ABF)* erzeugen. Die ABF ist für die inkrementelle Berechnung der Extension des Knotens verantwortlich; die ABF ist ein COMMON LISP-Funktionsobjekt, welches dem Knoten dann zugewiesen wird (s. Attribut `query-fn` der Klasse `Anfrage (Query)`). Dieser Vorgang (Schritt 5) wird durch die Methode `compile` (s. Klasse `Anfrage (Query)`) ausgelöst; wir sprechen

⁹Auch Atome sind Rümpfe.

5.2 Realisierung der Substrat-Anfragebeantwortungsmaschine

von *Compilierung oder Übersetzung*. Nach erfolgter Übersetzung ist die Anfrage *bereit*, wird als *vorbereitet registriert* (Schritt 6) und kann dann (evtl. zu einem späteren Zeitpunkt) *ausgeführt bzw. bearbeitet* werden (Schritt 7, Methode *execute*). Das Ausführen einer Anfrage *query* wird im Wesentlichen durch den Aufruf ihrer ABF mittels (`funcall (query-fn query)`) angestoßen. Nach erfolgter Anfragebearbeitung wird die Anfrage als *bearbeitet* registriert; sie kann alsdann z.B. erneut zu Ausführung gebracht oder gelöscht werden. Dieser *Lebenszyklus* einer Anfrage wird später detaillierter erläutert.

Prinzipiell wäre es denkbar, die ABFen als Methoden der entspr. ASG-Klassen vorzusehen, und nicht als einfache Funktionsobjekte (die als Wert des Attributes `query-fn` abgespeichert werden). Da die Anfragebearbeitung jedoch hocheffizient geschehen soll, sollte das mit Methodenaufrufen verbundene *späte Binden* möglichst nicht zur *Anfragebeantwortungszeit*, sondern lediglich zur *Anfrageübersetzungszeit* verwendet werden. Polymorphismus ist zur Beantwortungszeit zudem nicht notwendig, da die Typen der ABP-Knoten bereits zur Übersetzungszeit feststehen. Aus diesem Grund wird die ABF für jeden ABP-Knoten *zur Übersetzungszeit* durch einen Methodenaufruf (und somit ASG-klassenspezifisch) berechnet und als Wert des Attributes `query-fn` des Knotens abgespeichert.

Die durch den Übersetzer erzeugten ABFen haben eine Struktur, die zur Auswertungszeit eine Tiefensuche (Backtracking-Suche) nach erfüllenden Variablenbelegungen realisieren. Dieser Suchprozess kann auch als Ermittlung der Lösungen eines *Constraintsystems mit endlichen Domänen (Finite Domain Constraint System)* begriffen werden. Es ist wohlbekannt, wie hierzu Suchverfahren angewendet werden [Russel u. Norvig 2003, Kap. 5]. Wir setzen entsprechende Kenntnisse dieser klassischen Techniken voraus und erlauben uns daher eine abkürzende Darstellung.

Wir sehen vor, dass einer ABF eine *Fortsetzung (Downward Continuation)* als COMMON LISP-Funktionsobjekt als Argument übergeben werden kann. Dies ermöglicht flexible, datengesteuerte Funktionskomposition von ABFen.

Für die drei möglichen Arten von ABP-Knoten (Atom, $\wedge$ -Knoten, $\vee$ -Knoten) gehen die ABFen prinzipiell wie folgt vor:

- Für atomare Knoten k_i :
 - Hat k_i die Rolle eines Generators, so wird die ABF den *Generator* für dieses Atom aufrufen, um inkrementell Bindungen für seine Variable(n) zu generieren.

Stark vereinfacht kann man sich die Generator-ABF eines Atomes (x, C) wie folgt vorstellen (S sei das befragte Substrat):

```
(lambda (continuation)
  (loop-over-nodes (var S)
```



```
(when (matches-p var C)
      (setf (bound-to x) var)
      (funcall continuation)))
(return))
```

Wurde eine Bindung generiert, so wird mittels (funcall continuation) die Berechnung fortgesetzt.

Ist k_i ein Konjunkt (einer Konjunktion), und handelt es sich nicht um das letzte Konjunkt, so ist continuation durch die ABF des nächsten ABP-Knotens k_{i+1} gegeben; die Tiefe des so aufgespannten Suchbaums für die Tiefensuche entspricht dann offensichtlich der Anzahl der Konjunkte in der Konjunktion.

Für den Fall, dass es sich bei k_i um das letzte Konjunkt (einer Konjunktion) handelt, oder falls k_i ein Disjunkt (einer Disjunktion) ist, so können wir ein Ergebnistupel für die Extension des Eltern- $\wedge$ - oder $\vee$ -Knotens notieren¹⁰ – in diesem Fall ist continuation keine ABF, sondern eine Funktion, die das Ergebnistupel notiert. Optional können hier auch Duplikatstupel und Permutationen entdeckt und unterdrückt werden.

Können keine (weiteren) Bindung mehr vom Generator generiert werden, so wird die Kontrolle mittels return an die ABF des Eltern-Knotens zurückgegeben (bzw. die Anfragebearbeitung beendet, falls es keinen Eltern-Knoten mehr gibt).

- Hat k_i die Rolle eines Testers, so wird die ABF den *Tester* für diese Atom aufrufen, um die durch das Atom gegebene Bedingung für die bereits bestehende(n) Variablenbindung(en) (der Variable(n) des Atomes) zu überprüfen. Die ABF vereinfacht sich dann zu:

```
(lambda (continuation)
  (if (matches-p (bound-to x) C)
      (funcall continuation)
      ; else
      (return)))
```

- Die ABF eines $\wedge$ -Knotens k_i beschränkt sich im Wesentlichen darauf, die Tiefensuche für die Konjunktion „zu starten“. Hierzu ruft die ABF einfach die ABF des *ersten* Argument-Knotens auf, also die ABF des ABP-Knotens k_{i+1} . Die ABF des Knotens k_{i+1} startet dann eine Tiefensuche für die n Konjunkte des $\wedge$ -Knotens (wie beschrieben).
- Die ABF eines $\vee$ -Knotens k_i mit n Disjunkten ruft die ABFen der ABP-Knoten k_{i+1} bis einschließlich k_{i+1+n} auf. Die ABFen für die Knoten k_{i+1}

¹⁰Die Operatoren NNF und DNF stellen sicher, dass keine Konjunkte vorkommen, die selbst wieder Konjunktionen sind, und auch keine Disjunkte, die selbst wieder Disjunktionen sind.

5.2 Realisierung der Substrat-Anfragebeantwortungsmaschine

bis k_{i+n} werden jeweils mit der Ergebnistupel-Registrierungsfunktion für *continuation* aufgerufen.

Hierdurch wird die Vereinigungsmenge der Extensionen der Disjunkte berechnet. Die Vereinigung ist wohldefiniert, da durch $\mathcal{N}(q)$ sichergestellt ist, dass alle Disjunkte die gleiche Stelligkeit haben.

Wir sprechen hier von einer *Top Down*-Bearbeitung einer Anfrage, da die Extension eines komplexen Rumpfes durch *sukzessive inkrementelle Berechnung* der Extensionen der Teilrumpfe ermittelt wird. Der *inkrementellen* Top Down-Bearbeitung sind jedoch Grenzen gesetzt, da Rumpfe NAF-negiert sein können. Ist k ein komplexer Rumpf, so kann die Extension von $\backslash k$ nur dann inkrementell bestimmt werden, wenn die Extension von k bereits *vollständig* bekannt ist. Wir können die inkrementelle Berechnung jedoch beibehalten, wenn wir fordern, dass k in NNF und π -frei ist. Das Problem der inkrementellen Generierung von Bindungen für NAF-negierte Rumpfe wird dann den Generatoren bzw. Testern der Atome auferlegt, was wesentlich einfacher zu realisieren ist (s.u.). Dies motiviert nachträglich die Notwendigkeit der Normalform $\mathcal{N}(q)$. Für die *Referenz-Atome*, die ja durch Ersetzung von π -Rümpfen entstanden sind, müssen wir *n-stellige Generatoren und Tester* vorsehen.

5.2.2.3.1. Bestimmung der Anfragebearbeitungsfunktionen und Standard-ABFen

Wie bereits beschrieben, wird mittels *compile-query* für jeden ABP-Knoten seine ABF angefordert, und als Wert des Attributes *query-fn* abgespeichert. Je nach Rolle des Atomes im ABP wird der Anfrageübersetzer daher eine der folgenden Methoden aufrufen, die als Methoden der Klasse *Anfrage (Query)* vorgesehen sind:

- *get-tester-function*,
- *get-generator-function*,
- *get-successor-generator-function*,
- *get-predecessor-generator-function*.

Diese Methoden liefern für Instanzen der entspr. ASG-Klasse passende ABFen als Funktionsobjekte zurück.

Wir sehen Standard-Implementationen dieser Methoden vor, und zwar für die Klassen *Unäres-Atom (Unary-Query)* und *Binäres-Atom (Binary-Query)* (diese Klassen sind abstrakt). Die zurückgelieferten ABFen sind ihrerseits wiederum Schablonenmethoden, nutzen nun also spätes Binden *zur Anfragebeantwortungszeit*. Es ist daher ratsam, diese für spezielle ASG-Klassen zu überschreiben. Die Bereitstellung implementierter ABFen ist selbst für abstrakte Klassen sinnvoll, da sie herausfaktorisierbares Standard-Verhalten für instantiierbare, konkrete Unterklassen bereitstellen.

Diese „Standard-ABFen“ implementieren die *vereinfachte Semantik*, wie in Kap. 4.2.3, S. 208 definiert. Ist die vereinfachte Semantik anwendbar, so kann Folgerbarkeit rein knoten- bzw. kantenlokal entschieden werden: Es gilt dann $S \models (i, v)$ gdw. $L_V(i) \models (i, v)$, und $S \models (i, j, w)$ gdw. $L_E((i, j)) \models (i, j, w)$. Die Standard-ABFen verwenden die *matches-p-Methode*, welche $L_V(i) \models (i, v)$ und $L_E((i, j)) \models (i, j, w)$ implementiert. Sie ist für Argumente vom Typ Knoten (Node) $\times$ Unäres-Atom (Unary-Query) sowie Kante (Edge) $\times$ Binäres-Atom (Binary-Query) definiert.

Die Standard-Implementation von *matches-p* (s. Klasse Substrat-Objekt (Substrate-Object)) geht wie folgt vor. Für gegebenen Knoten i (Instanz der Klasse Knoten (Node)) bzw. gegebene Kante (i, j) (Instanz der Klasse Kante (Edge)) wird $L_V(i)$ bzw. $L_E((i, j))$ ermittelt (Attribut *description*), und dann an *implies-p* delegiert. Die Methode *implies-p* ist für Paare Beschreibung (Description) $\times$ Beschreibung (Description) definiert, und somit auch für Paare Beschreibung (Description) $\times$ Atomare-Anfrage (Atomic-Query).¹¹

Ist also für eine Klasse von Atomen die vereinfachte Semantik anwendbar, so muss zur Implementation der ABFen lediglich *implies-p* für Atome des neuen Typs (zweites Argument der Methode) überschrieben werden.

5.2.2.3.2. Minimierung von Funktionsaufrufen in den ABFen Wir hatten erläutert, dass ABFen sich über Fortsetzungen gegenseitig aufrufen. Hierbei bekommt die ABF des Rumpfes k_i die ABF des Rumpfes k_{i+1} als Argument *continuation* mitübergeben; die ABF von k_i kann dann mittels eines einfachen Funktionsaufrufes (*funcall continuation*) die Abarbeitung des ABPs fortsetzen. Ein normaler Funktionsaufruf ist i.d.R. effizienter als ein Methodenaufruf. Die Forderung, bei Bedarf möglichst viel Laufzeitaufwand einsparen zu können, um eine hocheffiziente Anfragebearbeitung sicherzustellen, führt bei konsequenter Anwendung zu der Idee, nicht nur dynamisches Binden und somit Methodenaufrufe, sondern *auch Funktionsaufrufe wann immer möglich einzusparen*, und auch Berechnungsergebnisse wenn möglich bereits zum Übersetzungszeitpunkt der Anfrage vorwegzunehmen:

- Funktionsaufrufe können eingespart werden, wenn ihre *Implementationen* bzw. *Funktionsbeschreibungen* *ineinander geschachtelt* werden: Benötigt Funktion A die Dienste von Funktion B, so wird statt einem B-Funktionsaufruf innerhalb von A einfach die Implementation von B an die entsprechende Stelle statt des B-Funktionsaufrufs textuell eingesetzt („Code Unfolding“). Dies ist immer dann möglich, wenn keine Rekursion verwendet wird, und ist in unserem Szenario für die ABFen der Fall.

¹¹Die Iteratoren wie *loop-over-matching-nodes* iterieren linear über die entspr. Knoten und verwenden ebenfalls diese *matches-p-Methode* (analog für *loop-over-matching-edges*, *loop-over-matching-successors*, *loop-over-matching-predecessors*).

5.2 Realisierung der Substrat-Anfragebeantwortungsmaschine

Dies führt zur Idee eines *codeschablonenbasierten Übersetzers*. Ein (funcall continuation)-Aufruf wird einfach gegen die Funktionsbeschreibung der gerufenen continuation (i.d.R. eine ABF) ersetzt. Durch Ineinanderschachtelung von ABF-Codeschablonen wird so eine komplexe ABF-Funktionsbeschreibung erzeugt, die keine (funcall continuation)-Aufrufe mehr enthält. Diese wird vom COMMON LISP-Compiler übersetzt (der Teil der Sprache selbst ist). So entsteht aus der ABF-Funktionsbeschreibung ein compiliertes Funktionsobjekt („Closure“), das dann als ABF zurückgeliefert wird. Idealerweise enthält diese ABF auch keine Methodenaufrufe mehr. Man kann sich leicht klarmachen, dass so zur Anfragebeantwortungszeit eine exponentielle Anzahl von (funcall continuation)-Aufrufen eingespart wird (allerdings zu dem Preis, dass die zurückgelieferte ABF für den ASG-Wurzelknoten nun sehr komplex ist und eventuell den COMMON LISP-Compiler überfordert).

- Die Verwendung von Codeschablonen ermöglicht zudem, schon zur Übersetzungszeit bestimmte Bedingungen auszuwerten, ja genereller, beliebige Berechnungen (wenn bereits zum Übersetzungszeitpunkt möglich) durchzuführen. Die Ergebnisse dieser Berechnungen können direkt in der ABF-Funktionsbeschreibung manifestiert werden, was ein Vorteil sein kann (wiederholte Berechnungen werden vermieden, etc.).

Der Anfrageübersetzer soll auch die Idee des codeschablonenbasierten Übersetzers realisieren. Mit einem codeschablonenbasierten Übersetzer kann *Laufzeitaufwand in Übersetzungsaufwand* überführt werden. Die Bereitstellung der Codeschablonen ist jedoch auch komplexer. Zur Definition von ABF-Codeschablonen werden wir eine Makrosprache vorsehen. Wir werden in Kap. 7 evaluieren, ob sich der Aufwand lohnt.

5.2.2.3.3. Inkrementelle (strombasierte) und informierte Generatoren Wir hatten dargestellt, dass die Maschine inkrementelle und daher ressourcenschonende Anfragebeantwortung ermöglichen soll. Es sollen daher nur die tatsächlich angeforderten Tupel bzw. Anfrageergebnisse bei Bedarf berechnet werden. Dies bedeutet insbesondere, dass Generatoren keine nicht erforderliche „Mehrarbeit“ leisten sollten.

Ein Generator enumeriert inkrementell Variablenbindungen. Wir hatten oben schon die stark vereinfachte Sichtweise eines (x, C) -Generators diskutiert:

```
(lambda (continuation)
  (loop-over-nodes (var S)
    (when (matches-p var C)
      (setf (bound-to x) var)
      (funcall continuation)))
  (return))
```

Dieser Generator arbeitet inkrementell, also ressourcenschonend. Offensichtlich leidet dieser Generator jedoch daran, dass er keine Indexstrukturen zur Einschränkung bzw. Ermittlung der zu betrachtenden Kandidatenmenge verwendet. Er ist nicht informiert. Jeder Knoten wird isoliert daraufhin überprüft, ob er C erfüllt. Informierter wäre daher folgender Generator:

```
(lambda (continuation)
  (loop-over-matching-nodes (var S C)
    (setf (bound-to x) var)
    (funcall continuation))
  (return))
```

Die Nutzung von S -spezifischen Indexstrukturen kann dann der Methode `loop-over-matching-nodes` auferlegt werden. Der Iterator ist dann also informiert und inkrementell.

Leider ist ein solch informierter und inkrementeller Generator oftmals nur aus *externer Perspektive* wirklich informiert und inkrementell. Insbesondere für eine ABox-Anfragesprache, die ein externes DL-System wie RACERPRO verwendet, ist ein solcher Generator aufgrund fehlender Schnittstellen bzw. aufgrund des Vorliegens eines *Impedance Mismatches* nicht wirklich realisierbar.

Zur Illustration der Problematik sei S nun eine RACERPRO-ABox. Ist C ein Konzept, so hat (x, C) die Semantik einer ABox-Instanzenretrievalanfrage (s. S. 208). Der erforderliche informierte Iterator `loop-over-matching-nodes` kann nun mittels der vorhandenen RACERPRO-API-Funktionen entweder als Iteration der Art

```
(dolist (var (concept-instances C S)) ...)
```

oder als Iteration der Art

```
(dolist (var (get-abox-individuals S))
  (when (individual-instance-p var C S) ...))
```

realisiert werden.

Im ersten Fall ist der Generator zwar informiert, aber nicht wirklich inkrementell – er leistet Mehrarbeit. Mangels einer inkrementellen Schnittstelle werden evtl. überflüssige C -Instanzen berechnet. Der Generator ist daher nur aus externer Perspektive ressourcenschonend, nicht jedoch aus interner Perspektive.

5.2 Realisierung der Substrat-Anfragebeantwortungsmaschine

Im zweiten Fall ist der Generator zwar ressourcenschonend und auch aus interner Perspektive inkrementell¹², er ist jedoch nicht informiert und kann daher keine Indexstrukturen nutzen, da auch hier eine entsprechende RACER-PRO-API-Funktion fehlt.

Wir sehen daher, dass zur Realisierung von Generatoren mit diesen Eigenschaften eine andere Art der Kopplung mit einem DL-System bzw. andere Schnittstellen erforderlich sind (die „hohe Kohäsion“ ermöglichen). Das MiDeLoRA-DL-System wird die hierfür erforderlichen *inkrementellen* ABox-Basis-Retrieval-Funktionen (s. Kap. 2.2.2.5, S. 78) bereitstellen. Die MiDeLoRA-Generatoren haben dann die geforderten Eigenschaften nicht nur aus externer, sondern auch aus interner Perspektive.

5.2.2.3.4. Zur Notwendigkeit von n -stelligen Generatoren und Testern Zur Realisierung der ABFen der Knoten- und Kanten-Atome sind ausschließlich unäre und binäre Generatoren bzw. Tester erforderlich. Zur Realisierung der ABFen der *Referenz-Atome* müssen jedoch n -stellige Generatoren und/oder Tester implementiert werden; diese sind auch zur Nutzung der QBox-Cacheeinträge, also bei einer erfolgten Anfrageklassifikation, erforderlich (s. Def. 90, S. 227).

Im Falle einer erfolgten QBox-Anfrageklassifikation der Anfrage q soll die gespeicherte Extension p^ε einer ausgewählten Eltern-Anfrage $p \in \text{parents}(q)$ als n -stelliger Generator verwendet werden. Zudem wird ein n -stelliger Tester benötigt, um anhand von p^ε etablierte Variablenbelegungen auf Mitgliedschaft in $\bigcap_{p' \in \text{parents}(q)} p'^\varepsilon$ und auf Nicht-Mitgliedschaft in $\bigcup_{c \in \text{children}(q)} c^\varepsilon$ zu prüfen (so sparen wir uns bereits beim Generieren eine sonst erforderliche Duplikatseliminierung von Tupeln, da ja die Menge $\bigcup_{c \in \text{children}(q)} c^\varepsilon$ unbesehen als Teilergebnis für q^ε übernommen wird).

Da Referenz-Atome auch NAF-negiert auftreten können, z.B. $\setminus(\vec{x}, \text{antwort}_i)$, muss hier auch ein Generator implementiert werden, der das n -dimensionale Komplement von $(\vec{x}, \text{antwort}_i)^\varepsilon$ enumerieren kann.

5.2.2.4. Über die Verwendung von Indexstrukturen

Prinzipiell sind die Generatoren der für die einzelnen ASG-Klassen definierten ABFen dafür verantwortlich, die vom befragten Substrat bereitgestellten Indexstrukturen entsprechend zu nutzen. Entsprechendes „Wissen“ bzw. „In-

¹²Obwohl auch hier schon durch `get-abox-individuals` evtl. *unnötig viele* ABox-Individuen über eine Netzwerkverbindung übertragen werden müssen, was insbesondere bei großen ABoxen zu einer starken Verlangsamung führen kann. Die API-Funktion `get-abox-individuals` ist jedoch im Gegensatz zu `concept-instances` günstig, da keine Inferenz von RACERPRO erforderlich ist.

formiertheit der Generatoren“ ist also hier zu kodieren. Wie erläutert, können die bereitgestellten generischen Standard-ABFen mangels spezifischen Wissens keine dedizierten Indexstrukturen nutzen. Wir erläutern die Nutzung spezieller Indexstrukturen später am Beispiel von NRQL.

Die QBox kann jedoch als eine allgemeine Indexstruktur angesehen werden, die ohne spezifisches Wissen über Indexstrukturen auskommt. Zur Nutzung der QBox muss jedoch die *entails-p*-Methode für die entsprechenden ASG-Klassen definiert sein, sodass Subsumption zwischen Anfragen (wenn auch nicht notwendigerweise für alle Anfragen) entschieden werden kann.

5.2.2.5. Unterstützung hybrider Substrat-Anfragesprachen

Die vorgestellte Anfragebeantwortungsmaschinerie kann auf einfache Weise zur Implementation hybrider Substrat-Anfragesprachen verwendet werden. Eine hybride Substrat-Anfragesprache hatten wir in Kap. 4.2.12, S. 228 als eine Anfragesprache definiert, die es erlaubt, Atome für die verschiedenen Substrate S_1, S_2 eines hybriden Substrates $(S_1, S_2, *)$ in einer Anfrage zu kombinieren. Der Parser muss das *Klassifikationsproblem* lösen können: Für gegebenes Atom muss er entscheiden, ob es sich um ein Atom für S_1 oder für S_2 handelt. Zu diesem Zweck wird eine entsprechende Klassifikationsmethode *get-substrate-for-atom* vorgesehen. Wir hatten in Kap. 4.2.12 entsprechende Disambiguierungstechniken diskutiert (Atome, die Objekte mit „*“-Präfix verwenden, beziehen sich auf S_2).

Um nun **-konsistente Variablenbindungen* (s. Def. 91, S. 229) zu garantieren, wird der Parser für jedes in der Anfrage referenzierte Anfrage-Objekt stets das *korrespondierende Objekt* vermerken (s. Attribut *corresponding-object*, Klasse Anfrage-Objekt (Query-Object)). Beispielsweise werden in der Anfrage $(x, C) \wedge (*x, D)$ die Objekte x und $*x$ als zueinander korrespondierend vermerkt. Wird nun diese Anfrage an ein hybrides Substrat gestellt, und wurde der Plan $\langle (x, C), (*x, D) \rangle$ ausgewählt, so muss der (x, C) -Generator sicherstellen, dass nur solche C -Instanzen als Bindungen für x etabliert werden, für die „*“ definiert ist („*“ ist ja eine partielle Funktion), um eine *-konsistente Variablenbindungen sicherzustellen. Wird x gegen i gebunden, so wird $*x$ automatisch gegen den korrespondierenden Knoten $*(i)$ gebunden. Das Atom $(*x, D)$ hat dann die Rolle eines Testers.

5.2.2.6. Operationale Aspekte: Lebenszyklus von Anfragen und Entwurf eines APIs

Den Anforderungen entsprechend, soll die Maschine Mehrbenutzerbetrieb ermöglichen, eine mengen- und eine tupelbasierte Schnittstelle bereitstellen, und

5.2 Realisierung der Substrat-Anfragebeantwortungsmaschine

eine Lebenszyklus-Verwaltung für Anfragen durchführen („Anfragen als Objekte“). Das API der Maschine wird im mengenbasierten Modus synchron, und im tupelbasierten Modus quasi asynchron operieren. Ein asynchrones, nichtblockierendes API kann softwaretechnisch durch Verwendung von nebenläufigen Konstrukten wie Kontrollfäden (Threads) oder Prozessen realisiert werden.

Oftmals wird nur ein Teil der Antwortmenge einer Anfrage benötigt bzw. anfordert werden. Die Ergebnismenge kann prinzipiell *statisch oder dynamisch begrenzt werden*:

- Die *statische Begrenzung* der Ergebnismenge setzt voraus, dass der Nutzer die maximal geforderte Anzahl an Anfrageergebnissen (Tupeln) im Voraus bekannt gibt; z.B. 20, weil eben nur 20 Treffer auf einer Präsentations-Webseite Platz haben (SWIR-Szenario).
- Die *dynamische Begrenzung* der Ergebnismenge erfordert hingegen, dass der Nutzer die Anfragemenge *Tupel per Tupel* erhält; hierzu ist eine API-Funktion *get-next-tuple* vorzusehen. Durch *Inspektion der bereits erhaltenen Tupel* kann entschieden werden, ob noch weitere Tupel benötigt werden oder nicht („inkrementelles Browsen“).

Da Anfragen in unserem Entwurf als Objekte (Instanzen der Klasse Anfrage (Query)) repräsentiert werden, liegt es nahe, einen *Lebenszyklus von Anfragen* zu verwalten. Der Lebenszyklus einer Instanz der Klasse Anfrage (Query) bzw. eines ASG-Wurzelknotens ist in Abb. 5.17 als UML-Zustandsdiagramm dargestellt. Eine Anfrage kann sich prinzipiell in vier Zuständen befinden:

- vorbereitet (Zustand prepared),
- aktiv (Zustand active),
- beendet (Zustand processed/terminated),
- gelöscht (Zustand deleted).

Der Zustand active hat eine komplexe Substruktur, die wir weiter unten erläutern. Der Zustand einer Anfrage wird durch entspr. Attribute der Klasse Anfrage (Query) verwaltet (s. Attribute processed-p, deleted-p, ...). Entsprechende Methoden der Klasse Anfrage (Query) können verwendet werden, um Anfragen vorzubereiten (Methode prepare), auszuführen (Methode execute), oder zu löschen (Methode delete). Diese Methoden können also *Zustandsübergänge* auslösen. Einige Zustandsübergänge geschehen jedoch auch spontan bzw. durch das Eintreten von Bedingungen; beispielsweise wird eine Anfrage automatisch in den Zustand processed / terminated versetzt, wenn das processed-p-Attribut der Instanz auf TRUE gesetzt wird. Dies ist z.B. der Fall, wenn keine weiteren Tupel mehr berechnet werden können.

Eine Anfrage bekommt eine eindeutige symbolische Identifikation (Attribut id). Der Identifikator (ID) kann in allen API-Funktionen der Maschine zur Refe-

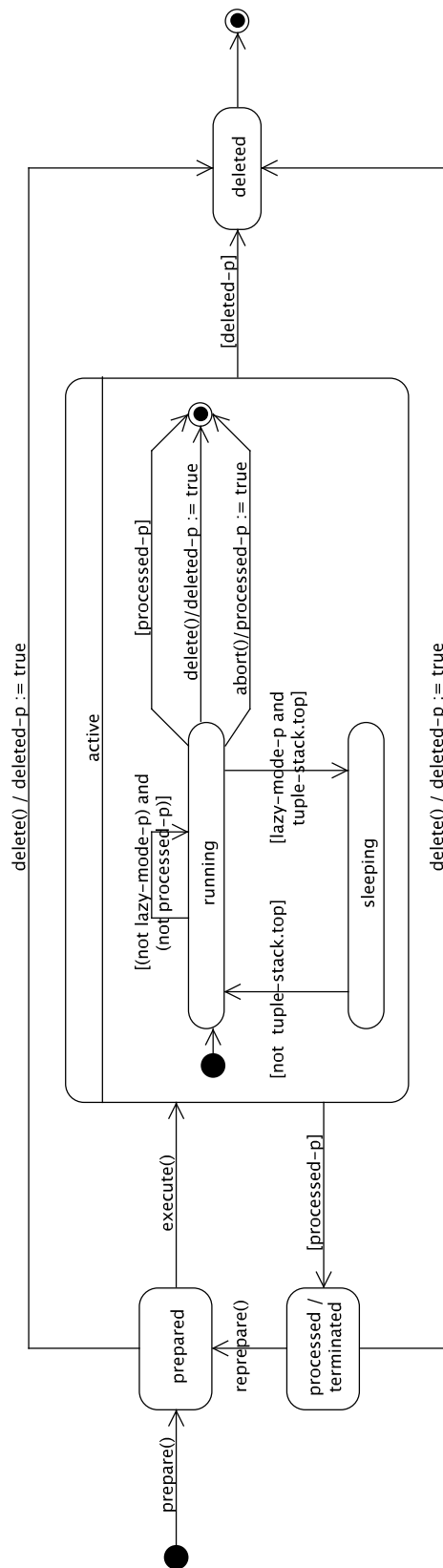


Abbildung 5.17.: Der Lebenszyklus einer Anfrage.

5.2 Realisierung der Substrat-Anfragebeantwortungsmaschine

renzierung der Anfrage verwendet werden. Der sog. Anfragen-Verwalter (Query-Manager) ist für die Lokalisierung der entspr. Instanz (Methode `find-query`) und Vergabe der IDs verantwortlich. Zudem verwaltet der Anfragen-Verwalter (Query-Manager) Listen von *vorbereiteten, aktiven, und beendeten Anfragen* (s. Assoziationen `prepared-queries`, `active-queries`, `processed-queries`). Eine Anfrage ist immer nur Element einer dieser Listen. Der Anfragen-Verwalter (Query-Manager) stellt auch das externe API der Maschine bereit. Die meisten dieser API-Funktionen delegieren einfach an die entsprechenden Methoden der Klasse `Anfrage` (Query) (sofern die Anfrage-Instanz bereits existiert), nachdem die (evtl. namentlich referenzierte) Anfrage-Instanz per `find-query` ermittelt wurde.

Das API der Maschine umfasst folgende Hauptfunktionen:

- Mit (`prepare-query <kopf> <rumpf> &key <id> <substrate> ...`)¹³ wird eine Anfrage vorbereitet.

Die Anfrage bekommt (sofern nicht bereits vom Nutzer mittels `<id>` als Argument explizit vorgegeben) einen Anfrage-Identifikator. Das zu befragende Substrat wird entweder explizit durch Angabe des Argumentes `<substrate>` (evtl. namentlich) spezifiziert; andernfalls bezieht sich die Anfrage auf das *aktuelle Substrat*. Der Anfragen-Verwalter (Query-Manager) beauftragt den Substrat-Verwalter (Substrate-Manager), das (evtl. namentlich) referenzierte Substrat zu ermitteln. Die Methode `get-parser-class-for-substrate` ermittelt eine für diesen Typ von Substrat passende Parser-Klasse und erzeugt eine entspr. Parser-Instanz. Der erzeugte Parser wird durch Aufruf seiner `parse-query`-Methode beauftragt, den ASG für die Anfrage `<kopf>`, `<rumpf>` zu konstruieren.

Alsdann wird die `prepare`-Methode (s. Klasse `Anfrage` (Query)) auf den zurückgelieferten ASG angewendet; die `prepare`-Methode der Klasse löst die in Kap. 5.2.2.1 beschriebenen Vorgänge (`optimize`, `compile`, ...) aus. Schließlich wird der Anfrage-Identifikator zurückgegeben, und die Anfrage wird vom Anfragen-Verwalter (Query-Manager) auf der Liste der vorbereiteten Anfragen vermerkt. Die Anfrage befindet sich nun im Zustand `prepared`.

- Mittels (`execute-query <query-id>`) wird eine vorbereitete Anfrage ausgeführt. Hier wird letztlich an die Methode `execute` der Klasse `Anfrage` (Query) delegiert. Die Anfrage wechselt dann von der Liste der vorbereiteten Anfragen auf die Liste der aktiven Anfragen, bzw. vom Zustand `prepared` in den Zustand `active`. Vorbereiten und Ausführen einer Anfrage kann auch in einem einzigen (unteilbaren) Schritt mittels

¹³Hierbei sind `<kopf>` und `<rumpf>` obligatorische, und `<id>`, `<substrate>` optionale Argumente.

answer-query (gleiche Signatur wie prepare-query) geschehen.

Um Mehrbenutzerbetrieb und das für die Inkrementalität benötigte asynchrone API zu realisieren, wird zur Ausführung der ABF der Anfrage ein neuer COMMON LISP-Prozess (vergleichbar einem Thread) gestartet.¹⁴ Diesen COMMON LISP-Prozess nennen wir im Folgenden den *ABF-Prozess*. Eine Anfrage kann nun in zwei verschiedenen Modi ausgeführt werden:

1. Im *mengenbasierten, synchronen Modus* kehrt der Aufruf von execute-query mit der Antwortmenge zurück. Das API ist solange blockiert – dieser Modus ermöglicht also keinen Mehrbenutzerbetrieb. Prinzipiell müssten wir hier *keinen* neuen ABF-Prozess starten.
2. Im *tupelbasierten (inkrementellen), asynchronen Modus* verhält sich das API anders: Hier kehrt der Aufruf von execute-query *sofort* zurück, und es wird lediglich der Anfrage-Identifikator zurückgegeben, nicht das Anfrageergebnis. Das API ist also wieder verfügbar. Im Hintergrund wird nun das Anfrageergebnis vom gestarteten ABF-Prozess berechnet. Ein neu berechnetes Ergebnistupel wird vom ABF-Prozess stets als oberstes Element eines *Ergebnistupel-Stapelspeichers* (kurz: *Ergebnisstapels*) in der Anfrage-Instanz abgelegt (s. Attribut tuple-stack der Klasse Anfrage (Query)).

API-Funktionen wie (get-next-tuple <query-id>) können nun verwendet werden, um einzelne Tupel von diesem Stapelspeicher anzufordern, auch während der ABF-Prozess noch aktiv ist. Ein einmal angefordertes Tupel wird vom Ergebnisstapel entfernt.¹⁵ Ist der Stapel jedoch gerade leer, so kann get-next-tuple erst dann zurückkehren, wenn ein neues Tupel vorliegt (hier ist also Prozesskommunikation erforderlich). Wir sehen auch eine API-Funktionen get-next-n-tuples vor, die die nächsten n Tupel anfordert, indem n mal get-next-tuple gerufen wird.

Aufgrund der ABF-Prozesse können also mehrere aktive Anfragen gleichzeitig inkrementell bearbeitet werden (entsprechende Synchronisation beim Zugriff auf gemeinsam genutzte Ressourcen natürlich vorausgesetzt).

Im tupelbasierten (inkrementellen) Modus unterscheiden wir zudem noch, ob die Anfrage *verzögert bzw. faul (lazy)* oder *fleißig (eager)* ausgeführt wird (s. Attribut lazy-mode-p in Anfrage (Query)). Diese Unterscheidung wird insbesondere durch die Entwurfsanforderung res-

¹⁴Diese werden in einem *Prozess-Pool* vorgehalten.

¹⁵Das vollständige Anfrageergebnis steht nach Terminierung des ABF-Prozesses dennoch zur Verfügung (sofern die Anfrage nicht abgebrochen wurde); hierzu werden die Tupel in einem zusätzlichen listenwertigen Attribut result der Anfrage vermerkt.

5.2 Realisierung der Substrat-Anfragebeantwortungsmaschine

sourcenschonende Anfragebeantwortung motiviert:

- Im *fleißigen Modus* wird der ABF-Prozess ohne Unterbrechungen die gesamte Antwortmenge im Voraus berechnen, auch wenn diese Tupel gar nicht angefordert wurden bzw. angefordert werden. Der fleißige Modus berechnet als u.U. zu viele Tupel.

Eine solche Anfrage terminiert stets automatisch (nachdem das letzte Ergebnistupel berechnet wurde).

- Im *faulen Modus* wird der ABF-Prozess nach Berechnung und Ablegung eines neuen Ergebnistupels auf dem Stapel stets *schlafen gelegt*. Der Prozess wird erst wieder aufgeweckt, wenn das nächste Tupel angefordert wird. Auf diese Art wird eine Berechnung bei Bedarf ermöglicht.

Die Liste der *aktiven Anfragen* wird daher in die Liste der *schlafenden* und die Liste der *laufenden Anfragen* partitioniert. Diese entsprechen den Zuständen *running* und *sleeping* in Abb. 5.17. Der Übergang vom Zustand *running* in den Zustand *sleeping* geschieht gemäß Abb. 5.17, wenn die Anfrage im faulen Modus läuft (*lazy-mode-p* = TRUE) und ein neue Element auf dem Stapel abgelegt wurde (*tuple-stack.top* = TRUE). Anfragen, die im fleißigen Modus gestartet wurden, bleiben hingegen bis zur vollständigen Abarbeitung (*processed-p* = TRUE) im Zustand *running*.

Da u.U. nicht alle Tupel angefordert werden, wird der Prozess i.d.R. nicht automatisch terminieren (es sei denn, die Antwortmenge ist leer). Es können daher auch *Timeouts* angegeben werden; prinzipiell kann eine Anfrage einfach durch Setzen des Attributes *processed-p* auf TRUE dazu gebracht werden, zu terminieren. Zudem kann eine Anfrage mittels *abort-query* manuell beendet oder gar mittels *delete-query* beendet und in den Zustand *deleted* versetzt werden.

Die Funktion *get-next-tuple* ist die zentrale API-Funktion zur Berechnung der Ergebnismengen. Sie wird von allen Funktionen gerufen, die Teilmengen von der Ergebnismenge oder gar die gesamte Ergebnismenge anfordern.

- Eine (wie auch immer) beendete Anfrage wird automatisch auf die Liste der beendeten Anfragen transferiert bzw. in den Zustand *processed* / *terminated* versetzt. Das Anfrageergebnis kann nun beliebig oft mittels *get-answer* (Klasse *Anfrage (Query)*) abgerufen werden (s. Attribut *result*).

Eine beendete Anfrage kann mittels *reprepare-query* auch erneut auf

die Liste der vorbereiteten Anfragen transferiert bzw. in den Zustand `prepared` und dann erneut mittels `execute` zur Ausführung gebracht werden (dies kann wiederum auch in einem Schritt geschehen). Eine Neuausführung kann sinnvoll sein, falls sich das befragte Substrat zwischenzeitlich geändert haben sollte, sodass ein verändertes Anfrageergebnis zu erwarten ist.

- Eine Anfrage kann in jedem Zustand mittels `delete-query` gelöscht und ihr Lebenszyklus somit beendet werden. Sie ist dann im Zustand `deleted` und kann (namentlich) nicht mehr referenziert werden; `find-query` findet diese Anfrage nicht mehr. Die Anfrage wird jedoch u.U. noch durch die `QBox` referenziert (wo sie lediglich als Cache dient).

Natürlich bietet das API auch Funktionen, um den Kopf oder Rumpf einer Anfrage zu erhalten, den Status einer Anfrage zu prüfen, die Listen der vorbereiteten, aktiven, und beendeten Anfragen zu erfragen, etc. Die meisten dieser API-Funktionen werden vom Anfragen-Verwalter (Query-Manager) einfach an entspr. Methoden (meist der Klasse `Anfrage (Query)`) weiterdelegiert.

Eine weitere Gruppe von API-Funktionen wird benötigt, um die Maschine in die entsprechenden Anfragebearbeitungsmodi zu bringen, die Anfrage-Optimierung(en) an- und auszuschalten, etc. Es sind also modusbestimmende API-Funktionen notwendig.

Es wird nun noch eine Gruppe von API-Funktionen zur Inanspruchnahme und Verwaltung der `QBox`-Funktionalität benötigt. Hier kann die `QBox`-Funktionalität ein- bzw. ausgeschaltet werden, Anfragen können klassifiziert werden, die Eltern- und Kinder-Anfragen sowie äquivalente Anfragen einer Anfrage können erfragt werden, etc.

5.2.2.7. Konkrete Syntax

Wir verwenden die klassische Lisp-Präfix-Syntax als konkrete Syntax für die Anfragesprache.

Variablen werden mit `?`-Präfix notiert. Wir sehen vier Arten von Variablen vor, die sich anhand ihrer Präfixe unterscheiden; am Beispiel eine Variablen `x`:

- `?x`: Wir sprechen von einer *injektiven Variablen* und fordern, dass die Bindungsfunktion, die solche injektiven Variablen auf die durch sie gebundenen Substrat-Knoten abbildet, injektiv ist. Zwei Variablen `?x` und `?y` können in einer Konjunktion daher nicht an den gleichen Substrat-Knoten gebunden werden. Es ist offensichtlich, dass Injektivität durch Ungleichheits-Atome in der Anfrage erzwungen werden kann.
- `$?x`: Eine „normale“ Variable (wie bisher verwendet). Die Bindungsfunkti-

5.2 Realisierung der Substrat-Anfragebeantwortungsmaschine

on kann also beliebig sein.

- $?*x$: Die zur injektiven Variablen $?x$ korrespondierende injektive Variable in einer hybriden Anfrage.
- $\$?*x$: Wie $?*x$, doch nicht injektiv.

Individuen bzw. Substrat-Knoten werden einfach literal verwendet: In einer Anfrage referenziert i den Substrat-Knoten i . In hybriden Anfrage referenziert $*i$ den Substrat-Knoten i in S_2 , und i den Substrat-Knoten i in S_1 .

Atome werden in der konkreten Syntax $(?x \ C)$, $(?x \ ?y \ R)$ etc. notiert.

Die Atome $(x, i, \sqsubseteq)$, $(x, \top)$ und $(x, \perp)$ gehören zum Kern der Substrat-Anfragesprache. Ihre konkrete Syntax ist durch $(\text{same-as } ?x \ i)$, $(\text{top } ?x)$ und $(\text{bottom } ?x)$ gegeben. Wir verwenden bewusst die Schreibweise $(\text{top } ?x)$ statt $(?x \ \text{top})$, um den besonderen Status und Typ dieses Atomes deutlich zu machen (analog für bottom).¹⁶ Referenzatome sind nicht Teil der Nutzersprache; dementsprechend existiert keine konkrete Syntax.

Die konkrete Syntax für Rümpfe ist ebenfalls nicht überraschend. Wir verwenden

- and für $\wedge$,
- union für $\vee$,
- neg für $\setminus$,
- pi, project-to für π .

Tatsächlich werden auch noch andere Schlüsselworte akzeptiert, z.B. or und not für $\vee$ und $\setminus$.

Die Anfrage

$$(x, C \vee D) \wedge (\setminus(i, E) \vee (i, F)) \wedge \pi(y)((y, E \wedge F) \wedge (x, y, R))$$

mit $x, y \in Vars$ und $i \in V$ lautet in konkreter Syntax dann

$$(\text{and } (\$?x \ (\text{:or } C \ D)) \ (\text{union } (\text{neg } (i \ E)) \ (i \ F)) \\ (\text{project-to } (\$?y) \ (\text{and } (\$?y \ (\text{:and } E \ F)) \ (\$?x \ \$?y \ R))))$$

Wir stellen schließlich noch konkrete Syntax für die Anfrage-Idiome (s. Kap. 4.2.9) bereit:

- $(?x \ (\text{has-known-successor } R)) =_{def} (\text{project-to } (?x) \ (?x \ ?y \ R))$
- $(?x \ \text{NIL } R) =_{def} (\text{neg } (?x \ (\text{has-known-successor } R)))$

¹⁶Im Falle der ABox-Anfragesprache wären $(?x \ \text{top})$, $(?x \ \text{bottom})$ Instanz einer anderen Klasse, s.u.

- $(NIL \text{ ?}x \ R) =_{def} (\text{neg } (?x (\text{has-known-successor } (\text{inv } R))))$
- $(\text{bind-individual } betty) =_{def} (\text{same-as } \$?betty \ betty)$, wobei $\$?betty$ die repräsentative Variable (s. Def. 82, S. 219) für $betty$ ist.

5.2.2.8. Bereitgestellte Standard-Atome

Ebenso wie das Rahmenwerk funktionsfähige Standard-Substrat- und Standard-Beschreibungsklassen bietet (s. Kap. 5.1.2.6), wird nun auch ein Standard-Angebot an Atomen bereitgestellt, welche unmittelbar zur Formulierung von Anfragen an Instanzen der Standard-Substrate genutzt werden können. Diese Atome werden dementsprechend „passend“ zum bereits vorgestelltem Angebot entworfen.

Wir stellen nun

- einfache Atome
(Klasse Einfaches-Beschreibungs-Atom (Simple-Description-Query)),
- RACERPRO-Beschreibungs-Atome
(Klasse RACERPRO-Beschreibungs-Atom (RACERPRO-Description-Query)), und
- λ -Atome (Klasse Lambda-Atom (Lambda-Query))

bereit; s. UML-Klassendiagramm Abb. 5.18. Für alle drei Arten von Atomen sind die bereitgestellten Standard-ABFen (s. Kap. 5.2.2.3.1) anwendbar. Insofern muss lediglich die Methode `implies-p` entsprechend überschrieben werden.

5.2.2.8.1. Einfache Atome Bei den *einfachen Atomen* (Klasse Einfaches-Beschreibungs-Atom (Simple-Description-Query)) handelt es sich um Atome, die *einfache Beschreibungen* (s. Kap. 5.1.2.5, Klasse Einfache-Beschreibung (Simple-Description)) für $\mathcal{L}_{QE}$ und $\mathcal{L}_{QV}$ verwenden. Diese Atome können zur Befragung eines Substrates verwendet werden, welches ebenfalls einfache Beschreibungen für $\mathcal{L}_V$, $\mathcal{L}_E$ verwendet. Die Klasse Einfaches-Beschreibungs-Atom (Simple-Description-Query) ist jedoch abstrakt (wie auch Einfache-Beschreibung (Simple-Description)).

Durch weitere Diskriminierung nach konkreter Art der Beschreibung (s. Unterklassen von Einfache-Beschreibung (Simple-Description)) und Stelligkeit der Beschreibung (s. Unterklassen von Unäres-Atom (Unary-Query), Binäres-Atom (Binary-Query)) erhalten wir durch Nutzung von Mehrfachvererbung spezialisierte instantiierbare Knoten- und Kanten-Atome, wie z.B. Einfaches-Konjunktives-Knoten-Atom (Simple-Con.-Descr.-Node-Query), Einfaches-Disjunktives-Knoten-Atom (Simple-Dis.-Descr.-Node-Query), etc. Die in Abb. 5.18 dargestellten Unterklassen von Einfaches-Beschreibungs-Atom (Simple-Description-Query) sollten selbsterklärend sein.

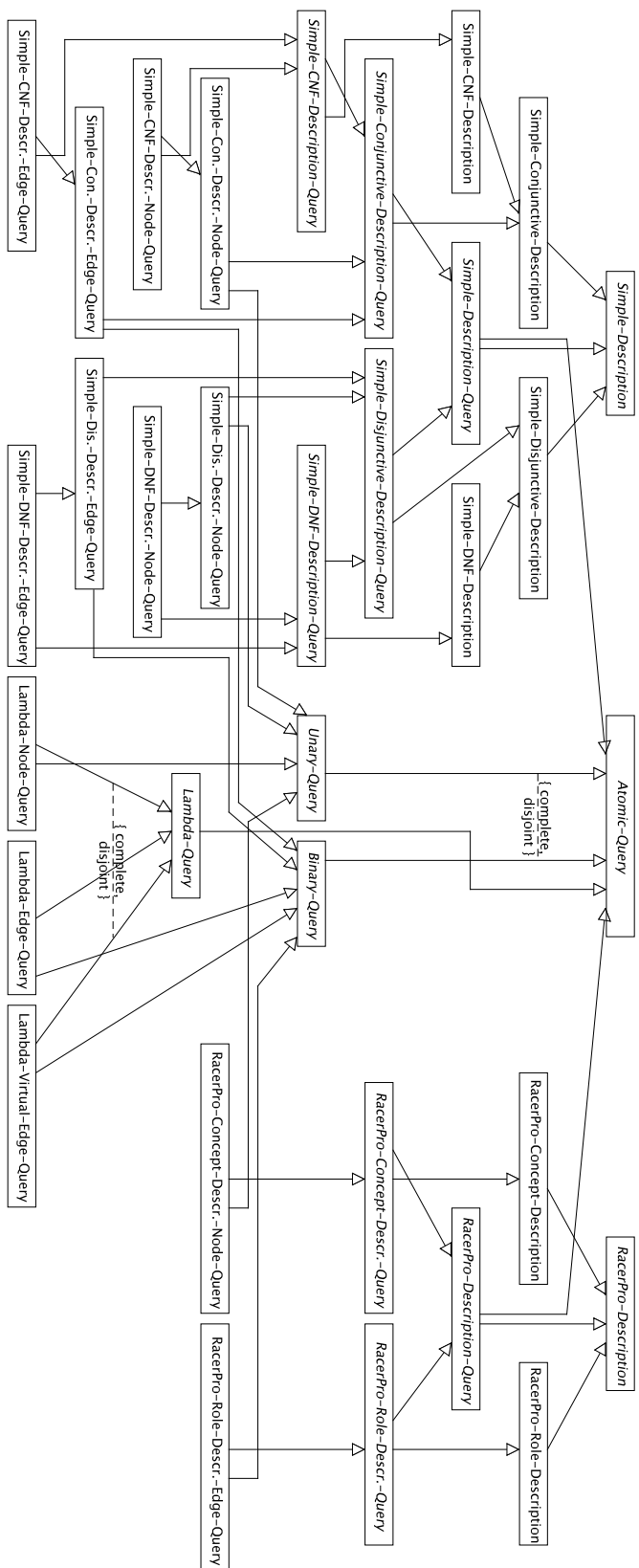


Abbildung 5.18.: Die Hierarchie der Standard-Atome.

Die instantiierbaren (konkreten) einfachen Atome sind wie dargestellt auch von Einfache-Beschreibung (Simple-Description) abgeleitet. Da die *implies-p*-Methode für (instantiierbare) Paare vom Typ Einfache-Beschreibung (Simple-Description) $\times$ Einfache-Beschreibung (Simple-Description) bereits definiert ist, kann die gesamte benötigte Funktionalität zur Auswertung dieser Atome ererbt werden.

Die konkrete Syntax der einfachen Knoten-Atome ist nicht überraschend, und sieht in Beispielen wie folgt aus:

- $(?x \text{ (:and } C1 \dots Cn)),$
- $(?x \text{ (:or } C1 \dots Cn)),$
- $(?x \text{ (:and (:or } C1 \ D1) \dots (:or \ Cn \ Dn))),$
- $(?x \ C1 \dots Cn)$ wird zudem als Kurzform für $(?x \text{ (:and } C1 \dots Cn))$ akzeptiert.

Für die einfachen Kanten-Atome verwenden wir die gleiche Syntax, also z.B. $(?x \ ?y \text{ (:and } R1 \dots Rn)).$

Zur Illustration der Semantik der Atome betrachten wir das (prädikatenlogisch notierte) einfache Substrat $S = \{(C \wedge D \wedge E)(i), R(i, j)\}$. Es gilt dann:

- $(x, C \wedge D)^{\mathcal{E}} = \{i\},$
- $(x, E \vee D)^{\mathcal{E}} = \{i\},$ und
- $(x, y, R \vee S)^{\mathcal{E}} = \{(i, j)\}.$

Offensichtlich können diese Atome auch sinnvoll zur Befragung einer Instanz der Klasse Daten-Substrat (Data-Substrate) verwendet werden. Wir werden unten jedoch eine erweiterte Anfragesprache speziell für das Daten-Substrat (Data-Substrate) vorstellen, die auch *Prädikate* bietet.

5.2.2.8.2. RACERPRO-Beschreibungs-Atome Die RACERPRO-Beschreibungs-Atome (Klasse RACERPRO-Beschreibungs-Atom (RACERPRO-Description-Query)) werden verwendet, um ein RACERPRO-DL-Beschreibungs-Substrat (RACERPRO-DL-Description-Substrate) zu befragen. Bei diesem handelt es sich ja um eine „Pseudo- bzw. Ersatz-ABox“, in der Knoten und Kanten mit RACERPRO-Konzeptausdrücken bzw. RACERPRO-Rollenausdrücken, also Beschreibungsinstanzen der Klasse RACERPRO-Beschreibung (RACERPRO-Description) annotiert sind. Wiederum ist die benötigte Implikationsmethode *implies-p* bereits definiert, da die Atome als Unterklassen der Klasse RACERPRO-Beschreibung (RACERPRO-Description) angelegt sind.

Die konkrete Syntax dieser Atome sieht am Beispiel wie folgt aus:

- $(?x \text{ (:racer } C)),$ wobei C ein RACERPRO-Konzept ist.

5.2 Realisierung der Substrat-Anfragebeantwortungsmaschine

- $(?x ?y (:racer D))$, wobei D ein RACERPRO-Konzept ist.
- $(?x ?y (:racer R))$, wobei R ein RACERPRO-Rollenterm ist.

Prinzipiell wäre es natürlich sinnvoller, hier ein DL-System zu verwenden, welches keine ABoxen bietet (etwa FACT).

5.2.2.8.3. λ -Atome λ -Atome (s. abstrakte Klasse Lambda-Atom (Lambda-Query)) erlauben die Nutzung der Wirtssprache COMMON LISP zur Definition *beliebiger Anfrageprädikate*. Diese Anfrageprädikate werden als COMMON LISP-Funktionsbeschreibungen (also λ -Ausdrücke) in den Atomen notiert. Die Funktionen müssen TRUE oder FALSE zurückliefern, also Prädikate sein (in COMMON LISP kann ohnehin jeder Wert als Wahrheitswert interpretiert werden).

Konzeptionell könnte man sagen, dass die Sprachen $\mathcal{L}_{QV}$ und $\mathcal{L}_{QE}$ nun durch *volle (berechnungsuniverselle) Prädikatenlogik* gegeben sind. Wir müssen also unseren formalen Beschreibungsrahmen nicht verlassen. Einfacher und treffender ist es hier jedoch, die Mengen $\mathcal{L}_{QV}$ und $\mathcal{L}_{QE}$ als Mengen von λ -Ausdrücken zu begreifen, bzw. als Mengen von COMMON LISP-Funktionsobjekten. Die Funktionsobjekte in $\mathcal{L}_{QV}$ haben ein formales Argument, und die Funktionsobjekte in $\mathcal{L}_{QE}$ haben entweder ein oder zwei formale Argumente.

Offensichtlich kann nun jedes Atom *seine* Folgerbarkeitsrelation bzw. Implikationsbeziehung lokal durch Angabe einer entsprechenden COMMON LISP-Funktionsbeschreibung selbst spezifizieren (die Implikationsbeziehung *implies-p* wird nun also nicht mehr klassenglobal festgelegt).

Wir sehen folgende λ -Atome vor:

1. Klasse Lambda-Knoten-Atom (Lambda-Node-Query),
 $(x, \lambda(node) \bullet P)$, konkrete Syntax: $(?x (:predicate <P>))$,
2. Klasse Lambda-Kanten-Atom (Lambda-Edge-Query),
 $(x, y, \lambda(edge) \bullet P)$, konkrete Syntax: $(?x (:predicate <P>))$,
3. Klasse Lambda-Virtuelles-Kanten-Atom (Lambda-Virtual-Edge-Query),
 $(x, y, \lambda(from, to) \bullet P)$, konkrete Syntax: $(?x ?y (:satisfies <P>))$.

Hierbei ist P eine beliebige Funktionsbeschreibung, die die formalen Argumente des λ -Ausdrucks nutzt. Die Semantik dieser Atome ergibt sich (unter Verwendung einer vereinfachten Schreibweise) nun als

1. $(x, \lambda(node) \bullet P)^{\mathcal{E}} =_{def} \{ i \in V \mid (\lambda(node) \bullet P)(i) = \text{TRUE} \}$,
2. $(x, y, \lambda(edge) \bullet P)^{\mathcal{E}} =_{def} \{ (i, j) \in E \mid (\lambda(edge) \bullet P)((i, j)) = \text{TRUE} \}$,
3. $(x, y, \lambda(from, to) \bullet P)^{\mathcal{E}} =_{def} \{ (i, j) \in V \times V \mid (\lambda(from, to) \bullet P)(i, j) = \text{TRUE} \}$.

Intuitiv liefert 1. also alle Knoten i , für die der λ -Ausdruck zu TRUE evaluiert, wenn er auf i angewendet wird. Im Falle von 2. wird der λ -Ausdruck nicht auf

einen Knoten i , sondern auf eine *explizit vorhandenen Kante* $(i, j) \in E$ (aus der Kantenmenge E) angewendet. Im 3. Fall werden auch solche Paare von Knoten i, j betrachtet, zwischen denen *keine* explizite Kante verläuft. Der entsprechende λ -Ausdruck wird daher auf beide Knoten angewendet – die Kante (i, j) ist also in gewisser Weise *virtuell*. Mit diesen Atomen könnten wir beispielsweise für Instanzen der Klasse Geometrisches-Substrat (Geometric-Substrate) die besprochene Auswertung von räumlichen Relationen zur Anfragebeantwortungszeit durch Inspektionsmethoden veranlassen. Hier wird P einfach eine entsprechende COMMON LISP-Funktion aufrufen, die die entspr. räumliche Beziehung anhand der Geometrie der Knoten prüft.

Für die Klassen Lambda-Knoten-Atom (Lambda-Node-Query) und Lambda-Kanten-Atom (Lambda-Edge-Query) müssen keine neuen ABFen implementiert werden. Die Standard-ABFen verwenden die Methode `matches-p` zur Prüfung, ob ein Knoten i bzw. eine Kante $(i, j) \in E$ Treffer eines Knoten- bzw. Kanten-Atoms ist. Wir überschreiben `matches-p` für diese λ -Atome nun einfach so, dass das im Atom gespeicherte Funktionsobjekt auf den Knoten bzw. die Kante angewendet und das erhaltene Ergebnis zurückgegeben wird (`implies-p` wird nicht verwendet).

Für Instanzen der Klasse Lambda-Virtuelles-Kanten-Atom (Lambda-Virtual-Edge-Query) ergibt sich das Problem, dass die Kante (i, j) ja gar nicht explizit vorhanden ist; die Standard-ABF für Kanten-Atome überprüft jedoch nur explizit vorhandene Kanten. Wir müssen daher die ABFen für solche Atome neu definieren, sodass beliebige Knotenpaare $V \times V$ überprüft werden.

Prinzipiell sind Anfragen mit λ -Atomen nun nicht mehr automatisch *sicher*, da λ -Atome berechnungsuniversell (Turing-vollständig) sind. Terminierung ist daher nicht automatisch gewährleistet. Natürlich ist auch Inferenz mit derartigen Atomen i.Allg. unentscheidbar. Daher sind die semantischen Anfrage-Optimierungsmöglichkeiten für diese Atome nicht anwendbar, und auch eine Indizierung ist kaum möglich. Zudem können diese Atome nur von Nutzern mit COMMON LISP-Programmierkenntnissen verwendet werden. Trotz dieser Nachteile stellen diese Atome ein sehr mächtiges Implementierungswerkzeug zur Realisierung spezialisierterer Atome bereit. Wir werden die λ -Atome später nutzen, um bestimmte räumliche Atome für die DISKQL zu realisieren. Durch Anfrageumschreibung werden diese Atome dann auf λ -Atome abgebildet werden.

5.2.3. Objektorientierte Implementation

Die Implementation wurde gemäß dem Entwurf vorgenommen. Alle Schritte der Anfragebearbeitungskette wurden als entsprechende Schablonenmethoden wie im Entwurf beschrieben realisiert.

5.2 Realisierung der Substrat-Anfragebeantwortungsmaschine

Zur *zusammengehörigen Definition* der verschiedenen benötigten ABFen für eine Klasse von Atomen wurde eine spezielle COMMON LISP-Makrosprache geschaffen. Die Makrosprache ermöglicht die zusammengehörige Definition der ABFen für alle möglichen Rollen (Generator, Tester, ...) einer Klasse von Atomen, als auch die Definition von ABF-Codeschablonen für den codeschablonenbasierten Anfrageübersetzer, s.S. 297, sowie die Definition von ABFen als „normale Funktionen“. Wir geben zur Illustration die Implementierung der Standard-ABFen in dieser Makrosprache an.

Natürlich wurden auch die ABFen für die anderen benötigten Basis-Atome der Kernsprache implementiert (same-as etc.), als auch die ABFen für die komplexen Rumpfe. Deren Implementation wurde wie in Kap. 5.2.2.3, S. 293 beschrieben realisiert.

5.2.3.1. Implementation der Anfragebearbeitungsfunktionen

Wir hatten dargestellt, dass der Anfrageübersetzer (die compile-Methode) jedem ABP-Knoten als Wert des Attributes query-fn eine ABF zuweist, die u.U. durch einen codeschablonenbasierten Expansions- und Einsetzungsprozess („Code Unfolding“) entstanden ist. Die Makrosprache unterstützt nun die zusammengehörige Definition der vom Übersetzer durch die Methoden

- get-tester-function,
- get-generator-function,
- get-successor-generator-function, und
- get-predecessor-generator-function

angeforderten ABF-Funktionsobjekte für eine Klasse von Atomen mittels des defquery-code-Makros. Der Übersetzer soll die durch diese Methoden zur Übersetzungszeit gelieferten Funktionsobjekte entweder

- anhand von kompilierten Codeschablonen erzeugen, mit oben diskutierten Vorteilen (zur Anfragebeantwortungszeit stattfindende Funktionsaufrufe können durch textuelle Schachtelung zur Kompilationszeit eingespart werden), oder aber
- direkt als normale Funktionsobjekte bereitstellen (die sich zur Anfragebeantwortungszeit mittels (funcall continuation) fortsetzen).

Das Makro defquery-code bietet hierfür entsprechende :compiler- und :runtime-Abschnitte. Für die ASG-Klasse Unäres-Atom (Unary-Query) sieht die gemeinsame Definition der generischen Standard-ABFen dann wie folgt aus:

```
(defquery-code (unary-query)

  (:tester
   (:compiler (body &rest args &key var &allow-other-keys)
    (declare (ignorable args))
    (with-slots (negated-p voi) query
      '(', (if negated-p
                'unless
                'when)
            (matches-p ,query ,(or var '(bound-to ,voi)))
            ,body)))
   (:runtime
    (continuation &rest args &key var &allow-other-keys)
    (with-slots (negated-p voi) query
      (let ((var (or var (bound-to voi))))
        (if negated-p
            (unless (matches-p query var )
                    (apply continuation :var var args))
            (when (matches-p query var)
                    (apply continuation :var var args)))))))

  (:enumerator
   (:compiler
    (body &rest args &key var &allow-other-keys)
    (declare (ignorable args))
    (with-slots (negated-p substrate) query
      (if negated-p
          '(', (loop-over-nodes (,var ,substrate)
                                (unless (matches-p ,query ,var)
                                      ,body))
              '(', (loop-over-nodes (,var ,substrate)
                                    (when (matches-p ,query ,var)
                                          ,body))))))
   (:runtime
    (continuation &rest args &key &allow-other-keys)
    (with-slots (negated-p substrate) query
      (if negated-p
          (loop-over-nodes (var substrate)
                          (unless (matches-p query var)
                                (apply continuation :var var args)))
          (loop-over-nodes (var substrate)
                          (when (matches-p query var)
                                (apply continuation :var var args)))))))
```

5.2 Realisierung der Substrat-Anfragebeantwortungsmaschine

Die wesentlichen Punkte sind hier, dass für die beiden möglichen Rollen des Atoms, `:tester` und `:generator`, jeweils eine `:runtime`- und eine `:compiler`-Implementierung angegeben werden. Die `:compiler`-Implementierung stellt eine Codeschablone bereit, während die `:runtime`-Implementierung als „normale Funktionsbeschreibung“ zu deuten ist.

Es wird deutlich, wie die `matches-p`-Methode verwendet wird, und dass ein NAF-negiertes Atom einfach durch `(not (matches-p ...))` implementiert wird, genau so wie es die Definition der Semantik verlangt.

Aufgrund der Generizität ist der Generator zudem nicht informiert; d.h., er verwendet lediglich den Standard-Iterator `loop-over-nodes` statt `loop-over-matching-nodes`. Wir hatten die Problematik in Kap. 5.2.2.3.3, S. 298 diskutiert.

In den Codeschablonen verwenden wir den „Backquote-Comma“-Mechanismus aus COMMON LISP, um konstante und variable Teile der Codeschablone zu beschreiben. Beispielsweise liefert die Auswertung der Schablone

```
'(,(if negated-p
      'unless
      'when)
  (matches-p ,query ,(or var '(bound-to ,voi)))
 ,body)
```

den Ausdruck

```
(unless (matches-p (?x C) x)
  (princ "test")),
```

falls `negated-p` zu `TRUE` evaluiert, `body` an den Ausdruck `(princ "test")`, `var` gegen `x`, und `query` gegen `(?x C)` gebunden ist.

5.2.3.2. Bereitstellung einer neuen Substrat-Anfragesprache im Rahmenwerk

Zur Illustration der Generizität des entworfenen und implementierten Rahmenwerks fassen wir die notwendigen Implementierungstätigkeiten zur Realisierung einer neuen Substrat-Anfragesprache bzw. Klasse von Atomen im Rahmenwerk zusammen (s. auch Kap. 5.3 für eine konkrete Umsetzung dieser Schritte):

1. Eine spezielle Parser-Klasse muss für die neue Anfragesprache (von der Klasse `Parser`) abgeleitet werden. Der Parser muss insbesondere die konkrete Syntax der neuen Atome kennen, und das Klassifikationsproblem mittels der Methode `get-expression-type` lösen können.

2. Es müssen ASG-Klassen für die neuen Atome (i.d.R. von Unäres-Atom (Unary-Query), Binäres-Atom (Binary-Query)) abgeleitet werden. ASG-Klassen für komplexe Rümpfe müssen i.d.R. nicht abgeleitet werden.
3. Wurde syntaktischer Zucker für die neue Anfragesprache definiert, so müssen die Methoden zum syntaktischen Umschreiben der Anfragen überschrieben werden: `rewrite-complex-query-body`, `rewrite-query-atom`.
4. Um die heuristische Optimierung zu verbessern, kann die `get-score-Bewertungsmethode` überschrieben werden. Dieser Schritt ist optional.
5. Soll die `QBox` als Index genutzt werden, so muss die `entails-p-Methode` definiert sein.

Zur Nutzung der *Anfrage-Realisierung* müssen noch einige weitere Methoden überschrieben werden. Beide Schritte sind optional.

6. Für die neuen Atome müssen u.U. die ABFen für alle möglichen Rollen des Atoms implementiert werden, sofern die bereitgestellten Standard-ABFen nicht anwendbar oder unvollständig sind. Hierzu müssen die Methoden
 - `get-tester-function`,
 - `get-generator-function`,
 - `get-successor-generator-function`, und
 - `get-predecessor-generator-function`

für die neuen Atome definiert werden, was auch einheitlich und zusammengehörig wie dargestellt durch das `defquery-code`-Makro geschehen kann. Das `defquery-code`-Makro sorgt insbesondere auch dafür, dass die zurückgegebenen ABF-Funktionsobjekte die erforderlichen Konventionen einhalten (natürlich müssen die zurückgelieferten ABF-Funktionsobjekte eine gewisse Funktionssignatur haben, etc.).

Sind die Standard-ABFen jedoch anwendbar bzw. ausreichend, so muss i.d.R. lediglich die `matches-p-` und/oder die `implies-p-Methode` entsprechend überschrieben werden, wie bereits anhand der Standard-Atome diskutiert (s. Kap. 5.2.2.8, S. 309).

5.3. Vier konkrete Substrat-Anfragesprachen

Es werden im Folgenden vier spezialisierte Substrat-Anfragesprachen sowie deren Implementationen im Rahmenwerk vorgestellt.

1. Wir beschreiben zunächst eine Anfragesprache für das Daten-Substrat (Data-Substrate), die *Daten-Substrat-Anfragesprache* (DSA). Die DSA wird später Bestandteil von NRQL sein, ist aber auch eigenständig und unabhängig von diesem Kontext nutzbar.
2. Wir definieren dann eine allgemeine ABox-Anfragesprache für beliebige ABox-DL-Systeme. Wir nennen diese Sprache die *Basis-ABox-Anfragesprache* (BAA), vgl. auch [Wessel 2003b]. Die BAA wird Konzept- und Rollen-anfrage-Atome bereitstellen und später als ABox-Anfragesprache für MIDELORA (s. Kap. 6) Verwendung finden sowie Basis für NRQL sein.
3. Die Sprache NRQL [Haarslev u. a. 2004b;a, Wessel u. Möller 2005], [Racer Systems 2006] ist eine spezifische Anpassung und Erweiterung der BAA für RACERPRO. NRQL stellt die Anfragesprache des SWIR-Dienstes auf RACERPRO-Basis bereit. Wir haben die sich aus diesem Szenario bzw. der *Leitdomäne 2* ergebenden Anforderungen und Entwurfsentscheidungen bereits erschöpfend diskutiert (s. Kap. 3.2.4, Kap. 4.1.4, Kap. 5.1.3). Wir kommen dennoch beim Fein- bzw. Detail-Entwurf von NRQL auf diese Anforderungen zurück.

Bereits beim Entwurf der Basis-Anfragebeantwortungsmaschine in Kap. 5.2 sind viele sich aus dem SWIR-Dienst ergebende Anforderungen eingeflossen (Inkrementalität, Mehrbenutzerfähigkeit, ...). Wir zeigen durch Integration bzw. Kombination mit der DSA, wie auch die verbleibenden in Kap. 3.2.4 formulierten Anforderungen an einen SWIR-Dienst in NRQL realisiert werden können. Insbesondere sollen ja auch die LUBM-Anfragen (s. Kap. 3.2.4.1) beantwortet werden können.

Wie in Kap. 3.2.4.1 erwähnt, ist der LUBM nicht nur für Benchmarking-Zwecke interessant. Er stellt auch ein interessantes SWIR-Szenario bereit, nämlich das eines *Universitäts-SSs*. Wir stellen daher den bereits erwähnten OWL TREE vor, der die grafisch-interaktive Erstellung von NRQL-Anfragen unter Verwendung des LUBM-Vokabulars und Absetzung dieser Anfragen an den LUBM-Universitäts-Datenbestand sowie die grafisch-interaktive Ergebnisinspektion ermöglicht.

4. Für die DISK bzw. *Leitdomäne 1* hatten wir in Kap. 5.1.3.1 die Karten-Substrate (von Karten-Substrat (Map-Substrate) abgeleitete Klassen) zur flexiblen Repräsentation der DISK entworfen und implementiert; wie auch für Leitdomäne 2 finalisieren wir die in den Kap. 3.1.6, Kap. 4.1.4, Kap. 5.1.3.1 erschöpfend durchgeführten Analyse- und Entwurfstätigkeiten

durch einen Fein-Entwurf.

Die als Resultat entstehende hybride Anfragesprache DISKQL ermöglicht die ontologiebasierte Anfragebeantwortung auf der DISK und realisiert die Anforderungen. Die DISKQL nutzt ebenfalls die BAA sowie die DSA. Auch hatten wir in Kap. 3.1.3.3 eine Reihe von Referenzanfragen definiert, die in der DISKQL formuliert werden sollen.

Die DISKQL wird als Anfragesprache des DLMAPS-Systems bereitgestellt, welches hier ebenfalls kurz vorgestellt wird.

5.3.1. Eine Anfragesprache für das Daten-Substrat – Die DSA

Das Daten-Substrat (Data-Substrate) (s. Kap. 5.1.2.6.2, S. 254) verwendet $\mathcal{L}_{SKNF}$ für $\mathcal{L}_\varepsilon$ und $\mathcal{L}_\nu$. Wir hatten erwähnt, dass die *einfachen Atome* (Instanzen der Klasse Einfaches-Beschreibungs-Atom (Simple-Description-Query), s. Kap. 5.2.2.8, S. 309) zur Befragung einer Instanz der Klasse Daten-Substrat (Data-Substrate) verwendet werden können, da diese für $\mathcal{L}_{QV}$ und $\mathcal{L}_{QE}$ ebenfalls $\mathcal{L}_{SKNF}$ verwenden. Die Sprache $\mathcal{L}_{SKNF}$ ist daher sowohl Annotations- als auch (Knoten- bzw. Kanten-) Anfragesprache.

Da die Knoten- und Kanten-Annotationen einer Instanz der Klasse Daten-Substrat (Data-Substrate) in $\mathcal{L}_{SKNF}$ nun jedoch auch nichtsymbolische Propositionen enthalten können, nämlich literale COMMON LISP-Datenwerte (s. Def. 92, S. 252), sind für $\mathcal{L}_{QV}$ und $\mathcal{L}_{QE}$ nun auch *prädikatsbasierte Anfragebedingungen* wünschenswert, die die „Reichhaltigkeit“ dieser Annotationen nutzen können (um z.B. alle Knoten ermitteln zu können, die mit einer Zahl zwischen 18 und 80 annotiert sind). Offensichtlich können die λ -Atome für diesen Zweck genutzt werden; auf S. 309 hatten wir jedoch argumentiert, dass dem Endnutzer einer Anfragesprache die Nutzung beliebiger λ -Atome nicht gestattet sein sollte.

Wir erweitern daher $\mathcal{L}_{SKNF}$ bzw. die *literal*-Regel in der Grammatik aus Def. 252 für $\mathcal{L}_{QV}$ und $\mathcal{L}_{QE}$ der DSA so, dass neben literalen Präpositionen auch *vordefinierte* Prädikate verwendet werden können; wir nennen diese im Folgenden *Prädikats-Literale*. Ein solches vordefiniertes Prädikats-Literal wird in konkreter Syntax wie ein λ -Atom notiert: ($\text{:predicate } \langle P \rangle$). Die so gewonnene Knoten-Anfragesprachen nennen wir $\mathcal{L}_{SKNF_{PV}}$, und die entspr. Kanten-Anfragesprache $\mathcal{L}_{SKNF_{PE}}$. Für die DSA gilt dann $\mathcal{L}_{QV} = \mathcal{L}_{SKNF_{PV}}$ und $\mathcal{L}_{QE} = \mathcal{L}_{SKNF_{PE}}$; wir geben unten eine EBNF für diese Sprachen an.

Die Semantik der Prädikats-Literale ist einfach zu verstehen: Sei $P[x]$ ein Prädikats-Literal und $w \in \mathcal{L}_{SKNF}$. Sei w' die DNF von w : $w' = w_1 \vee \dots \vee w_n = \text{DNF}(w)$. Dann gelte $w \models P[x]$ gdw. es für alle i von 1 bis n mit $w_i = w_{i,1} \wedge \dots \wedge w_{i,m}$ stets ein j zwischen 1 und m gibt, sodass $P[x]_{x \leftarrow w_{i,j}}$ zu wahr evaluiert. Man beachte, dass dies eine Generalisierung der Folgerungsbeziehung für einfache Li-

5.3 Vier konkrete Substrat-Anfragesprachen

terale ist, denn hier gilt $w \models v$ gdw. es für alle i von 1 bis n mit $w_i = w_{i,1} \wedge \dots \wedge w_{i,m}$ stets ein j zwischen 1 und m gibt, sodass $w_{i,j} = v$ ist (bzw. P ist das Gleichheitsprädikat). Die Verallgemeinerung auf Konjunktionen und Disjunktionen ist offensichtlich (wenn $P[x]$ bzw. v in komplexen KNF-Formeln vorkommt).

Während die Prädikats-Literale einstellig sind, möchte man nun ähnlich wie durch die *virtuellen λ -Kanten-Atome* (Instanzen der Klasse Lambda-Virtuelles-Kanten-Atom (Lambda-Virtual-Edge-Query)) ermöglicht, Prädikate zwischen Paaren beliebiger Knoten $(i, j) \in V \times V$ auswerten können, für die $(i, j) \notin E$ gilt und zwischen denen somit keine explizite Kante verläuft. Es werden daher auch *zweistellige* vordefinierte Prädikats-Literale benötigt. Wir verwenden für diese die konkrete Syntax der virtuellen λ -Atome: $(:satisfies <Q>)$. Ein zweistelliges Prädikats-Literal $Q[x, y]$ mit zwei freien Variablen x, y wird auf $w = \text{DNF}(L_V(i))$ und $v = \text{DNF}(L_V(j))$ angewendet. Analog zur Semantik der einstelligen Prädikats-Literale gelte nun $w, v \models Q[x, y]$ falls es für jedes Paar von Disjunkten (w_i, v_j) mit w_i aus w und v_j aus v stets ein Paar von Konjunkten in diesen Disjunkten gibt, sodass bei Substitution von x und y gegen diese Konjunkte das Prädikat Q zu wahr evaluiert. Gilt beispielsweise $L_V(i) = \text{"Achim"}$, $L_V(j) = \text{"Michael"}$, und ist $Q = \text{string-lessp}$, so wird $\text{string-lessp}_{x \leftarrow \text{"Achim"}, y \leftarrow \text{"Michael"}}$ wahr, und daher ist (i, j) in der Extension des Atoms. Offensichtlich sind insbesondere *Vergleichsprädikate* als zweistellige Prädikats-Literale nützlich.

Die Syntax für $v, w \in \mathcal{L}_{SKNF_{PV}}$ ist nun folgende:

v	$::=$ <i>konjunktion</i>
w	$::=$ <i>konjunktion</i> $(: \text{SATISFIES}$ $\{ \text{2stelliges\_praedikats\_literal} \mid$ $(\text{disjunktion\_2stelliger\_praedikats\_literale}^+) \})$
<i>konjunktion</i>	$::=$ <i>literal</i> (konjunkt^+)
<i>konjunkt</i>	$::=$ <i>literal</i> (literal^+)
<i>praedikats_literal</i>	$::=$ $(: \text{PREDICATE}$ $\{ \text{0stelliges\_praedikats\_literal} \mid$ $\text{1stelliges\_praedikats\_literal} \})$
<i>0stelliges_praedikats_literal</i>	$::=$ $\text{STRINGP} \mid \text{ZEROP} \mid \text{INTEGERP} \mid$ $\text{NUMBERP} \mid \text{CONSP} \mid \text{SYMBOLP} \mid$ $\text{KEYWORDP} \mid \text{RATIONALP} \mid \text{FLOATP} \mid$ MINUSP

```

literal ::= praedikats_literal |
           symbol_literal |
           zeichen_literal |
           zeichenkette_literal |
           numerisches_literal

1stelliges_praedikats_literal ::= ( { STRING= | STRING/= |
                                     STRING-EQUAL | STRING-NOT-EQUAL |
                                     STRING | STRING= |
                                     STRING> | STRING>= |
                                     STRING-LESSP | STRING-GREATERP |
                                     STRING-NOT-LESSP | STRING-NOT-GREATERP |
                                     SEARCH }
                                   zeichenkette_literal ) |

                                   ( { = | /= | > | < | >= | <= }
                                   numerisches_literal ) |

                                   (find zeichen_literal )

disjunktion_2stelliger_praedikats_literale ::= 2stelliges_praedikats_literal |
                                                ( 2stelliges_praedikats_literal+ )

2stelliges_praedikats_literal ::= STRING= | STRING/= |
                                   STRING-EQUAL | STRING-NOT-EQUAL |
                                   STRING | STRING= |
                                   STRING> | STRING>= |
                                   STRING-LESSP | STRING-GREATERP |
                                   STRING-NOT-LESSP | STRING-NOT-GREATERP |
                                   = | /= | > | < | >= | <=

```

5.3.1.1. Implementation der DSA

Für die DSA-Atome werden natürlich neue ASG-Klassen erforderlich. Die momentan implementierten ABFen für diese unterscheiden sich nicht wesentlich von den ABFen der λ -Atome; momentan werden noch keine Indexstrukturen genutzt. Die Generatoren sind also nicht informiert. Ähnlich wie für die λ -Atome gibt es aufgrund der Ausdrucksmächtigkeit hier ein prinzipielles Problem. Für Atome ohne Prädikats-Literale ist jedoch prinzipiell eine Indizierung möglich (das Rahmenwerk bietet u.a. AVL-Bäume für diesen Zweck).

5.3.2. Eine generische ABox-Anfragesprache – Die BAA

Als Anfragesprache für beliebige ABox-DL-Systeme definieren wir hier eine einfache ABox-Anfragesprache, die *Basis-ABox-Anfragesprache* (BAA). Für die BAA identifizieren wir $\mathcal{L}_{QV}$ mit der Menge der vom DL-System unterstützten Konzepte bzw. der implementierten DL, und analog $\mathcal{L}_{QE}$ mit der Menge der Rol-
lenterme. Wir hatten bereits in der Analyse in Kap. 3.2.4 auf S. 181 bemerkt, dass wir die Expressivität bzgl. $\mathcal{L}_{QE}$ erhöhen können, wenn wir auch (*klassische*) *negierte Rollen* zulassen. Die gemeinsame Verfügbarkeit von klassischer Negation und NAF-Negation innerhalb einer Anfragesprache kann als Alleinstellungsmerkmal der BAA (und somit auch von nRQL) angesehen werden. Wir hatten in Kap. 3.2.4 motiviert, dass beide Arten von Negation für einen (auf einer ABox-Anfragesprache basierenden) SWIR-Dienst relevant sind, aber natürlich auch für eine ABox-Anfragesprache per se.

5.3.2.1. Analyse und Entwurf der BAA

Es ist offensichtlich, dass die *vereinfachte Semantik* (s. Kap. 4.2.3) für die BAA nicht mehr anwendbar ist. Dementsprechend müssen die ABFen für die Atome der BAA neu implementiert werden. Die ABFen verwenden zur Entscheidung der Folgerbarkeitsbeziehung für die Atome die Basis-ABox-Inferenzprobleme, s. Kap. 2.2.2.5. Die ABFen der Knoten-Atome in der Generator-Rolle werden daher `concept_instances`, und in der Tester-Rolle `individual_instance?` verwenden; analog hierzu verwenden die ABFen der Rollen-Atome `individuals_related?`, `role_pairs`, `individual_role_successors`, und `individual_role_predecessors`.

Leider bieten Standard-DL-Systeme keine negierten Rollen; dementsprechend können die Standardinferenzdienste für Kanten-Atome mit negierten Rollen nicht verwendet werden. Aufgrund der ADS (s. Kap 4.2.1, S. 204) für die Variablen in der BAA kann Folgerbarkeit für Kanten-Atome mit negierten Rollen jedoch wie folgt entschieden werden (hier ist $(i, j) : \neg R$ eine Rollen-Assertion mit negierter Rolle):

Lemma 4 (Klassische negierte Rollen) Sei $\mathcal{A}$ eine erfüllbare ABox. Dann gilt $\mathcal{A} \models (i, j) : \neg R$, für $i, j \in \mathcal{A}$ gdw. $\mathcal{A} \cup \{(i, j) : R\}$ unerfüllbar ist. ■

Der Beweis ist trivial: $\mathcal{A} \cup \{(i, j) : R\}$ ist unerfüllbar gdw. es kein Modell $\mathcal{I}$ von $\mathcal{A}$ mit $\mathcal{I} \models (i, j) : R$ gibt. Dann gilt aber in allen Modellen $\mathcal{I}$ von $\mathcal{A}$: $\mathcal{I} \not\models (i, j) : R$ und somit $\mathcal{I} \models (i, j) : \neg R$. Daher ist $(i, j) : \neg R$ eine logische Konsequenz. □

Negierte Rollen-Assertionen können in ABoxen z.B. durch Rollen mit Bild- und Urbild-Restriktionen impliziert werden (s. S. 181), oder durch funktionale Rollen-Assertionen in Kombination mit der UNA, wie z.B. in der ABox $\mathcal{A} = \{(i, k) : f, j : \top\}$. Ist nun f eine funktionale Rolle ($f \in \mathcal{N}_{\mathcal{F}}$), so gilt durch

die UNA in allen Modellen $j^{\mathcal{I}} \neq k^{\mathcal{I}}$. Daher ist $\mathcal{A} \cup \{(i, j) : f\}$ unerfüllbar, denn hierdurch würde gerade $j^{\mathcal{I}} = k^{\mathcal{I}}$ gefordert. Somit muss $\mathcal{A} \models (i, j) : \neg f$ gelten.

5.3.2.1.1. Definition der BAA Nach diesen Vorbemerkungen können wir die BAA nun wie folgt definieren:

Definition 94 (Basis-ABox-Anfragesprache (BAA)) Für die *Basis-ABox-Anfragesprache* (BAA) sei $\mathcal{L}_{\mathcal{QV}} = \mathcal{DL}$ für eine DL $\mathcal{DL}$, die Menge der sog. *Anfragekonzepte*, und $\mathcal{L}_{\mathcal{QE}}$ die Menge der sog. *Anfragerollen* mit $\mathcal{N}_{\mathcal{R}} \subseteq \mathcal{L}_{\mathcal{QE}}$, welche unter Negation und Inversion abgeschlossen ist: Ist $R \in \mathcal{L}_{\mathcal{QE}}$, so gilt auch

- $\neg R \in \mathcal{L}_{\mathcal{QE}}$, und
- $R^{-1} \in \mathcal{L}_{\mathcal{QE}}$.

Zudem gelte $\neg\neg R = R$ und $(R^{-1})^{-1} = R$. Die Knoten-Atome der BAA werden Konzept-Atome, und die Kanten-Atome Rollen-Atome genannt. ■

Zwei Bemerkungen: Wir fordern für die BAA nicht, dass $\mathcal{DL}$ inverse Rollen bietet; inverse und negierte Rollen können unabhängig von $\mathcal{DL}$ stets in Rollen-Atomen verwendet werden. Die konkrete Syntax dieser Atome ergibt sich durch die konkrete Syntax des verwendeten DL-Systems; eine negierte Rolle $\neg R$ notieren wir mittels (`not R`), und eine inverse Rolle R^{-1} durch (`inv R`).

5.3.2.1.2. Effiziente ABFen für Rollen-Atome Für eine performante BAA-Anfragebeantwortung ist es essentiell, dass Rollen-Nachfolger und -Vorgänger effizient berechnet werden können. Wir hatten beschrieben, dass der Optimierer stets ABPs bevorzugt, die „navigierend“ arbeiten (und somit die Berechnung von Kreuzprodukten vermeiden). Leider ist das Navigieren bzw. Enumerieren von Rollen-Nachfolgern und -Vorgängern nun mit teurer ABox-Inferenz verbunden, da zur Prüfung auf $\mathcal{A} \models (i, j) : R$ i.Allg. ein ABox-Erfüllbarkeitstest erforderlich wird, s. Kap. 2.2.2.5, S. 79. Wir beobachten jedoch, dass für einfache DLs bis zur DL $\mathcal{ALCH}\mathcal{I}_{\mathcal{R}+}$ ein rein syntaktischer und daher sehr schneller Algorithmus für positive Anfragerollen R verwendet werden kann. Das folgende Lemma zeigt, dass zur Berechnung der positiven von einer ABox implizierten Rollen-Assertionen bis zur Sprache $\mathcal{ALCH}\mathcal{I}_{\mathcal{R}+}$ keine ABox-Erfüllbarkeitstests erforderlich werden, vgl. auch [Baader u. a. 2003a, S. 67] und [Horrocks u. Tessaris 2002]. Dieses Lemma ist eine direkte Konsequenz aus [Haarslev u. a. 2001b, S. 11, Def. 9] (und somit vor [Baader u. a. 2003a, S. 67] und [Horrocks u. Tessaris 2002] entstanden):

Lemma 5 (ABox-Traversierung) Sei $\mathcal{A}$ eine $\mathcal{ALCH}\mathcal{I}_{\mathcal{R}+}$ -ABox, und R eine positive Anfragerolle.

5.3 Vier konkrete Substrat-Anfragesprachen

Dann gilt $\mathcal{A} \models (i, j) : R$ gdw. $\mathcal{A} \rightsquigarrow (i, j) : R$. Die Relation $\rightsquigarrow$ ist wie folgt definiert: Es gilt $\mathcal{A} \rightsquigarrow (i, j) : R$ gdw.

- $(i, j) : R \in \mathcal{A}$, oder
- $\mathcal{A} \rightsquigarrow (i, j) : S$, für $S \sqsubseteq R$, oder
- $\mathcal{A} \rightsquigarrow (j, i) : S$, für $S^{-1} \sqsubseteq R$, oder
- $\mathcal{A} \rightsquigarrow (i, k) : R$ und $\mathcal{A} \rightsquigarrow (k, j) : R$, für irgendein $k \in \text{inds}(\mathcal{A})$, falls R transitiv ist. ■

Beweis 2 Die Richtung $\mathcal{A} \rightsquigarrow (i, j) : R$ impliziert $\mathcal{A} \models (i, j) : R$ ist trivial. Das syntaktische Kriterium $\rightsquigarrow$ ist also korrekt bzw. hinreichend für $\models$.

Die Richtung $\mathcal{A} \models (i, j) : R$ impliziert $\mathcal{A} \rightsquigarrow (i, j) : R$ erfordert die Berücksichtigung der Eigenschaften von $\mathcal{ALCHIT}_{\mathcal{R}^+}$.

Angenommen, $\mathcal{A} \models (i, j) : R$, aber $\mathcal{A} \not\rightsquigarrow (i, j) : R$.

Für den Fall, dass R nicht transitiv ist, kann $\mathcal{A} \models (i, j) : R$ nur durch ein Rollenaxiom der Form $(i, j) : S$, $(j, i) : S^{-1}$ mit $S \sqsubseteq R$ erzwungen werden, da $\mathcal{ALCHIT}_{\mathcal{R}^+}$ weder qualifizierte Anzahlrestriktionen noch funktionale Rollen hat. Daher muss auch $\mathcal{A} \rightsquigarrow (i, j) : R$ gelten, im Widerspruch zu Annahme.¹⁷

Für den Fall, dass R transitiv ist, gilt $R^{\mathcal{I}} = (R^{\mathcal{I}})^+$, d.h., $R^{\mathcal{I}}$ ist transitiv abgeschlossen. Gilt $\mathcal{A} \models (i, j) : R$, so gibt es für $(i^{\mathcal{I}}, j^{\mathcal{I}}) \in R^{\mathcal{I}}$ einen Pfad von ABox-Individuen $k_1, \dots, k_n$ mit $k_1 = i$ und $k_n = j$ maximaler Länge n , sodass alle k_i auf diesem Pfad verschieden sind und $(k_i^{\mathcal{I}}, k_{i+1}^{\mathcal{I}}) \in R^{\mathcal{I}}$ gilt. Da der Pfad maximal ist, gilt $\mathcal{A} \rightsquigarrow (k_m, k_{m+1}) : R$ für alle $m \in 1 \dots n$. Offensichtlich gilt dann aber lt. Definition von $\rightsquigarrow$ auch $\mathcal{A} \rightsquigarrow (k_l, k_m) : R$ für alle $l < m$ und $l, m \in 1 \dots n$. Somit auch $\mathcal{A} \rightsquigarrow (k_1, k_n) : R$, also $\mathcal{A} \rightsquigarrow (i, j) : R$, im Widerspruch zur Annahme. □

Es ist offensichtlich, dass das Lemma direkt in einen Algorithmus für die AB-Fen der Rollen-Atome umgesetzt werden kann; der Algorithmus ist seit Anfang 2004 in NRQL implementiert. Der Algorithmus wird für DLs wie die RACERPRO-Logik $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ bzw. für DLs mit funktionalen Rollen und/oder Anzahlrestriktionen jedoch unvollständig. Es gilt dann aber eine schwächere Form des Lemmas, das wir hier exemplarisch für $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ (als ausdrucksstärkste für diese Arbeit relevante Standard-DL) formulieren:

Lemma 6 Seit $\mathcal{A}$ eine $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ -ABox, und R eine positive Anfragerolle.

Dann impliziert $\mathcal{A} \models (i, j) : R$ die Existenz eines Pfades von ABox-Individuen $k_1, \dots, k_n$ mit $k_1 = i$ und $k_n = j$ maximaler Länge n , sodass alle k_i auf diesem Pfad verschieden sind und $(k_m, k_{m+1}) : S_m \in \mathcal{A}$ für $m \in 1 \dots n$ für beliebige Rollen

¹⁷Man beachte, dass die Einschränkung auf $\mathcal{ALCHIT}_{\mathcal{R}^+}$ wesentlich ist: In der ABox $\{(i, j) : F_1, i : \exists F_2.C\}$ mit $F_1 \sqsubseteq G$, $F_2 \sqsubseteq G$, und $G \in \mathcal{N}_{\mathcal{F}}$, gilt offensichtlich $\mathcal{A} \models (i, j) : F_2$, aber $\mathcal{A} \not\rightsquigarrow (i, j) : F_2$.

S_m bzw. invertierte Rollen S_m^{-1} gilt (wir nehmen an, dass für $(i, j) : S \in \mathcal{A}$ stets auch $(j, i) : S^{-1} \in \mathcal{A}$ gilt). ■

Das Lemma sagt also aus, dass $\mathcal{A} \models (i, j) : R$ nicht ohne „stützendes Skelett“ von Rollen-Assertionen in einer ABox gelten kann. Dies ist letztlich darin begründet, dass $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ keine *definierten Rollen* bietet.¹⁸ Das Lemma gilt offensichtlich auch nicht für negierte Anfragerollen oder sobald Nominale oder same-as-Assertionen in der WB bzw. ABox verwendet werden können (was in $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ nicht der Fall ist). Das Lemma kann wiederum für einen Algorithmus für die ABFen der Rollen-Atome bzw. zur Prüfung auf $\mathcal{A} \models (i, j) : R$ so ausgenutzt werden, dass der Test sofort fehlschlägt, wenn kein Pfad von Rollen-Assertionen zwischen i und j existiert. Der teure ABox-Erfüllbarkeitstest wird dann durch diesen syntaktischen und günstigen Test „bewacht“ (s. auch Kap. 2.2.7.1).

5.3.2.1.3. Ein Algorithmus zur Prüfung auf Anfragesubsumption Wir stellen hier einen Algorithmus zur Entscheidung von Anfragesubsumption vor, vgl. auch [Horrocks u. a. 1999a; 2000a]. Der hier vorgestellte Algorithmus [Wessel 2003b] ist durch die zitierten Arbeiten beeinflusst bzw. inspiriert, doch weichen die technischen Details ab.

Wie dargestellt, wird ein solcher Algorithmus zur Nutzung der QBox benötigt (Methode `entails-p`). Wir stellen hier exemplarisch einen Algorithmus vor, der vollständig für Anfragen in Normalform (s. Def. 93, S. 282) ohne NAF-negierte Atome und ohne Referenz-Atome ist (die Anfrage muss also durch Normalisierung einer π -freien Anfrage entstanden sein). Wir hatten bereits beschrieben, dass Unvollständigkeit in diesem Kontext unproblematisch ist (s. S. 285), wenn lediglich die Anfragen in die QBox eingetragen werden, für die der Algorithmus vollständig ist.

Die Basis-Idee für den Algorithmus ist, Anfragesubsumption¹⁹ auf Anfrage-Erfüllbarkeit zu reduzieren, vgl. auch [Horrocks u. a. 1999a; 2000a]. Seien $q_2 = (\text{antwort}_2(\vec{x}_2) \Leftarrow k_2)$ und $q_1 = (\text{antwort}_1(\vec{x}_1) \Leftarrow k_1)$ zwei π -freie Anfragen in DNF. Offensichtlich gilt nun $k_2 \sqsubseteq k_1$ gdw. $k_2^\mathcal{E} \subseteq k_1^\mathcal{E}$ für alle S gdw. $k_2^\mathcal{E} \cap (V^n \setminus (k_1^\mathcal{E})) = \emptyset$ für beliebiges Substrat S mit Knotenmenge V , wobei n die Stelligkeit der Disjunkte in k_1 ist. Da die k_i normalisiert wurden (s. Def. 93, S. 282) ist $n = |\text{vars}(k_1)|$. Es ist klar, dass die Relation $k_2^\mathcal{E} \subseteq k_1^\mathcal{E}$ nicht von den Namen der in $\text{vars}(k_i)$ verwendeten Variablen abhängen sollte. Wir müssen Variablen daher u.U. umbenennen (s.u.). Es gilt nun $k_2^\mathcal{E} \cap (V^n \setminus (k_1^\mathcal{E})) = \emptyset$ für beliebiges Substrat S genau dann wenn es *kein* Substrat S mit $k_2^\mathcal{E} \cap (V^n \setminus (k_1^\mathcal{E})) \neq \emptyset$

¹⁸Eine definierte Rolle entspricht prädikatenlogisch einer Biimplikation, analog den definierten Konzepten; z.B. ist $R(x, y) \Leftrightarrow C(x) \wedge D(y)$ eine Rollendefinition.

¹⁹Wir verwenden die Begriffe *Query Subsumption* und *Query Entailment* synonym.

5.3 Vier konkrete Substrat-Anfragesprachen

gibt. Dies bedeutet gemäß Def. 87, S. 226 und gemäß der Semantik von $\setminus$ und $\wedge$, dass $k_2 \wedge \setminus k_1$ unerfüllbar ist. Wir hatten bereits auf S. 227 diskutiert, dass wir diese rein mathematische Bedingung für das Vorliegen einer Subsumptionsbeziehung noch verschärfen werden, indem wir zusätzlich $\text{vars}(k_1) = \text{vars}(k_2)$ fordern; so stellen wir sicher, dass die Stelligkeiten der k_i^ε identisch sind, und die Cache-Einträge unmittelbar und ohne weitere Bearbeitung genutzt werden können.²⁰

Es stellt sich die Frage, wie die Gleichheits-Atome zu behandeln sind, die insbesondere auch durch Anwendung von `ersetze_individuen` in k_i entstanden sein können; wir kommen hierauf bei der Darstellung des Erfüllbarkeitstesters in Kap. 5.3.2.1.4 zurück.

Wurde k_1 bereits ausgeführt, so ist in der QBox k_1^ε als Wert des Attributes `result` (s. Klasse Anfrage (Query)) gespeichert. Ein Tupel aus k_1^ε repräsentiert eine Bindungsmöglichkeit für die Variablen $\text{vars}(k_1)$. Wir nehmen an, dass auf Vars eine lexikografische Ordnung definiert ist. Für ein Variablen-Tupel (bzw. Vektor) $\text{vars}(k_1) = (x_{1,1}, \dots, x_{1,n})$ repräsentiert ein Tupel $(i_1, \dots, i_n) \in k_1^\varepsilon$ dann die Bindungsliste $x_{1,1} \leftarrow i_1, \dots, x_{1,n} \leftarrow i_n$.

Betrachten wir beispielsweise die beiden Rumpfe $k_2 = (b, C \sqcap D) \wedge (b, a, R)$ und $k_1 = (x, C) \wedge (x, y, R)$, mit $\text{vars}(k_2) = (a, b)$ und $\text{vars}(k_1) = (x, y)$. Offensichtlich gilt nur dann $k_2^\varepsilon \subseteq k_1^\varepsilon$ und daher $k_2 \sqsubseteq k_1$, wenn die Ergebnistupel k_1^ε invertiert werden, also die Ordnung von (x, y) in (y, x) geändert wird (bzw. (a, b) in (b, a)). Wir können nun entweder den Standpunkt vertreten, dass die Ordnung relevant ist (in diesem Fall gilt $k_2 \sqsubseteq k_1$ nicht), oder aber sagen, dass eine Permutation $\Pi : \text{vars}(k_2) \rightarrow \text{vars}(k_1)$ existieren muss, sodass $\Pi(k_2) \sqsubseteq k_1$ gilt. Das Raten einer Permutation (und anschließendes Testen auf Subsumption) ist natürlich sehr teuer – in nRQL wird auf das Raten einer Permutation daher verzichtet. Offensichtlich kann eine solche Permutation nur dann existieren, wenn $|\text{vars}(k_2)| = |\text{vars}(k_1)|$ gilt. Prinzipiell wäre es im Fall $|\text{vars}(k_1)| > |\text{vars}(k_2)|$ möglich, zunächst eine $|\text{vars}(k_2)|$ -elementige Teilmenge aus $\text{vars}(k_1)$ zu raten, und dann die Permutation zu raten. Wir verzichten hierauf aus Gründen der Einfachheit (s. auch S. 227).

Nun wird $\Pi(k_2) \wedge \setminus k_1$ auf Unerfüllbarkeit getestet. Die k_i sind nun in DNF, π -frei,²¹ und enthalten weder NAF- noch $\sqsubseteq$ -Atome. Es muss dann Erfüllbarkeit von Anfragen der Form

$$(k_{2,1} \vee \dots \vee k_{2,n}) \wedge \setminus (k_{1,1} \vee \dots \vee k_{1,m}),$$

²⁰Natürlich dürfen die Ergebnismengen von Anfragen in der QBox nur dann als Obermengen- bzw. Kandidaten-Caches genutzt werden, wenn diese *vollständig* sind. Unvollständige Ergebnismengen treten insbesondere bei Anfragen auf, die abgebrochen wurden, oder für die eine statische oder dynamische Begrenzung der Ergebnismenge aktiv war, z.B. wenn im inkrementellen Modus nicht alle Tupel angefordert wurden.

²¹Man beachte, dass π der Projektionsoperator, aber Π die Bijektion ist.

entschieden werden, wobei die $k_{i,j}$ konjunktive Anfragen sind. Anwendung von DeMorgan ergibt

$$(k_{2,1} \vee \dots \vee k_{2,n}) \wedge \neg k_{1,1} \wedge \dots \wedge \neg k_{1,m},$$

und durch weiteres Ausmultiplizieren:

$$(k_{2,1} \wedge \neg k_{1,1} \wedge \dots \wedge \neg k_{1,m}) \vee \dots \vee (k_{2,n} \wedge \neg k_{1,1} \wedge \dots \wedge \neg k_{1,m}).$$

Da $k_{1,i} = k_{1,i,1} \wedge \dots \wedge k_{1,i,k}$, ist ergibt sich

$$\begin{aligned} & (k_{2,1} \wedge (\neg k_{1,1,1} \vee \dots \vee \neg k_{1,1,k}) \wedge \dots \wedge (\neg k_{1,m,1} \vee \dots \vee \neg k_{1,m,k})) \vee \\ & \quad \vdots \\ & (k_{2,n} \wedge (\neg k_{1,1,1} \vee \dots \vee \neg k_{1,1,k}) \wedge \dots \wedge (\neg k_{1,m,1} \vee \dots \vee \neg k_{1,m,k})). \end{aligned}$$

Speziell für den Fall, dass k_1 eine einfache Konjunktion ist, erhalten wir (da $m = 1$)

$$\begin{aligned} & (k_{2,1} \wedge (\neg k_{1,1,1} \vee \dots \vee \neg k_{1,1,k})) \vee \\ & \quad \vdots \\ & (k_{2,n} \wedge (\neg k_{1,1,1} \vee \dots \vee \neg k_{1,1,k})), \end{aligned}$$

und somit durch weiteres Ausmultiplizieren

$$\begin{aligned} & (k_{2,1} \wedge \neg k_{1,1,1}) \vee \dots \vee (k_{2,1} \wedge \neg k_{1,1,k}) \vee \\ & \quad \vdots \\ & (k_{2,n} \wedge \neg k_{1,1,1}) \vee \dots \vee (k_{2,n} \wedge \neg k_{1,1,k}) \end{aligned}$$

und durch Einsetzen von $k_{2,i} = k_{2,i,1} \wedge \dots \wedge k_{2,i,l}$ letztlich

$$\begin{aligned} & (k_{2,1,1} \wedge \dots \wedge k_{2,1,l} \wedge \neg k_{1,1,1}) \vee \dots \vee (k_{2,1,1} \wedge \dots \wedge k_{2,1,l} \wedge \neg k_{1,1,k}) \vee \\ & \quad \vdots \\ & (k_{2,n,1} \wedge \dots \wedge k_{2,n,l} \wedge \neg k_{1,1,1}) \vee \dots \vee (k_{2,n,1} \wedge \dots \wedge k_{2,n,l} \wedge \neg k_{1,1,k}). \end{aligned}$$

Da eine Disjunktion unerfüllbar ist gdw. alle Disjunkte unerfüllbar sind, ist ein Erfüllbarkeitstester für Rümpfe ohne Disjunktionen ausreichend, sofern wir für k_1 lediglich Rümpfe ohne Disjunktion zulassen. Wir halten daher als wichtiges Zwischenergebnis fest:

Sind k_1 und k_2 Rümpfe in Normalform ohne NAF- und Referenz-Atome, und ist k_1 eine einfache Konjunktion mit l Atomen, und hat k_2 n Disjunkte, so müssen $n * l$ Rümpfe auf Erfüllbarkeit getestet werden. Diese Rümpfe enthalten dann jeweils genau ein NAF-Atom.

Der Algorithmus zur Prüfung von $k_2 \sqsubseteq k_1$ funktioniert nun wie folgt:

1. Eingabe: k_1 und k_2 , beide in Normalform, ohne NAF- und ohne Referenz-Atome.

5.3 Vier konkrete Substrat-Anfragesprachen

2. Ausgabe: Gilt $k_2 \sqsubseteq k_1$: TRUE, FALSE, UNKNOWN.
3. Wenn $|\text{vars}(k_1)| \neq |\text{vars}(k_2)|$, gib FALSE zurück.
4. Wenn $k_2 \wedge \neg k_1$ nicht auf Erfüllbarkeit getestet werden kann aufgrund der Beschränkungen des SAT-Testers (s.u.), dann gib UNKNOWN zurück.
5. Solange es noch eine nichtprobierte Permutation Π von $\text{vars}(k_2)$ nach $\text{vars}(k_1)$ gibt:
 - Berechne $\text{DNF}(\Pi(k_2) \wedge \neg k_1) = d_1 \vee \dots \vee d_l$.
 - Gilt $\text{SAT}(d_i) = \text{FALSE}$ für alle $i \in 1 \dots l$, dann gib TRUE zurück.
6. Gib FALSE zurück.

5.3.2.1.4. Ein Algorithmus zur Prüfung auf Erfüllbarkeit von Anfragen Wir geben nun einen Algorithmus an, der im oben angegebenen Algorithmus zur Prüfung auf Anfragesubsumption als SAT-Tester verwendet werden kann. Zu Illustration der Basis-Idee betrachten wir den Rumpf $(x, C) \wedge \neg(x, D)$, der durch den obigen Algorithmus bei Prüfung von $(x, C) \sqsubseteq (x, D)$ erzeugt wird. Aufgrund einer anderen Semantik sind die Algorithmen aus [Horrocks u. a. 1999a; 2000a] nicht direkt anwendbar.

Offensichtlich ist der Rumpf $(x, C) \wedge \neg(x, D)$ unerfüllbar gdw. für beliebiges Substrat S mit Knotenmenge V gilt: $(x, C)^\varepsilon \cap (\neg(x, D))^\varepsilon = \emptyset$. Laut Definition muss dann

$$\{i \mid i \in V_S, \forall \mathcal{I}, \mathcal{I} \models S.\mathcal{I} \models C(i)\} \cap V_S \setminus \{i \mid i \in V_S, \forall \mathcal{I}, \mathcal{I} \models S.\mathcal{I} \models D(i)\} = \emptyset$$

bzw.

$$\{i \mid i \in V_S, \forall \mathcal{I}, \mathcal{I} \models S.\mathcal{I} \models C(i)\} \cap \{i \mid i \in V_S, \neg(\forall \mathcal{I}, \mathcal{I} \models S.\mathcal{I} \models D(i))\} = \emptyset$$

bzw.

$$\{i \mid i \in V_S, \forall \mathcal{I}, \mathcal{I} \models S.\mathcal{I} \models C(i)\} \cap \{i \mid i \in V_S, \exists \mathcal{I}, \mathcal{I} \models S.\mathcal{I} \not\models D(i)\} = \emptyset$$

bzw.

$$\{i \mid i \in V_S, \forall \mathcal{I}, \mathcal{I} \models S.\mathcal{I} \models C(i)\} \cap \{i \mid i \in V_S, \exists \mathcal{I}, \mathcal{I} \models S.\mathcal{I} \models \neg D(i)\} = \emptyset$$

gelten. Der Rumpf ist also unerfüllbar, wenn die beiden Mengen disjunkt sind, d.h. für ein beliebiges $i \in V_S$ gilt entweder

1. $i \in \{i \in V_S, \forall \mathcal{I}, \mathcal{I} \models S.\mathcal{I} \models C(i)\}$ und $i \notin \{i \in V_S, \exists \mathcal{I}, \mathcal{I} \models S.\mathcal{I} \models \neg D(i)\}$, oder
2. $i \notin \{i \in V_S, \forall \mathcal{I}, \mathcal{I} \models S.\mathcal{I} \models C(i)\}$ und $i \in \{i \in V_S, \exists \mathcal{I}, \mathcal{I} \models S.\mathcal{I} \models \neg D(i)\}$.

Dabei ist zu beachten, dass wir hier über die Menge aller möglichen Modelle von S quantifizieren (s. Def. 4, S. 32). Dann ist gemäß 2. $C(i)$ entweder keine logische Folgerung aus S , oder aber gemäß 1. folgt $C(i)$ logisch aus S , aber $i \notin \{i \mid i \in V_S, \exists \mathcal{I}, \mathcal{I} \models S, \mathcal{I} \models \neg D(i)\}$, d.h. es gibt kein Modell, in dem $i^{\mathcal{I}} \in (\neg D)^{\mathcal{I}}$ gilt. Also muss in allen Modellen $i^{\mathcal{I}} \in D^{\mathcal{I}}$ gelten; dies kann für *beliebiges* S jedoch nur dann gelten, wenn bereits $C(i) \models D(i)$ gilt.

Um diese beiden Bedingungen zu testen, können wir eine ABox $\{x : C \sqcap \neg D\}$ konstruieren und auf Unerfüllbarkeit testen. Ist dies der Fall, so ist der Rumpf unerfüllbar, und $(x, C) \sqsubseteq (x, D)$ gilt (denn i ist ja beliebig).

Leider funktioniert diese Art der Argumentation nur für Rümpfe, die *höchstens ein NAF-negiertes Atom haben*. Wir hatten oben jedoch festgestellt, dass solche Rümpfe gerade vom Subsumptionstester generiert werden, wenn die k_i ausschließlich positive Atome verwenden, und k_1 eine Konjunktion ist. Für Rümpfe, die höchstens ein NAF-negiertes Atom haben, kann die NAF-Negation jedoch einfach “nach innen” gezogen und daher in eine klassische DL-Negation verwandelt werden, wie anhand des Beispiels gezeigt.

Der Rumpf $\setminus(x, C) \wedge \setminus(x, \neg C)$ hat hingegen zwei NAF-negierte Atome. Der Rumpf ist erfüllbar, wie das Substrat $S = \{i : \top\}$ zeigt. Hier gilt $S \not\models C(i)$ und $S \not\models \neg C(i)$, und daher $S \models \setminus(i, C)$ und $S \models \setminus(i, \neg C)$. Also gilt $(\setminus(x, C) \wedge \setminus(x, \neg C))^{\mathcal{E}} = \{i\}$. Andererseits würde durch „Umwandlung“ von $\setminus$ in $\neg$ die ABox $\{x : \neg C \sqcap C\}$ entstehen, die offensichtlich unerfüllbar ist, obwohl der Rumpf erfüllbar ist (das Verfahren würde also ein falsches Ergebnis liefern).

Die Vorbemerkungen geben Anlass zu folgendem Algorithmus zur Entscheidung von SAT. Ähnlich wie ein Tableau-Beweiser durch Konstruktion eines *kanonischen Modells* (in Form einer Vervollständigung) einer ABox ihre Erfüllbarkeit nachzuweisen sucht, so konstruiert der folgende SAT-Tester ein kanonisches Substrat bzw. eine ABox $\mathcal{A}_k$, für die $\mathcal{A}_k \models k$ gilt gdw. $k^{\mathcal{E}} \neq \emptyset$, womit die Erfüllbarkeit von k demonstriert wird.

Für einen Rumpf der Bauart $k = p_1 \wedge \dots \wedge p_k \wedge \setminus n_1 \wedge \dots \wedge \setminus n_l$ seien die Operatoren `positive_konjunkte` und `negative_konjunkte` wie folgt definiert: $\text{positive\_konjunkte}(k) =_{\text{def}} \{p_1, \dots, p_k\}$, $\text{negative\_konjunkte}(k) =_{\text{def}} \{n_1, \dots, n_l\}$.

Der SAT-Algorithmus arbeitet nun wie folgt:

1. Eingabe: Ein Rumpf k in Normalform ohne Referenz-Atome.
2. Ausgabe: TRUE, falls k beweisbar erfüllbar ist, FALSE falls k beweisbar unerfüllbar ist, und sonst UNKNOWN.
3. Gilt $|\text{negative\_konjunkte}(k)| > 1$, dann gib UNKNOWN zurück.
4. Sonst konstruiere folgende ABox $\mathcal{A}_k$; hier sind $eq, ref \in \mathcal{N}_{\mathcal{F}}$, also funktionale Rollen:

5.3 Vier konkrete Substrat-Anfragesprachen

- a) $\mathcal{A}_0 := \{ (\boxed{a}, a) : ref \mid a \in inds(k) \}$;
 $\boxed{a}$ ist ein frisches Symbol, das anhand von a gebildet wird (es wird sowohl als Individuum als auch als atomares Konzept verwendet),
- b) $\mathcal{A}_1 := \{ \boxed{a} : \boxed{a}, \boxed{a} : \neg \boxed{b} \mid a, b \in inds(k), a \neq b \}$,
- c) $\mathcal{A}_2 := \{ a : C \mid (a, C) \in positive\_konjunkte(k) \}$,
- d) $\mathcal{A}_3 := \{ a : \neg C \mid (a, C) \in negative\_konjunkte(k) \}$,
- e) $\mathcal{A}_4 := \{ (a, b) : R \mid (a, b, R) \in positive\_konjunkte(k) \}$,
- f) $\mathcal{A}_5 := \{ a : \forall R. \neg \boxed{Rab}, b : \boxed{Rab} \mid (a, b, R) \in negative\_konjunkte(k) \}$;
 $\boxed{Rab}$ ist ein frisches atomares Konzept, das anhand von a, b, R gebildet wird,
- g) $\mathcal{A}_6 := \{ (\boxed{ab}, \boxed{a}) : eq, (\boxed{ab}, \boxed{b}) : eq \mid (a, b, \boxed{=}) \in positive\_konjunkte(k) \}$,
- h) $\mathcal{A}_7 := \{ \boxed{a} : \neg \boxed{b} \mid (a, b, \boxed{=}) \in negative\_konjunkte(k) \}$,
- i) $\mathcal{A}_k =_{def} \bigcup_{i \in 0 \dots 7} \mathcal{A}_i$.

5. Gib das Ergebnis des (ohne UNA) durchgeführten ABox-Erfüllbarkeits-tests von $\mathcal{A}_k$ zurück.

Die funktionale Rolle bzw. das Funktionssymbol ref hatten wir bereits zur prädikatenlogischen Übersetzung des $\boxed{=}$ -Prädikats verwendet; die Konstruktion wurde in Kap. 4.2.1 auf S. 204 erläutert. Die zweite funktionale Rolle eq wird verwendet, um die durch die Gleichheits-Atome erzwungenen Co- bzw. Kontrareferenzen zwischen Objekten zu repräsentieren. Zur Illustration des Algorithmus' betrachten wir nun ein komplexeres Beispiel:

5.3.2.1.5. Ein Beispiel zur Anfragensubsumption Sei $mother \sqsubseteq woman \sqcap \exists has\_child. \top$, $mother \sqsubseteq woman$, $woman \sqsubseteq person$, $man \sqsubseteq person$, $has\_child = has\_parent^{-1}$ in der TBox gegeben.

Betrachten wir die Anfragen

$$k_1 = (x, woman) \wedge (x, y, has\_child) \wedge (y, person)$$

und

$$k_2 = (a, man) \wedge (a, b, has\_parent) \wedge (b, mother).$$

Hier gilt $k_2 \sqsubseteq k_1$, denn für die Permutation $\Pi(a) = y, \Pi(b) = x$ finden wir, dass

- $(y, man) \wedge (y, x, has\_parent) \wedge (x, mother) \wedge \neg (x, woman)$ unerfüllbar ist, weil bereits $\{x : mother, x : \neg woman\} \subseteq \mathcal{A}_k$ unerfüllbar ist (wir geben hier nur die relevanten ABox-Assertionen an), und
- $(y, man) \wedge (y, x, has\_parent) \wedge (x, mother) \wedge \neg (x, y, has\_child)$ unerfüllbar ist, weil $\{(y, x) : has\_parent, x : \forall has\_child. \neg \boxed{has\_child\_x.y}, y : \boxed{has\_child\_x.y}\} \subseteq \mathcal{A}_k$

unerfüllbar ist, da $\{(y, x) : \text{has\_parent}\} \models \{(x, y) : \text{has\_child}\}$, und letztlich auch

- $(y, \text{man}) \wedge (y, x, \text{has\_parent}) \wedge (x, \text{mother}) \wedge \neg(y, \text{person})$ unerfüllbar ist, weil $\{y : \text{man}, y : \neg \text{person}\} \subseteq \mathcal{A}_k$ unerfüllbar ist.

5.3.2.2. Implementation der BAA im Rahmenwerk

Die zur Realisierung einer neuen Anfragesprache wie der BAA im Rahmenwerk vorzunehmenden Implementierungstätigkeiten hatten wir in Kap. 5.2.3.2, S. 316 diskutiert: Wir leiten einen ABox-Anfrage-Parser (ABox-Query-Parser) vom Parser ab und implementieren die benötigten Methoden, u.a. `get-expression-type`. Der Parser ist prinzipiell DL-System-spezifisch, doch verwenden die meisten DL-Systeme KRSS-Syntax. Wir hatten erwähnt, dass die BAA für beliebige ABox-DL-Systeme lauffähig sein soll. Wir müssen daher stets gegen eine generische Schnittstelle, und nicht gegen die spezifische Dienstschnittstelle (oder Syntax) eines speziellen DL-Systems implementieren.

Offensichtlich müssen auch ASG-Klassen zur Repräsentation von Konzept- und Rollen-Atomen abgeleitet werden; wir sehen entspr. Klassen vor (Konzept-Atom (Concept-Query), Rollen-Atom (Role-Query)). Wir implementieren die ABFen mittels des `defquery-code`-Makros; wiederum sollen diese ABF-Implementationen für beliebige ABox-DL-Systeme lauffähig sein, sofern diese die von den ABF-Implementationen benötigten Basis-ABox-Inferenzdienste anbieten. Die lose Kopplung an das DL-System erreichen wir wieder durch eine generische Dienstschnittstelle, die durch die abstrakte Unterklasse ABox-BAA-Substrat (ABox-BAA-Substrate) der (ebenfalls abstrakten Klasse) ABox-Substrat (ABox-Substrate) bereitgestellt wird, s. Abb. 5.19. So wird beispielsweise die Generator-ABF für Konzept-Atome die Schnittstellenmethode `dl-prover-get-concept-instances` statt einer DL-System-spezifischen `concept-instances`-Methode rufen.

Wie in Kap. 2.2.2.5, S. 78 diskutiert, können prinzipiell alle ABox-Basis-Retrieval-Probleme auf ABox-Erfüllbarkeitstests reduziert werden. Prinzipiell wäre daher bereits die Funktion `abox-consistent-p` als „minimales und generisches API“ zur Implementation der ABFen der BAA ausreichend. Doch ist dann mit einem hohen Performanzverlust zu rechnen, da DL-Systeme wie RACERPRO (und auch MIDELORA, s. Kap. 6) hocheffiziente und optimierte Implementationen dieser Basis-ABox-Inferenzdienste bereitstellen, deren Performanz nicht ohne Weiteres durch Reduktion auf einzelne ABox-Erfüllbarkeitstests reimplementiert werden kann.

Je nachdem, ob ein lokales (z.B. MIDELORA) oder ein entferntes (z.B. der RACERPRO-Server) ABox-DL-System für die BAA verwendet wird, werden die benötigten Inferenzdienste nun entweder durch lokalen (gewöhnlichen) oder

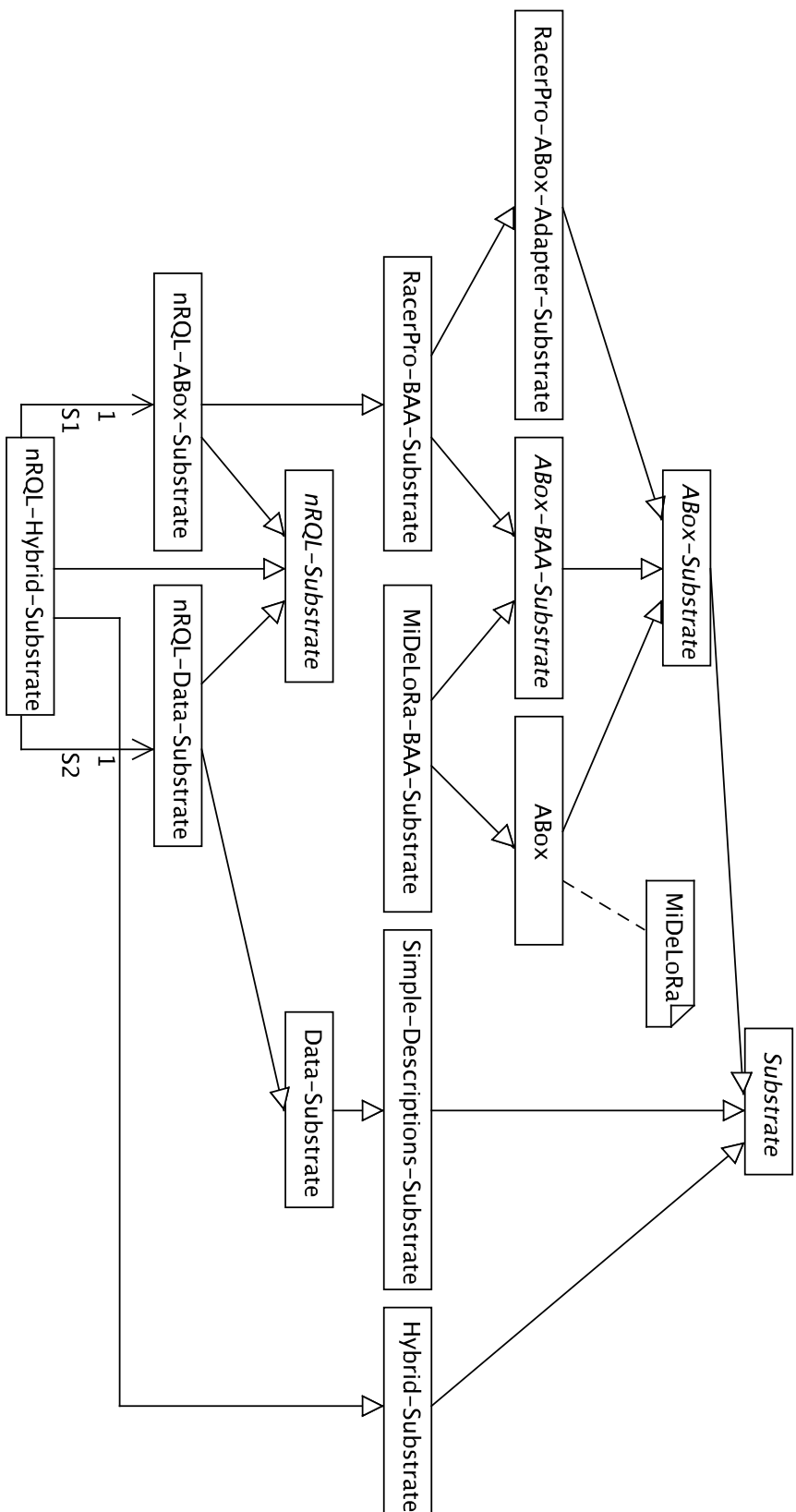


Abbildung 5.19.: Klassen für die BAA und NRQL.

entfernten (RPC-artigen) Prozedur- bzw. Methoden-Aufruf in Anspruch genommen. Tatsächlich nutzte die speziell für RACERPRO angepasste BAA-Erweiterung namens NRQL zunächst den RACERPRO-Server, wurde dann aber im weiteren Verlauf dieser Arbeiten auf Quellcode-Ebene in RACERPRO integriert. Die entfernten RACERPRO-API-Aufrufe wurden dann zu lokalen Aufrufen; am Quellcode von NRQL musste nichts geändert werden.

Insbesondere für die Nutzung eines entfernten DL-Systems müssen wir Caching-Funktionalität zur Minimierung der Netzwerk- bzw. Kommunikations-Latenz vorsehen; wir kommen hierauf bei der Darstellung von NRQL in Kap. 5.3.3 zurück. Für die lokale Nutzung wären solche Caches jedoch wahrscheinlich redundant. Durch Bildung gemeinsamer Unterklassen der Klasse ABox-BAA-Substrat (ABox-BAA-Substrate) und existierender konkreter Unterklassen der Klasse ABox-Substrat (ABox-Substrate) werden konkrete Klassen für die einzelnen DL-Systeme geschaffen, in deren Implementationen derartige Aspekte adäquat berücksichtigt werden können.

Beispielsweise wird eine Instanz der Klasse MiDeLoRA-BAA-Substrat (MiDeLoRA-BAA-Substrate) genutzt, um die BAA für MiDeLoRA bereitzustellen. Aus MiDeLoRA-Perspektive handelt es sich hierbei um eine spezielle ABox; ein und dieselbe Instanz kann also, ohne dass Redundanz durch erforderliche Caches und/oder Datenkopien geschaffen würde, für verschiedene Zwecke genutzt werden.

Im Gegensatz hierzu ist das RACERPRO-BAA-Substrat (RACERPRO-BAA-Substrate) zur BAA-Bereitstellung für RACERPRO ein „virtuelles Substrat“ (verwaltet keinen Objektgraphen, s. S. 244, S. 263), da es von der Klasse RACERPRO-ABox-Adapter-Substrat (RACERPRO-ABox-Adapter-Substrate) abgeleitet ist. Wir erben bewusst nicht von der Klasse ABox-Adapter-Spiegel-Substrat (ABox-Adapter-Mirror-Substrate), welches eine Spiegel- bzw. Proxy-Repräsentation der ABox bereitstellen würde, da die BAA die Basis für NRQL bereitstellen soll, und eine solche Doppelrepräsentation der ABox aus RACERPRO-Perspektive dann redundant und evtl. auch zu schwergewichtig für ein direkt auf Quellcode-Ebene in RACERPRO integriertes NRQL wäre.

Das API von RACERPRO wurde nicht ausgelegt, um eine *effiziente* BAA-Implementierung zu erlauben, da es unter anderen Gesichtspunkten entworfen wurde. Insbesondere fehlten zu Beginn der Arbeiten an der BAA (ca. 2003) Indexstrukturen für RACERPRO-ABoxen. So war es damals mit dem RACERPRO-API nicht möglich, auf effiziente Weise auf die Rollen-Nachfolger und/oder -Vorgänger eines ABox-Individuums zuzugreifen. Dies ist für eine effiziente Implementierung des durch Lemma 5 implizierten Verfahrens jedoch essentiell. Da RACERPRO (aus der externen Server-Perspektive) nicht auf Quellcode-Ebene um diese benötigte Zusatz-Funktionalität erweiterbar war, musste zur Realisierung einer effizienten NRQL-Anfragebearbeitung damals eine weitere Un-

5.3 Vier konkrete Substrat-Anfragesprachen

terklasse namens NRQL-ABox-Substrat (NRQL-ABox-Substrate) von der Klasse RACERPRO-BAA-Substrat (RACERPRO-BAA-Substrate) abgeleitet werden, die Caching-Funktionalität und insbesondere auch derartige Indexstrukturen für RACERPRO-ABoxen realisierte. Wir pflegen diese Index- und Cache-Strukturen jedoch erst für NRQL nach (s.u.).

5.3.3. (Leitdomäne 2) Der SWIR-Dienst und NRQL

NRQL ist eine hybride Anfragesprache, die eine für RACERPRO erweiterte BAA mit der DSA kombiniert, und die in Kap. 3.2.4 identifizierten Anforderungen an eine Anfragesprache für einen SWIR-Dienst (auf RACERPRO-Basis) umsetzt.

Wir hatten in Kap. 3.2.4 motiviert, dass es aus Entwurfs- und pragmatischen Gründen sinnvoll ist, durch NRQL „zwei Fliegen mit einer Klappe zu schlagen“, und daher nicht eine Sprache wie OWL-QL als Basis für einen SWIR-Dienst zu wählen, sondern eine erweiterte ABox-Anfragesprache. Wir entwerfen und realisieren NRQL daher *primär* als ausdrucksstarke ABox-Anfragesprache, die die BAA um OWL-spezifische Aspekte erweitert, und so die Anforderungen an einen SWIR-Dienst umsetzt.

NRQL wurde z.T. auch durch Nutzer des RACERPRO-Systems motiviert, die eine ausdrucksstarke ABox-Anfragesprache forderten, z.B. [Van Der Straeten u. a. 2003]. Zu diesem Zeitpunkt war die erforderliche Basis-Funktionalität des Rahmenwerks jedoch bereits vorhanden, s. [Wessel 2003b]. Die im Rahmen dieser Arbeit durchgeführte Entwicklung und anschließende Integration von NRQL in RACERPRO befriedigte auch diese Anforderungen. Die *Weiterentwicklung* von NRQL wurde ebenfalls z.T. durch die ersten Nutzer motiviert, so z.B. die einfachen Regeln. Wir möchten hier insbesondere Ragnhild v.d. Straeten erwähnen, s. auch [Haarslev u. a. 2004b].

Damit das SWIR-Szenario nicht zu abstrakt bleibt, stellen wir auch den von Studierenden der TUHH entwickelten OWL TREE als prototypische Webanwendung für einen SWIR-Dienst auf NRQL- und RACERPRO-Basis vor, welche den LUBM zur Realisierung eines SEMANTIC WEB-basierten Universitäts-ISS nutzt.

5.3.3.1. Erweiterte und detailliertere Anforderungen an NRQL

Bereits die BAA realisiert einen Großteil der in Kap. 3.2.4 als wünschenswert identifizierten Sprachmittel, wie gemeinsame Verfügbarkeit von NAF-Negation und klassischer Negation in einer Sprache (s. S. 180), usw. Im Vergleich zur BAA muss NRQL nun jedoch noch drei neue Gruppen von Anforderungen befriedigen:

1. Alle von RACERPRO gebotenen ABox-Spezifika, insbesondere auch die sich

durch die *konkrete Domäne* ergebenden, müssen mit NRQL abgedeckt bzw. adressiert werden können (die BAA bietet keine spezifische Unterstützung der konkreten Domäne). So soll es u.a. möglich sein, die in der konkreten Domäne literal spezifizierten Datenwerte, sog. *Told Values (TVs)* (s.u. für eine genauere Definition), durch NRQL-Anfragen zu ermitteln.

Insbesondere werden wir auch eine neue Art von Kanten-Atomen, die sog. *Constraint-Atome*, zur Adressierung der konkreten Domäne bereitstellen. Diese sind den $\text{:satisfies-}\lambda$ -Atomen (s. Kap. 5.2.2.8) ähnlich.

2. Die DSA bietet prinzipiell die Möglichkeit, die in Kap. 3.2.4 als wünschenswert identifizierten erweiterten Filter-Prädikate für TVs bereitzustellen; die DSA muss für NRQL so genutzt bzw. instantiiert werden, dass sie sowohl für OWL-Dokumente, als auch für „normale“ RACERPRO-ABoxen (mit konkreter Domäne) eine Bereicherung darstellt bzw. erweiterte Anfragefunktionalität bietet.

Für OWL-Dokumente bedeutet dies, dass wir mit der DSA erweiterte Filter-Prädikate zur Formulierung von Anfragebedingungen an literale Füller (TVs) von OWL-Datatype und/oder OWL-Annotation Properties bereitstellen wollen. Die gleiche durch die DSA gebotene Zusatzfunktionalität soll prinzipiell aber auch für die konkrete Domäne genutzt werden können.

RACERPRO repräsentiert den extensionalen Anteil eines OWL-Dokuments als ABox. Teilweise werden hierbei komplizierte Umkodierungen vorgenommen, insbesondere zur Repräsentation der literalen XSD-Datenwerte bzw. TVs eines OWL-Dokuments. Um diese Komplexität vom Nutzer zu verbergen, erweitern wir für NRQL die Menge der *Anfragekonzepte* (s. Def. 94, S. 323) gegenüber $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ und stellen eine erweiterte Syntax bereit, die eine nutzerfreundlichere Anfrageformulierung bzw. Abstraktion von diesen Aspekten gestattet.

3. Bereits die Basis-Anfragebeantwortungsmaschine (s. Kap. 5.2) realisiert die für einen SWIR-Dienst relevanten Anforderungen *inkrementelle Anfragebeantwortung und Mehrbenutzerfähigkeit*.

Zusätzlich hatten wir in Kap. 3.2.4 motiviert, dass auch *unvollständige* und *zweiphasige Anfragebearbeitungsmodi* im SWIR-Szenario bedeutsam sind. Für NRQL sehen wir daher solche Modi vor, die wir hier (formal fundiert) entwerfen.

Wir diskutieren diese Anforderungen nun detaillierter.

5.3.3.1.1. Unterstützung der konkreten Domäne Wir hatten in Kap. 3.2.4 auf S. 180 an OWL-QL (bzw. OWL) u.a. kritisiert, dass

5.3 Vier konkrete Substrat-Anfragesprachen

- keine benutzerdefinierten XML-Schemadatentypen möglich sind, und dass
- lediglich unäre Prädikate existieren.

So ist es in OWL-QL nicht möglich, eine Anfrage zu schreiben,

1. die Personen, welche älter als 18 sind, oder
2. Personen gleichen Alters

ermittelt. Beides soll mit NRQL möglich sein:

1. RACERPRO implementiert $ALCQHL_{\mathcal{R}^+}(D^-)$ und bietet somit den *eingeschränkten Prädikats-Existenz-Operator* $\exists(A_1, \dots, A_n).P$, s. Kap. 2.2.3.4. Durch Verwendung eines Konzept-Atoms der Art $(x, \exists(age). \geq_{18})$ ist es bereits mit der BAA möglich, 1. zu erfüllen. Wir hatten in Kap. 3.2.4, S. 183 diskutiert, dass einige (für eine Anfragesprache) bedeutsame Anfrageprädikate in der konkreten Domäne von RACERPRO fehlen. Diese werden durch die DSA bereitgestellt und weiter unten erläutert.

Betrachten wir zur Illustration einen Ausschnitt aus der Beispiel-ABox aus Kap. 2.2.5 (s. auch Kap. 2.2.3.4):

$$\{michael : person, (michael, michael\_age) : age, (michael\_age) : =_{35} .\}$$

Hierbei ist *michael* ein ABox-Individuum, *michael_age* ein KD-Individuum, $(michael\_age) : =_{35}$ eine *Prädikats-Assertion*, und $=_{35}$ ist ein *KD-Prädikat*.

Schließlich kann der gleiche Sachverhalt auch noch durch eine Konzept-Assertionen der Art

$$\{michael : \exists(age). =_{35}\}$$

modelliert werden. Auch in diesem Falle möchte man 35 als expliziten Wert einer Anfrage ermitteln können. Offensichtlich “fehlt” hier sogar ein explizites KD-Individuum (*michael_age*). Es sollte aus der Perspektive der Anfragesprache irrelevant sein, in welcher Form 35 in der ABox repräsentiert wird.

Literale Datenwerte wie 35, die syntaktisch explizit (und somit in allen Modellen impliziert) sind, nennen wir im Folgenden *Told Values (TVs)*.

Offensichtlich ist nun (in beiden Fällen)

$$(x, \exists(age). \geq_{18})^{\mathcal{E}} = \{michael\}.$$

Da Variablen (wie in der BAA) ausschließlich gegen ABox-Individuen (die Elemente $\text{inds}(\mathcal{A})$), aber nicht an KD-Individuen (die Elemente $\text{kd\_inds}(\mathcal{A})$) gebunden werden können, können wir kein ABox-Knoten-Atom der Art $(x, \geq_{18})$ bereitstellen, welches *x* gegen 35 binden könnte. Zur Lösung des Problems werden wir unten die sog. *Kopf-Projektionsoperatoren (KPOs)* bereitstellen; diese ermöglichen die Ermittlung solcher TVs. KPOs kommen ausschließlich im Kopf einer NRQL-Anfrage vor. Somit wird die Semantik der Variablen für NRQL beibehalten. Wir argumentieren unten, dass

eine veränderte Variablen-Semantik zu semantischen Problemen führen würde.

Mittels der folgenden Fixpunkt-Iteration können wir die TVs zudem *maximieren* (die Schleife wird durchlaufen, bis $\mathcal{A}_n = \mathcal{A}_{n+1}$ gilt):

```

n := 0,  $\mathcal{A}_0 := \mathcal{A}$ 
loop
  n := n + 1
   $\mathcal{A}_n =_{def} \mathcal{A}_{n-1} \cup \{ (b) :=_m \mid (a) :=_m \in \mathcal{A}_{n-1}, (a, b) := \in \mathcal{A}_{n-1} \}$ 
     $\cup \{ (a_A) :=_m, (a, a_A) : A \mid a : \dots \sqcap \exists(A). =_m \sqcap \dots \in \mathcal{A}_{n-1} \}$ 
     $\cup \{ (a, b) := \mid (b, a) := \in \mathcal{A}_{n-1} \}$ 
     $\cup \{ (a, b) := \mid (a, c) := \in \mathcal{A}_{n-1}, (c, b) := \in \mathcal{A}_{n-1} \}$ 
until  $\mathcal{A}_n = \mathcal{A}_{n-1}$ 

```

Zusätzlich können die so maximierten TVs auch in ein mit der ABox assoziiertes Daten-Substrat gespeichert und dann mittels der DSA erfragt werden; NRQL wird dann als hybride Anfragesprache genutzt.

Das Daten-Substrat wird durch *Spiegeln* der ABox bevölkert. Beim Spiegeln werden die Konzept-Assertionen analysiert; Konzepte, die TVs repräsentieren (z.B. $\exists(A). =_n$ im obigen Beispiel) werden identifiziert. Für diese werden dann im Daten-Substrat entsprechende explizite Knoten (und auch Kanten) angelegt, sodass Retrieval mit der DSA ermöglicht wird. Da TVs nun als Knoten im Daten-Substrat existieren, können auf diesem Substrat tatsächlich Variablen gegen TVs gebunden werden. Dies ist nur mit Datensubstrat-Variablen in einer hybriden NRQL-Anfrage möglich, nicht jedoch mit den ABox-Variablen (weil dies zu semantischen Problemen führen würde).

Da die Verwendung des assoziierten Daten-Substrates jedoch optional sein sollte (schließlich ist das Spiegeln und Analysieren ein zeitaufwendiger Prozess), bieten wir eine alternative Lösungsvariante an, die besagten KPOs. Ohne assoziiertes Daten-Substrat können zudem die durch die DSA bereitgestellten erweiterten Anfrageprädikate nicht genutzt werden.

2. Mittels Prädikats-Assertionen (der Art $(a_1, \dots, a_n) : P$) ist es möglich, n -stellige Beziehungen zwischen KD-Individuen zu etablieren, s. Kap. 2.2.5: $(birgit\_age, michael\_age) : >$. Wir beschränken uns auf binäre Prädikate.

Offensichtlich wird nun ein neues binäres Anfrage-Atom benötigt, um Paare von Individuen bzw. Personen zu ermitteln, die bzgl. ihres Alters in einer $<$ -Beziehung stehen (oder die gleich alt sind). Wir müssen hierfür eine neue Art von Kanten-Atom vorsehen, die sog. *Constraint-Atome*.

RACERPRO bietet eine API-Funktion namens `constraint-entailed-p` zur Entscheidung des *eingeschränkten Constraint-Prüfungsproblems*, s.

5.3 Vier konkrete Substrat-Anfragesprachen

Kap. 2.2.3.4, S. 92. Offensichtlich gilt in der Beispiel-Wissensbasis WB nun

$$WB \models (michael\_age, birgit\_age) :<$$

und daher $constrained\_entailed?(michael\_age, birgit\_age, <) = \text{TRUE}$.

Wie auch unter 1. diskutiert, sollen nun binären Prädikate wie $<$ zwischen Attributfüllern von ABox-Individuen geprüft werden können, unabhängig davon, ob für diese Attributfüller explizit angelegte KD-Individuen existieren oder nicht.

Das Constraint-Atom (a, b, w) wird in der Kanten-Anfragebedingung $w \in \mathcal{L}_{QE}$ nun die Attribute und das KD-Prädikat nennen; die möglichen Anfrage-Prädikate sind durch die möglichen RACERPRO-KD-Prädikate vorgegeben. Offensichtlich muss die $constrained\_entailed$ -p-Funktion zur Implementierung der ABFen des Constraint-Atoms verwendet werden, welches nur solche Prädikate verarbeiten kann.

Erweiterte Anfrage-Prädikate für das Retrieval von TVs können wir wiederum durch das optionale mit der ABox assoziierte Daten-Substrat bereitstellen, sodass wir auf diesem die Kanten-Atome der DSA mit Bedingungen aus $\mathcal{L}_{SKNF_{PE}}$ nutzen können.

5.3.3.1.2. Spezifische OWL-Unterstützung RACERPRO repräsentiert den extensionalen Anteil eines OWL-Dokuments (den RDF-Instanzen-Graph) in einer ABox. Insbesondere zur Repräsentation von TVs, welche Füller von Datatype Properties sind (sog. *TV-Füller*), werden komplizierte Umkodierungen bzw. Abbildungen auf die konkrete Domäne vorgenommen. Die technischen Einzelheiten dieser Kodierung sollten für den Nutzer von NRQL transparent sein. Wir können eine nutzerfreundlichere Syntax bereitstellen, indem wir die im Rahmenwerk bereits vorgesehenen Möglichkeiten zur *Anfrageumschreibung* nutzen.

Ist in einem OWL-Dokument z.B. 35 als TV-Füller des Datatype Properties *age* vom Typ `XMLSchema#int` für das Individuum `michael` mittels

```
<person rdf:ID="michael">
  <age rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >35</age>
</person>,
```

angegeben, so wird dies in der ABox durch eine Konzept-Assertion

$$michael : \exists age. (\exists racer\_internal\%has\_integer\_value. =_{35})$$

repräsentiert. Hier ist `racer-internal%has-integer-value` ein Attribut mit Bildmenge `integer`, und *age* eine normale Rolle, die zur Repräsentation des Datatype Properties verwendet wird.

Ohne Anfrageumschreibung wäre es daher mühsam, Anfragen bzgl. Datatype Properties zu formulieren: Zur Ermittlung der volljährigen Personen müssten wir nun ein Atom der Art

$$(x, \exists age. (\exists racer - internal \% has - integer - value. \geq_{18}))$$

verwenden, bzw. in konkreter Syntax:

(?x (some age (min racer-internal%has-integer-value 18))).

Wir stellen für NRQL daher eine erweiterte Syntax für die *Anfragekonzepte* bereit, sodass auch der Ausdruck (min age 18) und somit das nutzerfreundlichere Konzept-Atom (?x (min age 18)) verwendet werden kann.

Die zum Retrieval von TV-Füllern von Attributen anwendbaren KPOs sollen zudem auch für TV-Füller von Datatype und/oder Annotation Properties anwendbar sein.

Auch im Falle von OWL profitieren wir vom assoziierten Daten-Substrat, wie bereits beschrieben. Das durch Spiegeln eines OWL-Dokuments erzeugte Daten-Substrat wird nun verschiedene „Arten“ bzw. durch spezielle Annotationen besonders ausgezeichnete Knoten enthalten, sodass wir mit der DSA anhand dieser Annotationen unterscheiden können, ob ein Knoten ein Individuum, ein TV-Füller eines Annotation Properties, ein TV-Füllers eines Datatype Properties usw. darstellt.

Wir hatten *Evaluierung des LUBM*, s. Kap. 3.2.4.1, mit auf die Liste der Anforderungen für NRQL gesetzt. Wir schicken voraus, dass die Expressivität von NRQL mehr als ausreichend für die LUBM-Anfragen ist.

5.3.3.1.3. Modi zur unvollständigen und zweiphasigen Anfragebearbeitung Wir hatten die verwendeten Begriffe *unvollständige* und *zweiphasige Anfragebearbeitung* in Kap. 3.2.4 erläutert und motiviert, dass entsprechende Modi für einen SWIR-Dienst und auch eine ABox-Anfragesprache per se bedeutsam sind. Wir fassen zusammen:

Zur Beantwortung einer BAA- (und somit NRQL-) Anfrage werden ABox-Inferenzen erforderlich, welche i.d.R. einen oder sogar eine ganze Reihe von ABox-Erfüllbarkeitstests benötigen, die potentiell sehr teuer sind. Unvollständige Modi sind unter dem Gesichtspunkt der Ressourcenkonfigurierbarkeit interessant. Zudem kann durch ihre Bereitstellung eine zweiphasige Anfragebearbeitung realisiert werden („erst billige, dann teure Tupel bei Bedarf“). Unvollständige Modi werden zudem vollständig für eingeschränkte DLs bzw. Wissensbasen, die nicht den vollen Sprachumfang nutzen (tatsächlich ist durch den LUBM eine solche WB gegeben). Aus rein pragmatischen Gründen ist ein unvollständiges Anfrageergebnis oftmals „vollständig genug“ für ein bestimmtes Anwendungsszenario. Unvollständige Modi sind auch unter dem Gesichtspunkt *sehr große oder komplexe WBs* relevant, da für diese u.U. aufgrund der

5.3 Vier konkrete Substrat-Anfragesprachen

Komplexität oder Größe mittels vollständiger Methoden keine Ergebnisse (in einem vernünftigen Zeitrahmen) geliefert werden können. In diesem Fall stehen immer noch die unvollständigen Modi zur Verfügung, welche i.d.R. zumindest einige Antworten errechnen können (was natürlich von der WB abhängt).

Die Basis-Idee der unvollständigen Modi ist einfach – die Modi inspizieren und werten die vorhandenen ABox-Assertion aus; man spricht auch von *Told* oder *Asserted Information*. Es handelt sich um einfache Inferenzverfahren, die sehr effizient durchgeführt werden können:

- Für positive Rollen-Atome (x, y, R) hatten wir mit Lemma 5, S. 323 einen Algorithmus vorgestellt, der rein syntaktisch arbeitet (und somit hocheffizient ist) und $(x, y, R)^\varepsilon$ approximiert. Die Approximation ist bis zur DL $\mathcal{ALCHIT}_{\mathcal{R}^+}$ sogar perfekt, also vollständig.
- Für Konzept-Atome können wir $(x, C)^\varepsilon$ approximieren, indem wir (auf syntaktischer Ebene) die $i : C$ -Konzept-Assertionen in der ABox identifizieren:

$$\{i \mid i : C \in \mathcal{A}\} \subseteq (x, C)^\varepsilon.$$
- Diese Approximation von $(x, C)^\varepsilon$ wird noch besser bzw. „vollständiger“, wenn die TBox-Axiome genutzt werden. Wir fordern daher, dass die Taxonomie der TBox zuvor berechnet wurde.

Gilt nun $\mathcal{T}_A \models D \sqsubseteq C$, so können auch Assertionen der Art $i : D$ zur Approximation von $(x, C)^\varepsilon$ genutzt werden:

$$\{i \mid i : D \in \mathcal{A}, \mathcal{T}_A \models D \sqsubseteq C\} \subseteq (x, C)^\varepsilon.$$

NRQL bietet mehrere Variationen dieses Themas, die sich hinsichtlich der Komplexität der für die Approximation berücksichtigten TBox-Axiome $D \sqsubseteq C$, also bzgl. C und D , unterscheiden.

Wie auch für die Rollen-Atome gilt, dass diese Approximation für einfache WBs perfekt bzw. vollständig sein kann. In diesen Fällen kann beim Retrieval vollständig auf eine Inanspruchnahme der teuren ABox-Retrievaldienste verzichtet werden kann; wir möchten darauf hinweisen, dass auch die anderen Substrate (z.B. das Daten-Substrat) letztlich Variationen dieses Themas sind.

Prinzipiell ist die Dimension „Grad der Vollständigkeit“ orthogonal zur (bereits in der BAA manifestierten) Dimension „mengenbasiert vs. tupelbasiert“. Die möglichen *zweiphasigen Anfragebeantwortungsmodi* in diesem durch die beiden Achsen aufgespanntem Raum sind nun wie folgt gegeben:

- In *Phase 1* eines zweiphasigen Modus' wird zunächst ein *unvollständiger Modus* zur Berechnung der billigen Tupel verwendet.
- In *Phase 2* eines zweiphasigen Modus' werden dann die *verbleibenden Tupel* mit vollständige Methoden zur Vervollständigung des Anfrageergebnisses berechnet.

- Wird ein zweiphasiger Modus *inkrementell bzw. tupelbasiert* verwendet, so kann der Nutzer beim Übergang von Phase 1 zu Phase 2 durch die Auslieferung einer Warnung bzw. eines speziellen Benachrichtigungs-Symbols darüber informiert werden, dass die Berechnung des nächsten Tupels eventuell sehr viel länger dauern wird. Es obliegt dann dem Nutzer, Phase 2 der Anfragebearbeitung nach Erhalt der Benachrichtigung zu initiieren, oder die Anfrage zu beenden.
- Wird der zweiphasige Modus *mengenbasiert* verwendet, so wird der Nutzer von der internen zweiphasigen Organisation nichts merken. Eine intern zweiphasig organisierte Anfragebeantwortung kann jedoch *unnötige Anfragen* an RACERPRO vermeiden: Wurde bereits $i \in (x, C)^\varepsilon$ in Phase 1 errechnet, so muss i nicht erneut in Phase 2 betrachtet werden. RACERPRO unterstützt die Spezifikation von zu betrachtenden Kandidatenmengen für einige ABox-Retrievalfunktionen (s.u.).

5.3.3.2. Entwurf des Substrates für NRQL

Die Substratklassen für NRQL sind in Abb. 5.19 dargestellt. NRQL benötigt eine Instanz einer konkreten Unterklasse der abstrakten Klasse NRQL-Substrat (NRQL-Substrate). Von dieser existieren drei konkrete Unterklassen: Die Klasse NRQL-ABox-Substrat (NRQL-ABox-Substrate) zur ABox-Repräsentation, die Klasse NRQL-Daten-Substrat (NRQL-Data-Substrate) zur Repräsentation des (optional mit der ABox) assoziierten Daten-Substrates, und die Klasse NRQL-Hybrid-Substrat (NRQL-Hybrid-Substrate), welche zwei Instanzen der ersten beiden Klassen kombiniert. Wird NRQL auf einem NRQL-Hybrid-Substrat (NRQL-Hybrid-Substrate) genutzt, so steht die DSA zur Verfügung, und NRQL ist hybrid. Die in Abb. 5.19 aufgezeigten Vererbungsbeziehungen und Assoziationen sind selbsterklärend.

Wir beschreiben nun den Entwurf der nichthybriden Klassen detaillierter.

5.3.3.2.1. Das NRQL-ABox-Substrat Das nun zu entwerfende NRQL-ABox-Substrat (NRQL-ABox-Substrate) wird umfangreiche Caching-Funktionalität bereitstellen, zur Vermeidung von

1. *repetitiven* und
2. *unnötigen*

RACERPRO-Anfragen wann immer möglich. Dies ist umso dringlicher im Mehrbenutzerbetrieb eines SWIR-Dienstes, da das RACERPRO-API synchron arbeitet und RACERPRO andernfalls unnötigerweise blockiert wäre. Wir erhöhen somit die Verfügbarkeit.

5.3 Vier konkrete Substrat-Anfragesprachen

Repetitive Anfragen treten insbesondere dann auf, wenn die gleiche API-Funktion auf unveränderter WB erneut aufgerufen wird, z.B. eine Instanzenretrieval-Anfrage. Das Ergebnis eines solchen API-Aufrufs wird daher memoisiert bzw. in einem Cache gespeichert, und ein entspr. Protokoll zur Lösung des *Cache-Synchronisationsproblems* wird vorgesehen.

Natürlich sind repetitive Anfragen auch unnötig. *Unnötige Anfragen* an RACERPRO sind solche Anfragen, zu deren Beantwortung eigentlich keine (oder nur schwache) Inferenz erforderlich ist, und die somit lokal beantwortet werden könnten, wie z.B. die Prüfung von Folgerbarkeit für Rollen-Assertionen bis zur Sprache $\mathcal{ALCHIT}_{\mathcal{R}^+}$, s. Lemma 5, S. 323. Die effiziente relationale Traversierung der ABox anhand der folgerbaren Rollen-Assertionen durch die ABFen der Rollen-Atome muss durch Cache- und Indexstrukturen effizient unterstützt werden, wie bereits in Kap. 5.3.2.2 angesprochen.

Um die R -Nachfolger oder -Vorgänger eines Knotens i schnell ermitteln zu können, wird nicht etwa in einer Liste von Rollen-Assertionen gesucht, sondern es wird eine Hashtabelle mit Schlüssel (i, R) verwendet, sodass das Auffinden der Liste der R -Nachfolger von i eine zeitkonstante Operation ist; offensichtlich sind dann diese Operationen ähnlich effizient implementierbar wie in einer objektgraphbasierten Substrat-Repräsentation (das NRQL-ABox-Substrat (NRQL-ABox-Substrate) ist ja ein virtuelles Substrat; dementsprechend existiert kein lokaler Objektgraph, der traversiert werden könnte).

Die lokalen Cache- und Indexstrukturen der Klasse NRQL-ABox-Substrat (NRQL-ABox-Substrate) ermöglichen die einfache Bereitstellung der unvollständigen Modi. Üblicherweise wird im Falle einer Anfrage, die nicht anhand der Cache-Einträge beantwortet werden kann, ein *Cache Fail* ausgelöst, der dann zur Weiterdelegation an die entspr. RACERPRO-API-Funktion zur Beantwortung der Anfrage und anschließender Eintragung des Ergebnisses in den Cache führt.

Wird nun ein unvollständiger Modus zur Anfragebeantwortung verwendet, so wird ein zur *Anfragebeantwortungszeit ausgelöster Cache Fail* einfach nicht weiterdelegiert. Die initialen Cache-Einträge werden in einer Initialisierungsphase des Substrates anhand der von RACERPRO erfragten ABox-Assertionen durch entsprechendes Spiegeln und Aufbereiten erzeugt. Diese Informationen können nun je nach Vollständigkeit des verwendeten Modus noch um weitere, lokal inferierte Informationen angereichert werden (durch Berücksichtigung von TBox-Axiomen, s.o.). Wir geben unten eine formale Beschreibung dieser Anreicherungsprozesse an.

5.3.3.2.2. Das NRQL-Daten-Substrat Wird das hybride NRQL-Hybrid-Substrat (NRQL-Hybrid-Substrate) verwendet, so wird zusätzlich zum NRQL-ABox-Substrat (NRQL-ABox-Substrate) ein NRQL-Daten-Substrat (NRQL-Data-Substrate) als *kompletter Spie-*

gel der Assertionen der befragten RACERPRO-ABox angelegt. Im Unterschied zum NRQL-ABox-Substrat (NRQL-ABox-Substrate) enthält das NRQL-Daten-Substrat (NRQL-Data-Substrate) der ABox auch *KD-Individuen* und *TVs* als Knoten (plus entsprechender Kanten zwischen diesen). Im NRQL-Daten-Substrat (NRQL-Data-Substrate) finden sich dementsprechend drei Arten von Knoten:

- Knoten, die ABox-Individuen (Elemente aus $\text{inds}(\mathcal{A})$) repräsentieren;
- Knoten, die KD-Individuen (Elemente aus $\text{kd\_inds}(\mathcal{A})$) repräsentieren, und
- Knoten, die *literale Datenwerte (TVs)* (Werte aus $\mathcal{D}$) repräsentieren.

Die Knoten behalten jeweils ihre Namen aus der ABox; im Falle eines TVs ist der Name des Knotens durch das Literal selbst gegeben. Knoten, die TVs darstellen, enthalten das Literal (bzw. ihren Namen) selbst als Konjunkt (reflexive Annotation). Knoten, die ABox-Individuen i repräsentieren, führen als Konjunkte in $L_V(i) \in \mathcal{L}_{SKNF}$ zudem alle atomaren Konzepte auf, von denen sie *bekannte* Instanzen sind.

Zur Kennzeichnung der Art bzw. des „Typs“ eines Knotens verwenden wir entspr. *Annotations-Markierungssymbole* (wodurch die Knoten *selbstbeschreibend* werden):

- `:abox-object` als Oberbegriff für
 - `:abox-individual`,
 - `:abox-concrete-domain-object`,
 - `:abox-concrete-domain-value`.

Für einen KD-Knoten i würde daher

$$\{\text{:abox-concrete-domain-object}, \text{:abox-object}\} \subseteq L_V(i)$$

gelten.

Speziell für ABoxen, die durch Einlesen von OWL-Dokumenten entstehen (sog. „OWL-ABoxen“), werden noch weitere Markierungssymbole eingeführt:

- `:owl-datatype-value` (zur Kennzeichnung von XSD-Literalen, die Füller bzw. TVs von Datatype Properties sind),
- `:owl-annotation` als Oberbegriff für
 - `:owl-annotation-datatype-object` (zur Kennzeichnung des annotierten Objektes),
 - `:owl-annotation-value` (zur Kennzeichnung des annotierenden Wertes bzw. XSD-Literales).

Analog zu den Knoten gibt es mehrere Arten von Kanten, je nachdem, ob eine Kante eine

5.3 Vier konkrete Substrat-Anfragesprachen

- Rollen-Assertion $((i, j) : R, i, j \in \text{inds}(\mathcal{A}), R \in \mathcal{N}_{\mathcal{R}})$, eine
- Attributs-Assertion $((i, k) : A, i \in \text{inds}(\mathcal{A}), k \in \text{kd\_inds}(\mathcal{A}), A \in \mathcal{N}_{\mathcal{A}})$, oder eine
- KD-Individuum-TV-Beziehung $((k) :=_n \text{ für entspr. Gleichheitsprädikat}, k \in \text{kd\_inds}(\mathcal{A}), n \in \mathcal{D} \text{ der entspr. konkreten Domäne})$

darstellt. Wir verwenden entsprechende Markierungssymbole für die Kanten-Beschreibungen $L_E((i, j))$:

- `:abox-relationship` als Oberbegriff für
 - `:abox-role-relationship`,
 - `:abox-attribute-relationship`,
 - `:abox-told-value-relationship`.

Durch diese Annotationen werden nun eine Art von „ABox-Meta-Anfragen“ mit der DSA möglich: Beispielsweise können wir mit einem Kanten-Atom wie $(?x ?y : \text{abox-role-relationship})$ alle Paare von Individuen ermitteln, die in *irgendeiner* Beziehung zueinander stehen.

Für OWL-ABoxen verwenden wir dann zusätzlich

- `:owl-object-relationship` (für Object Properties, welche normalen Rollen entsprechen),
- `:owl-datatype-relationship` (für Datatype Properties),
- `:owl-annotation-relationship` als Oberbegriff für
 - `:owl-annotation-object-relationship` (für annotierte Object Properties),
 - `:owl-annotation-datatype-relationship` (für annotierte Datatype Properties).

Zur Population der Instanz der Klasse `NRQL-Daten-Substrat` (`NRQL-Data-Substrate`) müssen u.U. Knoten und Kanten erzeugt werden, für die es keine expliziten Entsprechungen in der ABox gibt:

- *Annotations-Werte* (*Markierungssymbol* `:annotation-value`) tauchen z.B. gar nicht als Assertionen in der RACERPRO-ABox auf, da keine Inferenz mit diesen stattfindet. Sie werden von RACERPRO lediglich auf Anforderung mittels einer speziellen API-Funktion zurückgegeben und sind sonst „unsichtbar“. Nichtsdestotrotz sollten sie zum Retrieval herangezogen werden können.
- Ähnlich verhält es sich für Füller von Datatype Properties, die lediglich implizit durch Assertionen wie
$$michael : \exists age. (\exists racer\text{-}internal \% has\text{-}integer\text{-}value. =_{35})$$
in der ABox repräsentiert werden (s.S. 338).

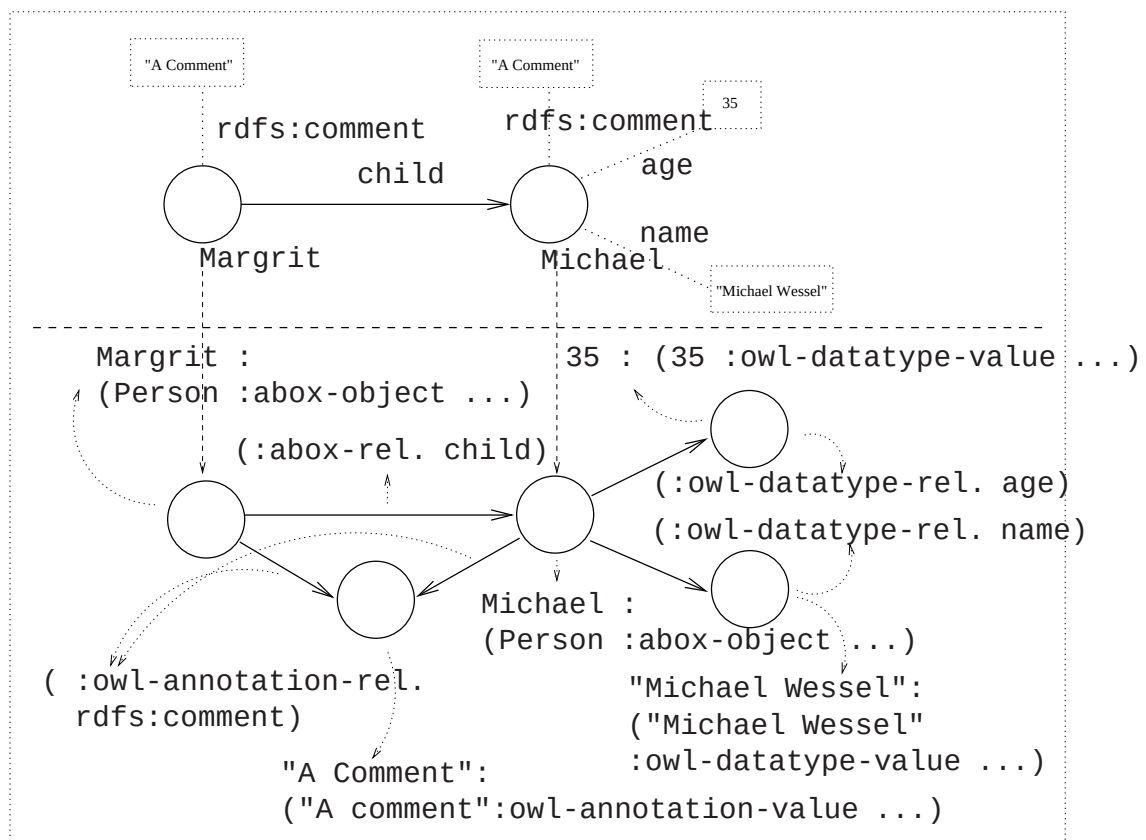


Abbildung 5.20.: Hybride Repräsentation eines OWL-Dokuments mit einem NQL-Hybrid-Substrat (NQL-Hybrid-Substrate): Oben NQL-ABox-Substrat (NQL-ABox-Substrate), unten NQL-Daten-Substrat (NQL-Data-Substrate).

Wir erzeugen für diese Konzept-Assertionen beim Initialisieren der Instanz der Klasse NQL-Daten-Substrat (NQL-Data-Substrate) durch Analyse der Assertionen nun ebenfalls entsprechende Knoten und Kanten zur expliziten Repräsentation der Füller und Kanten, die durch die Existenzquantoren lediglich implizit repräsentiert werden.

Für 35 wird somit ein Knoten namens 35 angelegt, der sowohl mit 35 als auch mit :owl-datatype-value annotiert wird, etc.

Zur Verdeutlichung zeigt Abb. 5.20 die hybride Substrat-Repräsentation des folgenden OWL-Dokuments:

```
<person rdf:ID="margrit">
  <child>
    <person rdf:ID="michael">
      <age rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
```

5.3 Vier konkrete Substrat-Anfragesprachen

```
>35</age>
<name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>Michael Wessel</name>
<rdfs:comment
      rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >A Comment</rdfs:comment>
</person>
</child>
</person>
</rdf:RDF>
```

Hier sind name und age Datatype Properties, und rdfs:comment ist ein Standard-OWL bzw. Standard-RDF(S) Annotation Property. Bei child handelt es sich um ein Object Property.

Wir werden die durch die DSA ermöglichten erweiterten Anfragemöglichkeiten weiter unten durch Beispielanfragen demonstrieren.

5.3.3.3. Entwurf der NRQL-Anfragesprache

NRQL ist eine hybride Anfragesprache. Sie kombiniert eine erweiterte BAA mit der DSA:

- Der ABox-spezifische Teil von NRQL ist eine Spezialisierung der BAA. Als solche unterstützt sie:
 - Konzept-Atome, wobei $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-) \subseteq \mathcal{L}_{\mathcal{QV}}$ gilt.
 $\mathcal{L}_{\mathcal{QV}}$ bietet eine nutzerfreundlichere Syntax zur Formulierung von Anfragebedingungen an Füller von Datatype und Annotation Properties, wie bereits auf S. 339 beschrieben. Die erweiterte Menge der Anfragekonzepte nennen wir im Folgenden *NRQL-Anfragekonzepte*.
 - Rollen-Atome, wie in der BAA.
 - Die bereits auf S. 337 motivierten *Constraint-Atome*. Als erstes Beispiel für ein solches Atom sei $(x, y, \exists(\text{age}, \text{age}). <)$ angegeben, welches das auf S. 337 diskutierte Anfrageformulierungsproblem löst. Wir werden diese Atome weiter unten formal definieren.

Eine weitere Besonderheit gegenüber der BAA sind die bereits erwähnten *Kopf-Projektionsoperatoren (KPOs)* zur Ermittlung von TVs.

- Der Anteil von NRQL, der sich auf das Daten-Substrat bezieht, ist identisch mit der DSA.

In hybriden NRQL-Anfragen können die beiden Arten von Atomen beliebig kombiniert bzw. „gemischt“ werden, s. Kap. 5.2.2.5, S. 301.

Wir entwerfen nun die „NRQL-Neuerungen“ in der Reihenfolge

1. Anfragekonzepte,
2. Constraint-Atome,
3. Kopf-Projektionsoperatoren (KPOs)

detaillierter.

5.3.3.3.1. NRQL-Anfragekonzepte

Definition 95 (NRQL-Anfragekonzept) Die Menge der NRQL-Anfragekonzepte ist wie folgt definiert:

- Jedes $ALCQHI_{\mathcal{R}^+}(\mathcal{D}^-)$ - bzw. RACERPRO-Konzept ist ein NRQL-Anfragekonzept.
- Wenn P eine aussagenlogische Formel über *einstelligen* Prädikatsnamen aus der konkreten Domäne ist, und R_{DTP} eine für ein OWL Datatype Property DTP erzeugte Rolle ist, dann ist auch $\exists(R_{DTP}).P$ ein NRQL-Anfragekonzept. ■

Hierbei handelt es sich also um eine Erweiterung des *eingeschränkten Prädikats-Existenz-Operators* aus $ALCQHI_{\mathcal{R}^+}(\mathcal{D}^-)$. Als Erweiterung werden hier nicht nur Attribute zugelassen, sondern auch Datatype Properties, die von RACERPRO als Rollen kodiert werden. Für den Nutzer bedeutet diese Erweiterung, dass er nun mit Datatype Properties wie mit Attributen in der Anfragesprache arbeiten kann, sodass die interne Kodierung (s. S. 338) aus Sicht des NRQL-Nutzers transparent wird. Zudem können nun *aussagenlogische Formeln als Prädikate* verwendet werden, was ebenfalls nützlich ist.

Zur Illustration betrachten wir das Anfragekonzept

$\exists|\text{http://www.owl-ontologies.com/unnamed.owl\#age}|.(\geq_{20} \sqcap \leq_{35} \sqcup =_{39})$

bzw. in konkreter (KRSS-artiger) Syntax

```
(some |http://www.owl-ontologies.com/unnamed.owl\#age|
  (or (and (min 20) (max 35)) (equal 39))).
```

Offensichtlich ist $\geq_{20} \sqcap \leq_{35} \sqcup =_{39}$ eine aussagenlogische Formel von unären Prädikaten, die ein nichtkonvexes Intervall $[20; 35] \cup \{39\}$ darstellt. RACERPRO selbst unterstützt keine solche komplexen Prädikate; das Anfragekonzept wird daher wie folgt auf ein von RACERPRO verarbeitbares Konzept umgeschrieben:

5.3 Vier konkrete Substrat-Anfragesprachen

$$\exists | \text{http://www.owl-ontologies.com/unnamed.owl\#age} | .$$

$$(\exists \text{racer-internal\%has-integer-value.} \geq_{20} \sqcap$$

$$\exists \text{racer-internal\%has-integer-value.} \leq_{35} \sqcup$$

$$\exists \text{racer-internal\%has-integer-value.} =_{39}),$$

bzw. in konkreter Syntax

```
(some |http://www.owl-ontologies.com/unnamed.owl#age|
  (or (equal racer-internal%has-integer-value 39)
    (and (min racer-internal%has-integer-value 20)
      (max racer-internal%has-integer-value 35))))
```

Offensichtlich ist diese Umschreibung aufgrund der Funktionalität (hier des Attributes `racer-internal%has-integer-value`) zulässig, sodass alle Prädikate, die sich auf ein Attribut beziehen, konjunktiv zusammengefasst werden können. Andererseits können komplexe aussagenlogische Prädikate stets durch Nutzung der Konzeptkonstruktoren $\{\neg, \sqcap, \sqcup\}$ auf die von RACERPRO angebotenen Basis-Prädikate syntaktisch heruntergebrochen werden, aufgrund der aussagenlogischen Vollständigkeit von $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$, und weil die Menge der Prädikate in der konkreten Domäne unter Negation abgeschlossen ist.

5.3.3.3.2. NRQL-Constraint-Atome

Definition 96 (NRQL-Constraint-Atom) Die allgemeine Form eines Constraint-Atoms lautet

$$(a, b, \exists(R_1 \circ \dots \circ R_n \circ A_1, S_1 \circ \dots \circ S_m \circ A_2).P),$$

wobei $a, b \in \mathcal{O}_S$ ist, und $n, m = 0$ möglich ist.

Hier sind R_i, S_j *Anfragerollen* (s. Def. 94, S. 323), $A_1, A_2 \in \mathcal{N}_A$ sind Attribute, und P ist ein *binärer* Prädikatsausdruck aus der konkreten Domäne, wie er in RACERPRO-ABoxen in ähnlicher Form (s.u.) auch in *constraints-* bzw. RACERPRO-Prädikats-Assertionen erlaubt ist. Die Ausdrücke $R_1 \circ \dots \circ R_n$ und $S_1 \circ \dots \circ S_m$ werden *Anfrage-Rollenketten* genannt. ■

Der Operator $\exists(\text{age}, \text{age}). =$ ist vergleichbar mit dem (vollen) Prädikats-Existenz-Operator aus $\mathcal{ALC}(\mathcal{D})$, oder auch mit dem *rollenbildenden Prädikats-Existenz-Operator* aus $\mathcal{ALCRP}(\mathcal{D})$ [Haarslev u. a. 1999]. Die Semantik der Ketten unterscheidet sich jedoch von der Semantik der Ketten in den entsprechenden Operatoren in $\mathcal{ALC}(\mathcal{D})$ und $\mathcal{ALCRP}(\mathcal{D})$. Zum einen lassen wir in den Anfrage-Rollenketten nicht nur funktionale Rollen (wie in $\mathcal{ALC}(\mathcal{D})$ oder $\mathcal{ALCRP}(\mathcal{D})$), sondern *Anfragerollen* zu (die somit auch negiert sein dürfen). Zum anderen haben die Anfrage-Rollenketten eine *autoepistemische Semantik*; eine Kette der Art $f_1 \circ \dots \circ f_n$ ist äquivalent zu einer Konjunktion von Rollen-Atomen. Das Atom

$$(a, b, \exists(R_1 \circ \dots \circ R_n \circ A_1, S_1 \circ \dots \circ S_m \circ A_2).P),$$

ist somit per Definitionem äquivalent zum Rumpf

$$\begin{aligned} & (a, a_1, R_1) \wedge \dots \wedge (a_{n-1}, a_n, R_n) \wedge \\ & (b, b_1, S_1) \wedge \dots \wedge (b_{m-1}, b_m, S_m) \wedge \\ & (a_n, b_m, \exists(A_1, A_2).P). \end{aligned}$$

Wir müssen daher lediglich Implementationen für *einfache* Constraint-Atome der Art $(a, b, \exists(A_1, A_2).P)$ bereitstellen; *komplexe* Constraint-Atome können durch syntaktisches Umschreiben dekomponiert werden.

Das binäre Prädikat P aus der konkreten Domäne kann wieder als prädikatenlogische Formel mit zwei *getypten* freien Variablen x und y begriffen werden: $P[x, y]$. Semantisch gilt $(i, j) \in (a, b, \exists(A_1, A_2).P)^{\mathcal{E}}$ gdw. $P_{x \leftarrow A_1(i), y \leftarrow A_2(j)}$ wahr wird. Hier wird also die freie Variable x in P gegen den Term $A_1(i)$ ersetzt, und y gegen $A_2(j)$. Wir können dies auch als $P(A_1(i), A_2(j))$ schreiben. Offensichtlich können wir zur Entscheidung von $\mathcal{A} \models P(A_1(i), A_2(j))$ das *Constraintprüfungs-Problem* (s. Kap. 2.2.3.4, S. 89) verwenden.

Betrachten wir zur Illustration einen Ausschnitt aus der Beispiel-WB aus Kap. 2.2.5:

$$\{ (michael, michael\_age) : age, (birgit, birgit\_age) : age, \\ (michael\_age) : =_{35}, (michael\_age, birgit\_age) : < \},$$

mit $age \in \mathcal{N}_{\mathcal{A}}$. Hier gilt $< (age(michael), age(birgit))$ und daher auch $(michael, birgit) \in (x, y, \exists(age, age). <)^{\mathcal{E}}$.

Die RACERPRO-Funktion *constraint-entailed-p* lässt nicht nur einfache Prädikatssymbole, sondern *komplexere Prädikatsausdrücke* zu. Gemäß RACERPRO-Handbuch [Racer Systems 2006, Kap. 3.2] unterstützt RACERPRO Prädikatsausdrücke der Bauart $P(aexpr_1, aexpr_2)$. Beispielsweise ist das Prädikat $michael\_age + 4 \leq birgit\_age$ bzw. $\leq (michael\_age + 4, birgit\_age)$ wahr, wenn $birgit\_age$ und $michael\_age$ um wenigstens 4 (Jahre) differieren.

Je nach Typ des referenzierten KD-Individuums sind für *aexpr* unterschiedliche Ausdrücke zulässig. Der Typ eines KD-Individuums $a \in \text{kd\_inds}(\mathcal{A})$ ergibt sich anhand des Typs des einführenden Attributes A , also anhand der Assertion $(i, a) : A \in \mathcal{A}$. Bereits bei der Definition des Attributes A wird die entspr. konkrete Domäne und somit der Typ für die Füller des Attributes (und daher auch der Typ von a) festgelegt, s. Kap. 2.2.8.1. Somit gilt $\text{typ}(a) =_{\text{def}} \text{range}(A)$. Bzgl. der Typen sind nun zwei Fälle für $aexpr_i$ möglich:

1. Die in $P(aexpr_1, aexpr_2)$ verwendeten KD-Individuen $a \in \text{kd\_inds}(\mathcal{A})$ haben numerische Typen, also $\text{typ}(a) \in \{\mathbb{C}, \mathbb{N}, \mathbb{Z}\}$.

Dann ist $P \in \{>, \geq, <, \leq, =, \neq\}$, und $aexpr_i$ haben die folgende Bauart. Sei $r \in \mathbb{R}$, $n \in \mathbb{N}$, $i \in \mathbb{Z}$:

5.3 Vier konkrete Substrat-Anfragesprachen

$$\begin{aligned}
 aexpr &::= a & | \\
 & r & | \\
 & expr1 \{+ expr1\}^* & | \\
 \\
 expr1 &::= expr2 & | \\
 & expr3 & | \\
 & expr5 & | \\
 \\
 expr2 &::= r & | \\
 & a \text{ für } \text{typ}(a) \in \{\mathbb{R}, \mathbb{C}\} & | \\
 & r * a \text{ für } \text{typ}(a) \in \{\mathbb{R}, \mathbb{C}\} & | \\
 \\
 expr3 &::= r & | \\
 & a \text{ für } \text{typ}(a) \in \{\mathbb{C}\} & | \\
 & i \{ * expr4 \}^+ & | \\
 \\
 expr4 &::= a \text{ für } \text{typ}(a) \in \{\mathbb{C}\} & | \\
 & a^n \text{ für } \text{typ}(a) \in \{\mathbb{C}\} & | \\
 \\
 expr5 &::= i & | \\
 & a \text{ für } \text{typ}(a) \in \{\mathbb{N}\} & | \\
 & i * a \text{ für } \text{typ}(a) \in \{\mathbb{N}\} & |
 \end{aligned}$$

2. Andererseits haben die in $P(aexpr_1, aexpr_2)$ verwendeten KD-Individuen nicht-numerische Typen, sind also vom Typ Zeichenkette (String) oder Boolescher Wert (Boolean). In diesem Fall ist $P \in \{=, <>\}$, und für $aexpr_i$ gilt lediglich die Syntaxregel $aexpr ::= a$.

Oben hatten wir bereits die Idee dargestellt, den Nutzer von der internen (komplizierten) ABox-Kodierung von Datatype Properties abzuschirmen, und diese mittels erweiterter Anfragekonzepte für den Nutzer transparent zu machen. Die Idee ist auch anwendbar für Constraint-Atome. Idealerweise sollten wir also nicht nur Attribute für A_i in Constraint-Atomen erlauben, sondern auch Datatype Properties (welche ja das OWL-Analogon zu Attributen darstellen). Dies ist auch tatsächlich möglich, jedoch muss aufgrund der momentanen RACERPRO-Kodierung die ABox dann um zusätzliche Hilfs-Assertionen erweitert werden, da die Füller von Datatype Properties nur implizit repräsentiert werden. Wir wollen auf die Problematik nicht näher eingehen.

5.3.3.3. Kopf-Projektionsoperatoren (KPOs) zur Ermittlung von KD-Individuen und TVs Zur Verdeutlichung der semantischen Probleme, die sich bei der Ermittlung von Werten aus der konkreten Domäne *durch Variablen einer ABox-Anfragesprache* ergeben, betrachten wir erneut die obige ABox:

$$\{ (michael, michael\_age) : age, (birgit, birgit\_age) : age, \\ (michael\_age) : =_{35}, (michael\_age, birgit\_age) : < \}.$$

Ein KD-Individuum wie *birgit_age* kann als Variable eines (in unserem Beispiel arithmetischen) *KD-Constraintsystems* gedeutet werden. Das Constraint-system ist erfüllbar, wenn es eine Belegung der Variablen bzw. KD-Individuen mit Werten aus der entspr. konkreten Domäne gibt, sodass alle Constraints bzw. Prädikate erfüllt sind. RACERPRO kann solch ein Constraintsystem auf Erfüllbarkeit testen, also die Existenz einer erfüllenden Variablenbelegung entscheiden, diese Lösungen jedoch nicht enumerieren bzw. aufzählen. Offensichtlich müssen die Domänen für eine Variable hier nicht endlich sein, oder sind sogar überabzählbar. Es ist daher nicht möglich, die „Lösungen“ von *birgit_age* aufzuzählen, da nur $35 < birgit\_age$ bekannt ist. Lediglich endliche Lösungsmengen könnten zum Enumerieren von Variablenbindungen durch Generator-ABFen verwendet werden. Dies ist im Fall von *michael_age* zwar der Fall, da die Lösungsmenge $\{35\}$ ist (es handelt sich ja um einen TV).

Ließen wir jedoch Atome der Art (x, y, age) zu und erwarteten, dass die Variable *y* gegen Elemente aus der Lösungsmenge der entspr. Variablen im Constraint-system gebunden würde, so führte dies zu semantischen Problemen. Selbst wenn die Lösungsmenge einelementig wäre, wie im Falle von *michael_age*, so könnte doch letztlich auch dieses Atom *NAF-negiert* werden, wodurch das Problem erneut entstünde.

Wir müssten beim Zulassen solcher Variablen daher entweder *Unsicherheit der Anfragesprache* oder *Unvollständigkeit* in Kauf nehmen, oder aber die Orthogonalität der Sprache durch Syntaxeinschränkungen zerstören. Um den ABox-spezifischen Anteil von *NRQL* klar, einfach und semantisch sauber zu halten, entscheiden wir uns gegen diese Möglichkeiten. Stattdessen werden wir die sog. *Kopf-Projektionsoperatoren (KPOs)* vorsehen, die eine saubere und pragmatische Lösung des Problems ermöglichen (indem es aus dem Rumpf der Anfrage entfernt wird). Atome der Art (x, y, age) sind daher schlichtweg syntaktisch inkorrekt; es handelt sich nicht um ein Rollen-Atom, da $\mathcal{N}_{\mathcal{R}} \cap \mathcal{N}_{\mathcal{A}} = \emptyset$.

Eine andere pragmatische Lösung des Variablenbindungsproblems wäre, die ADS für Variablen wie *x* und *y* in (x, y, age) zu verwenden (s. S. 204), und zwar bzgl. der Menge aller in einer ABox vorkommender TVs. Mit dieser eingeschränkten Semantik ist dann klar, dass die Variable *y* im Atom (x, y, age) lediglich gegen TVs gebunden werden kann; dementsprechend kann für $x = birgit$ keine *y*-Bindung generiert werden. Die NAF-negierten Atome haben dann ebenfalls eine unter dieser Betrachtungsweise konsistente Semantik, und die Anfragen sind wieder sicher. Diese pragmatische Lösung ist jedoch bereits durch die DSA und das *NRQL-Daten-Substrat (NRQL-Data-Substrate)* realisiert – wir müssen lediglich $(*x, *y, age)$ statt (x, y, age) verwenden. Da TVs nun jedoch auch mit nichthybridem *NRQL* ermittelt werden sollen, führen wir zusätzlich

5.3 Vier konkrete Substrat-Anfragesprachen

die sog. KPOs ein, insbesondere auch, weil das NRQL-Daten-Substrat (NRQL-Data-Substrate) zusätzlichen Speicher benötigt.

Bei den KPOs handelt es sich nun um *Funktionen*, die in einem letzten Projektionsschritt auf die ermittelten Ergebnistupel vor deren Auslieferung angewendet werden. Ein KPO wird im Kopf einer NRQL-Anfrage notiert:

Definition 97 (Kopf-Projektionsoperatoren (KPOs)) Sei $(antwort(\vec{h}) \leftarrow k)$ eine NRQL-Anfrage, und $\mathcal{A}$ die zu befragende ABox.

Sei $\vec{h} = (h_1, \dots, h_n)$ der NRQL-Kopf. Die h_i werden *Kopf-Projektionsoperatoren (KPOs)* genannt.

Eine KPO bzw. h_i ist eine partielle Funktion mit $h_i : O_{\mathcal{A}} \rightarrow (V \cup \mathcal{N}_{\mathcal{CD}} \cup \Delta_D^{\mathcal{I}})$. Hier ist $O_{\mathcal{A}} =_{\text{def}} Vars \cup V$ (s. Def. 68, S. 205), $\Delta_D^{\mathcal{I}}$ die Vereinigung aller Wertebereiche der unterstützten konkreten Domänen, $\mathcal{N}_{\mathcal{CD}}$ die Menge der KD-Objekte, und V die Menge der Individuen (bzw. Knoten, da für eine ABox stets $V = \text{inds}(\mathcal{A})$ gilt).

Sei $a_i \in O_{\mathcal{A}}$ (d.h., a_i ist ein Individuum oder eine Variable). Für h_i lassen wir dann folgende Ausdrücke zu:

- $h_i = a_i$,
- $h_i = A(a_i)$ für ein Attribut $A \in \mathcal{N}_{\mathcal{A}}$,
- $h_i = \text{told\_value}(A(a_i))$ für ein Attribut $A \in \mathcal{N}_{\mathcal{A}}$,
- $h_i = \text{datatype\_fillers}(R_{DTP}(a_i))$ für eine Rolle $R_{DTP} \in \mathcal{N}_{\mathcal{R}}$, die für ein Datatype Property angelegt wurde,
- $h_i = \text{annotation\_fillers}(R_{AP}(a_i))$ für eine Rolle $R_{AP} \in \mathcal{N}_{\mathcal{R}}$, die für ein Annotation Property angelegt wurde.

Für eine gegebene Bindung $a_i = j$ mit $j \in V$ ist $h_i(a_i)$ dann wie folgt definiert:

- falls $h_i = a_i$, so ist $h_i(a_i) =_{\text{def}} j$,
- falls $h_i = A(a_i)$, so ist $h_i(a_i) = A(j) =_{\text{def}} \{k \mid (j, k) : A \in \mathcal{A}\}$,
- falls $h_i = \text{told\_value}(A(a_i))$, so ist $h_i(a_i) =_{\text{def}} \text{told\_values}(A, j)$,
- falls $h_i = \text{datatype\_fillers}(R_{DTP}(a_i))$, so ist $h_i(a_i) =_{\text{def}} \text{datatype\_fillers}(R_{DTP}, j)$,
- falls $h_i = \text{annotation\_fillers}(R_{AP}(a_i))$, so ist $h_i(a_i) =_{\text{def}} \text{annotation\_fillers}(R_{AP}, j)$.

■

Die h_i -Funktionen liefern also u.U. Mengen. Die Hilfsfunktion told_values ist in NRQL wie folgt definiert; dabei sei angenommen, dass zuvor die auf S. 335 beschriebene TV-Maximierung durch Anreicherung der ABox vorgenommen wurde:

$$\text{told\_values}(A, j) =_{\text{def}} \{val \mid (j, k) : A \in \mathcal{A}, (k) :=_{val} \in \mathcal{A}\}.$$

Wir verzichten hier auf eine entspr. Definition von datatype_fillers und $\text{annotation\_fillers}$.

Das Ergebnis einer NRQL-Anfrage ergibt sich nun durch komponentenweise Anwendung der h_i -Funktionen auf die Ergebnistupel:

Definition 98 (Semantik einer NRQL-Anfrage) Sei $\vec{x} = (x_1, \dots, x_n)$, so ist die Semantik bzw. das Ergebnis der NRQL-Anfrage

$$(\text{antwort}((h_1(x_1), \dots, h_n(x_n))) \leftarrow k)^\varepsilon$$

gegeben durch

$$\{ (h_1(v_1), \dots, h_n(v_n)) \mid (v_1, \dots, v_n) \in (\text{antwort}(\vec{x}) \leftarrow k)^\varepsilon \}.$$

■

Angewendet auf unsere Beispiel-ABox ergibt jetzt die Anfrage

$$\text{antwort}((x, \text{age}(x), \text{told\_value}(\text{age}(x)))) \leftarrow (x, \text{person} \wedge \exists \text{age}. \top_{\mathcal{D}})$$

die Antwortmenge mit den beiden Tupeln

$$\{(\text{michael}, \{\text{age\_of\_michael}\}, \{35\}), (\text{michael}, \{\text{age\_of\_birgit}\}, \{\})\}.$$

In der Implementation werden statt $\{\}$ entspr. Symbole wie `:no-told-value` zurückgegeben (und statt Mengen werden Listen verwendet).

5.3.3.3.4. Hybride NRQL-Anfragen Unter Verwendung von KPOs können wir mit „reinen“ ABox-Anfragen, wie am Beispiel von

$$\text{antwort}((x, \text{age}(x), \text{told\_value}(\text{age}(x)))) \leftarrow (x, \text{person} \wedge \exists \text{age}. \top_{\mathcal{D}})$$

gezeigt, TVs und KD-Individuen ermitteln. Unter Einbeziehung der DSA-Atome können wir jedoch auch eine *hybride NRQL-Anfrage* für das Beispiel wie folgt formulieren:

$$\begin{aligned} \text{antwort}((x, *age\_of\_x, *told\_value\_of\_age\_of\_x)) \leftarrow \\ (x, \text{person} \wedge \exists \text{age}. \top_{\mathcal{D}}) \wedge \\ (*x, *age\_of\_x, \text{has\_age} \wedge : \text{abox-attribute-relationship}) \wedge \\ (*age\_of\_x, *told\_value\_of\_age\_of\_x, : \text{abox-told-value-relationship}) \end{aligned}$$

Die Anfrage liefert das Ergebnis

$$\{(\text{michael}, *age\_of\_michael, 35)\}.$$

Hier ist `*age_of_michael` der zu `age_of_michael` korrespondierende Knoten im NRQL-Daten-Substrat (NRQL-Data-Substrate). Aufgrund der vorausgeschickten Erklärungen sollte die Semantik der Anfrage klar sein.

5.3 Vier konkrete Substrat-Anfragesprachen

Wir können noch weitere DSA-Atome zur Verschärfung der Anfrage hinzufügen, z.B.

$$(*told\_value\_of\_age\_of\_x, (: predicate (< 40))).$$

Offensichtlich wäre diese Bedingung auch mittels

$$(x, person \wedge \exists age. <_{40})$$

formulierbar. Die DSA kann solche Prädikate für TVs aufgrund der schwächeren Semantik jedoch wesentlich effizienter (durch eine Art *Model Checking*) auswerten als RACERPRO; zudem stellt die DSA ja viele Prädikate bereit, die in RACERPRO nicht zuletzt aus Gründen der Entscheidbarkeit fehlen. Wir hatten diese Punkte bereits erschöpfend diskutiert.

5.3.3.4. Konkrete Syntax von NRQL und Beispiele

Die konkrete Syntax von NRQL ergibt sich aus der konkreten Syntax für die BAA, der konkreten Syntax für die DSA, der Syntax von RACERPRO für Konzepte und Prädikate, sowie der Syntax für Constraint-Atome. Zusätzlich kommen noch die Kopf-Projektionsoperatoren hinzu. Die konkrete Syntax für Anfragekonzepte und Anfragerollen ist ebenfalls diskutiert worden.

5.3.3.4.1. Konkrete Syntax der Objekte NRQL kennt 4 Arten von Objekten, deren Syntax wurde in Kap. 5.2.2.7 beschrieben. In einem NRQL-Hybrid-Substrat (NRQL-Hybrid-Substrate) ist S_1 stets das NRQL-ABox-Substrat (NRQL-ABox-Substrate), und S_2 das NRQL-Daten-Substrat (NRQL-Data-Substrate). Dementsprechend werden

- ABox-Variablen mit einem ?-Präfix, und
- DSA-Variablen mit *?-Präfix versehen.

Für die Individuen bzw. Knoten gilt entsprechendes.

5.3.3.4.2. Konkrete Syntax der Atome Hier muss lediglich die konkrete Syntax der Constraint-Atom nachgereicht werden. Das Atom

$$(x, y, \exists(R_1 \circ \dots \circ R_n \circ A_1, S_1 \circ \dots \circ S_m \circ A_2).P)$$

(mit $x, y \in Vars$) wird als

$$(?x ?y (constraint (R1 \dots Rn A1) (S1 \dots Sm A2) P))$$

notiert.

Ein Beispiel für ein einfaches Constraint-Atom in konkreter Syntax ist

$$(?x ?x (constraint (has-father age) (has-mother age) =)).$$

Die Extension dieses Atomes enthält also Paare von Individuen (i, i) , deren Eltern das gleiche Alter haben.

Ein Constraint-Atom, das einen komplexen Prädikatsausdruck (s. Def. 96) verwendet, ist durch

$$(\text{?x ?y (constraint (has-father age) (has-mother age) (<= (+ has-age-1 8) age-2)))$$

gegeben. Nun ist ein Paar (i, j) in der Extension dieses Atomes, wenn der Vater von i mindestens 8 Jahre älter als die Mutter von j ist. Hier ist $P = \leq (+(\text{age}_1, 8), \text{age}_2)$ bzw. $P = \text{age}_1 + 8 \leq \text{age}_2$. In dieser Schreibweise sind age_1 und age_2 die freien Variablen; sie werden anhand der Attribute gebildet. Die Suffixe -1, -2 werden zur Disambiguierung angehängt, wenn das gleiche Attribut verwendet wird. So referenziert age-1 das Alter des Vaters von ?x bzw. i , aber age-2 das Alter der Mutter von ?y bzw. j .

5.3.3.4.3. Konkrete Syntax der Kopf-Projektionsoperatoren Die im Kopf einer NRQL-Anfrage verwendbaren KPOs h_i werden in konkreter Syntax wie folgt notiert:

- für $h_i = a_i$: offensichtlich (s. Kap. 5.2.2.7),
- für $h_i = A(a_i)$ als $(A \ a_i)$, wobei a_i in konkrete Syntax gebracht wird;
- für $h_i = \text{told\_value}(A(a_i))$ als $(\text{told-value } A(a_i))$, wobei $A(a_i)$ in konkrete Syntax gebracht wird;
- $h_i = \text{datatype\_fillers}(R_{DTP}(a_i))$ als $(\text{datatype-fillers (R-DTP } a_i))$, wobei a_i in konkrete Syntax gebracht wird,
- $h_i = \text{annotation\_fillers}(R_{AP}(a_i))$ als $(\text{annotation-fillers (R-AP } a_i))$, wobei a_i in konkrete Syntax gebracht wird.

5.3.3.4.4. Beispiel-Anfragen in konkreter Syntax Zur Illustration der konkreten Syntax und des bereits für die BAA beschriebenen Basis-APIs der Anfragebeantwortungsmaschine betrachten wir eine Reihe von NRQL-Anfragen, die wir an die Beispiel-Wissensbasis aus Kap. 2.2.8.1 stellen:

```
> (retrieve (?x)
    (?x (and person (> age 30))))

(((?X MICHAEL))
 ((?X BIRGIT)))

> (retrieve (?x)
    (?x person))
```

5.3 Vier konkrete Substrat-Anfragesprachen

```
((?X MICHAEL)) (?X MARGRIT))
(?X MARA))    (?X ALINA))
(?X BIRGIT))  (?X RUDOLF))

> (retrieve (?x ?y)
    (and (?x woman)
        (?x ?y has-child) (?y man)))

(((?X MARGRIT) (?Y MICHAEL)))

> (retrieve (?x)
    (and (?x mother)
        (neg (project-to (?x)
            (and (?x ?y has-child) (?y man))))))

(((?X BIRGIT)))

> (retrieve (?x (age ?x)
    (told-value (age ?x))) (?x (an age)))

(((?X MICHAEL)
    ((AGE ?X) (MICHAEL-AGE)) (:TOLD-VALUE (AGE ?X) (35.0)))

((?X MARGRIT)
    ((AGE ?X) :NO-KNOWN-CD-OBJECTS)
    (:TOLD-VALUE (AGE ?X) :NO-KNOWN-CD-OBJECTS))

((?X MARA)
    ((AGE ?X) :NO-KNOWN-CD-OBJECTS)
    (:TOLD-VALUE (AGE ?X) :NO-KNOWN-CD-OBJECTS))

((?X ALINA)
    ((AGE ?X) :NO-KNOWN-CD-OBJECTS)
    (:TOLD-VALUE (AGE ?X) :NO-KNOWN-CD-OBJECTS))

((?X BIRGIT)
    ((AGE ?X) (BIRGIT-AGE))
    (:TOLD-VALUE (AGE ?X) (:NO-TOLD-VALUE)))

((?X RUDOLF)
    ((AGE ?X) :NO-KNOWN-CD-OBJECTS)
    (:TOLD-VALUE (AGE ?X) :NO-KNOWN-CD-OBJECTS)))
```

```
> (retrieve (?x ?y (told-value (age ?x)) (told-value (age ?y)))  
    (?x ?y (constraint age age <)))
```

```
((?X MICHAEL) (?Y BIRGIT)  
  (:TOLD-VALUE (AGE ?X)) (35.0))  
  (:TOLD-VALUE (AGE ?Y)) (:NO-TOLD-VALUE))))
```

Als Beispiel für eine hybride NRQL-Anfrage fügen wir einige DSA-Atome hinzu – welche TVs (vom Typ Zeichenkette) enthalten die Zeichenkette "Michael"? Nach Aktivierung des NRQL-Daten-Substrates liefert dann

```
(retrieve (?x (age ?x) (told-value (age ?x)) ?*x-name-value)  
  (and (?x (and person (> age 30)))  
    (?*x ?*x-name name)  
    (?*x-name ?*x-name-value :abox-told-value-relationship)  
    (?*x-name-value (:predicate (search "Michael")))))
```

das Ergebnis

```
((?x michael) ((age ?x) (michael-age)) (:told-value (age ?x)) (35))  
  (?*x-name-value "Michael Wessel")))
```

Wie diskutiert, ist es *ausschließlich* mit der DSA möglich, Variablen wie ?*x-name-value an TVs zu binden; für den Ausdruck (:predicate (search "Michael")) gibt es zudem keine Entsprechung in der konkreten Domäne.

5.3.3.4.5. LUBM-Anfragen mit NRQL Die in Kap. 3.2.4.1 vorgestellte LUBM-Anfrage

```
(type GraduateStudent ?X)  
(type University ?Y)  
(type Department ?Z)  
(memberOf ?X ?Z)  
(subOrganizationOf ?Z ?Y)  
(undergraduateDegreeFrom ?X ?Y)
```

würde in NRQL nun wie folgt aussehen:

```
(retrieve (?x ?y ?z)  
  (and (?x GraduateStudent)  
    (?y University)
```

5.3 Vier konkrete Substrat-Anfragesprachen

```
(?z Department)
(?x ?z memberOf)
(?z ?y subOrganizationOf)
(?x ?y undergraduateDegreeFrom)))
```

(der LUBM legt den Kopf nicht fest; wir können hier $(?x ?y ?z)$, oder auch nur $(?x)$ aufnehmen).

5.3.3.5. Entwurf der operationalen Aspekte von NRQL

Wir entwerfen nun die unvollständigen und zweiphasigen Modi. Schließlich gehen wir auf die Details der für NRQL erforderlichen Anpassungen und Erweiterungen der Anfragebearbeitungskette (s. Abb. 5.14) ein, wodurch deutlich wird, wie NRQL im bestehenden Rahmenwerk „eingehängt“ wird.

5.3.3.5.1. Entwurf der unvollständigen Modi Wir hatten erläutert, dass beim initialen Spiegeln der ABox-Assertionen in die NRQL-lokalen Cache- und Indexstrukturen nun in Abhängigkeit vom NRQL-Modus noch weitere, durch „einfache“ Inferenz ermittelbare zusätzliche Einträge in diese zur Maximierung der „Told Information“ gemacht werden können. Diese Ableitungs- bzw. Anreicherungsprozesse können durch Fixpunktiterationen beschrieben werden, die die ABox um weitere Assertionen ergänzen. Diese zusätzlichen Assertionen existieren jedoch nur in den lokalen Cache- und Indexstrukturen; die originale ABox wird hierfür nicht verändert.

NRQL bietet prinzipiell 4 Modi, die Mode 0, 1, 2, und 3 genannt werden. Der Grad der Vollständigkeit dieser Modi nimmt in dieser Reihenfolge zu. Modus 3 ist der vollständige Modus. Die unvollständigen Modi können zum einen durch die formale Definition der verwendeten Anreicherungsprozesse definiert werden. Zum anderen können sie durch Charakterisierung der ABoxen bzw. WBs, für die sie vollständig sind, definiert werden.

Für alle unvollständigen Modi gilt, dass sie lediglich für Konzept-Atome (x, C) mit *atomarem* Anfragekonzept C , also $C \in \mathcal{N}_C$, eine u.U. annehmbare Vollständigkeit erreichen; dies liegt daran, dass die Anreicherungsprozesse letztlich nicht „beliebige“ Konzept-Assertionen $i : C$ zur ABox hinzufügen können, sondern lediglich solche mit *atomarem* C .

Modus 0: In diesem Modus wird ausschließlich die durch ABox-Assertionen gegebene „Told Information“ ohne weitere Anreicherung zur Anfragebeantwortung genutzt.

Gilt beispielsweise $betty : mother \in \mathcal{A}$, so gilt in Modus 0 $(x, mother)^{\mathcal{E}} = \{(betty)\}$, aber auch $(x, woman)^{\mathcal{E}} = \emptyset$, selbst wenn $mother \sqsubseteq woman$ gilt.

Um eine „minimale“ Vollständigkeit in Modus 0 zu erhalten, werden konjunktive Konzept-Assertionen der Art $i : C_1 \sqcap \dots \sqcap C_n$ aufgebrochen und als zusätzliche Konzept-Assertionen $i : C_i$ für $i \in 1 \dots n$ explizit gemacht. Gilt also $betty : mother \sqcap employee \in \mathcal{A}$, so wird $\{betty : mother, betty : employee\} \subseteq \mathcal{A}$ in den Cache- und Indexstrukturen für $\mathcal{A}$ sichergestellt ($\mathcal{A}$ bleibt unverändert).

Den Anreicherungsprozess können wir durch folgende Fixpunkt-Iteration beschreiben (s. S. 335 für die umgebende Schleife):

- $\mathcal{A}_0 =_{def} \mathcal{A}$
- $\mathcal{A}_{n+1} =_{def} \mathcal{A}_n \cup \{i : C_j \mid i : C_1 \sqcap \dots \sqcap C_m \in \mathcal{A}_n, j \in 1 \dots m\}$.

Wozu ist dieser Modus gut? Teilweise werden ABoxen nur zur Repräsentation relationaler Strukturen oder annotierter Graphen verwendet (bzw. für Daten, ähnlich RDF). Die ABoxen sind dann isomorph zu ihren minimalen (eindeutigen) Herbrandmodellen, bzw. der relationalen Struktur selbst. Modus 0 ist daher vollständig für sog. *Daten-ABoxen*:

Definition 99 (Daten-ABox) Für eine Daten-ABox $\mathcal{A}$ gilt:

- $\mathcal{T}_{\mathcal{A}} = \emptyset$.
- Alle Konzept-Assertionen $i : C$ haben die Form $C = C_1 \sqcap \dots \sqcap C_n$ mit $C_j \in \mathcal{N}_{\mathcal{C}}$, oder $C_j = \neg C_k$ und $C_k \in \mathcal{N}_{\mathcal{C}}$, für alle $j \in 1 \dots n$.
- Für alle Rollen-Assertionen $(i, j) : R$ gilt $R \notin \mathcal{N}_{\mathcal{F}}$.
- Andere Assertionen sind nicht erlaubt. ■

Im Gegensatz zu in der Ausdrucksmächtigkeit vergleichbaren Klassen von Substraten werden hier die Eigenschaften transitiver, symmetrischer und zueinander inverser Rollen berücksichtigt²², und auch negierte Präpositionen sind möglich, was ein echter Mehrwert ist.

Modus 1: Zusätzlich zu Modus 0 werden die durch die assoziierte TBox implizierten Subsumptionsbeziehungen zwischen atomaren Konzepten ausgenutzt, die ja durch die *Taxonomie der TBox* explizit gemacht werden können. Für Modus 1 muss also zuvor die Taxonomie berechnet werden, sodass für gegebenes atomares Konzept C $\text{synonyms}(C)$ und $\text{subsumers}(C)$ zur Verfügung steht:

- $\mathcal{A}_0 =_{def} \mathcal{A}$
- $\mathcal{A}_{n+1} =_{def} \mathcal{A}_n \cup \{i : C_j \mid i : C_1 \sqcap \dots \sqcap C_m \in \mathcal{A}_n, j \in 1 \dots m\} \cup \{i : C_j \mid i : C \in \mathcal{A}_n, C \in \mathcal{N}_{\mathcal{C}}, C_j \in \text{synonyms}(C) \cup \text{subsumers}(C)\}$.

²²Wir gehen davon aus, dass die Axiome zur Deklaration der Rolleneigenschaften in einer RBox und nicht in der TBox (wie hatten ja $\mathcal{T}_{\mathcal{A}} = \emptyset$ gefordert) enthalten sind, s. Kap. 2.2.3.3.

5.3 Vier konkrete Substrat-Anfragesprachen

Gilt nun wieder $betty : mother \in \mathcal{A}$, und ist $female\_parent$ ein Synonym für $mother$, und die Menge der Oberkonzepte von $mother$ in der Taxonomie durch $\{parent, woman, person, \top\}$ gegeben, dann wird durch die Anreicherung $\{betty : female\_parent, betty : parent, betty : woman, betty : person, betty : \top\}$ hinzugefügt.

Dieser Modus ist vollständig für Daten-ABoxen mit assoziierten TBoxen, die lediglich $C \sqsubseteq D$ -Axiome mit $C, D \in \mathcal{N}_C$ enthalten. Eine solche TBox ist ausreichend, um thesaurusbasiertes IR zu ermöglichen.

Im Gegensatz zu Modus 0 wird dieser Modus nun $(x, woman)^\mathcal{E} = \{betty\}$ liefern. Obwohl (wie oben erläutert) immer noch $(x, woman \sqcap person)^\mathcal{E} = \emptyset$ ist, gilt jetzt $((x, woman) \wedge (x, person))^\mathcal{E} = \{(betty)\}$.

Modus 2: Stellt eine Erweiterung von Modus 1 dar. Hier wird die Beschränkung auf *atomare Konzepte* ($C \in \mathcal{N}_C$) bei der Berechnung der erweiterten ABox fallen gelassen:

- $\mathcal{A}_0 =_{def} \mathcal{A}$
- $\mathcal{A}_{n+1} =_{def} \mathcal{A}_n \cup \{i : C_j \mid i : C_1 \sqcap \dots \sqcap C_m \in \mathcal{A}_n, j \in 1 \dots m\} \cup \{i : C_j \mid i : C \in \mathcal{A}_n, C_j \in \text{synonyms}(C) \cup \text{subsumers}(C)\}.$

Gilt nun $betty : woman \sqcap mother \in \mathcal{A}$, so werden die Synonyme und subsumierenden Oberkonzepte von $woman \sqcap mother$ berechnet und zu $\mathcal{A}$ hinzugefügt, u.a. also die Assertionen $betty : person$, da sowohl $woman \sqsubseteq person$ als auch $mother \sqsubseteq person$ gilt.

Die für diesen Modus erforderliche *Klassifikation beliebiger Konzepte*, wie $woman \sqcap mother$, ist jedoch recht aufwendig. Die Vorwegberechnung der Taxonomie der TBox ist hier natürlich nicht mehr ausreichend.

Modus 3: Der Standardmodus von NRQL. Dieser Modus ist vollständig (bzgl. der BAA- bzw. NRQL-Semantik).

Zu Mode 2 vergleichbare Verfahren werden auch im „Instance Store“ angewendet [Horrocks u. a. 2004b]. Allerdings ist der Instance Store auf ABoxen ohne Rollen-Assertionen beschränkt (da Vollständigkeit stets garantiert werden soll), während NRQL auch transitive und inverse Rollen sowie Rollenhierarchien berücksichtigt (und Vollständigkeit ist nur für die soeben beschriebenen Arten von Wissensbasen garantiert).

5.3.3.5.2. Entwurf der zweiphasigen Modi Die zweiphasigen, tupelbasierten Modi werden nun so gestaltet, dass die billigen Tupel in Phase 1 zunächst durch Modus 0, 1 oder 2 bereitgestellt werden. Für Phase 2 wird stets Modus 3 verwendet. Je nachdem, ob für Phase 1 nun Modus 0, Modus 1 oder Modus 2 verwendet wird, sprechen wir vom zweiphasigen Modus 4, 5 bzw. 6. All diese Modi arbeiten inkrementell.

Sind alle Tupel aus Phase 1 ausgeliefert, so kann der Nutzer durch Auslieferung eines Benachrichtigungs-Symbols darüber informiert werden, dass der Phasenübergang stattfindet (s. Kap. 5.3.3.1.3, S. 340).

Im Gegensatz zu den Modi 4, 5, und 6, ist Modus 7 ein *mengenbasierter* Modus, der Modus 1 für Phase 1 verwendet (s. Kap. 5.3.3.1.3, S. 340).

5.3.3.5.3. Illustration der zweiphasigen Modi Zur Illustration des zweiphasigen Modus 4 betrachten wir folgende Beispiel-WB:

- ABox:
 $\{doris : spouse, betty : spouse, adam : man, eve : woman, (eve, adam) : married\_to\},$
- TBox:
 $\{man \sqsubseteq person, woman \sqsubseteq person, spouse \sqsubseteq woman \sqcap \exists married\_to.man\}.$

Schalten wir NRQL in den Modus 4, so ergibt sich folgendes Verhalten:

```
> (set-nrql-mode 4)

:OKAY-MODE-4

> (retrieve (?x) (?x spouse))

(:QUERY-1 :RUNNING)

> (get-next-tuple :query-1)

((?X DORIS))

> (get-next-tuple :query-1)

((?X BETTY))

> (get-next-tuple :query-1)

:WARNING-EXPENSIVE-PHASE-TWO-STARTS

> (get-next-tuple :query-1)

((?X EVE))

> (get-next-tuple :query-1)

:EXHAUSTED
```

5.3 Vier konkrete Substrat-Anfragesprachen

```
> (get-answer :query-1)
```

```
((?X DORIS)) (?X BETTY)) (?X EVE)))
```

Als Notifikationssymbol wurde `:WARNING-EXPENSIVE-PHASE-TWO-STARTS` vor dem Phasenübergang geliefert. Lediglich für `((?X EVE))` wurde also ABox-Inferenz erforderlich.

5.3.3.5.4. Anfragebearbeitungskette in NRQL Wir hatten die generische Anfragebearbeitungskette, s. Abb. 5.14, beim Entwurf der Basis-Maschine in Kap. 5.2.2 beschrieben. Bevor eine Anfrage ausgeführt wird (Schritt 7), wird die Anfrage *vorbereitet*, also die Schritte 1 bis 6 durchlaufen. Aufgrund des potentiell asynchronen APIs bzw. der Lebenszyklus-Verwaltung müssen Vorbereitung und Ausführung nicht unmittelbar aufeinander folgen, s. Kap 5.2.2.6.

Wir hatten dargestellt, dass der Anfragen-Verwalter (Query-Manager) für die Lokalisierung des zu befragenden Substrates und für die Erzeugung des Parsers für dieses Substrat zuständig ist. Der Anfragen-Verwalter (Query-Manager) veranlasst jedoch nicht die *Erzeugung* eines entspr. Substrates (sollte es nicht existieren). Aus der Perspektive eines RACERPRO- bzw. NRQL-Nutzers sollte die Erzeugung und Initialisierung eines Substrates jedoch transparent und automatisch geschehen, da aus seiner Perspektive lediglich ABoxen (und nicht *Substrate*) befragt werden. Das automatische Erzeugen einer Instanz der Klasse `NRQL-ABox-Substrat` (`NRQL-ABox-Substrate`) bzw. der Klasse `NRQL-Hybrid-Substrat` (`NRQL-Hybrid-Substrate`) sollte daher automatisch und somit für den Nutzer transparent geschehen. Wir leiten daher einen spezialisierten `NRQL-Anfrage-Verwalter` (`NRQL-Query-Manager`) von der Klasse `Anfragen-Verwalter` (`Query-Manager`) ab und überschreiben die `prepare-query`-Methode, sodass sich folgendes Verhalten ergibt:

1. Die Methode `prepare-query` ermittelt zunächst die ABox, für die die Anfrage bestimmt ist; die ABox wird entweder als Argument explizit spezifiziert, oder aber die *aktuelle* RACERPRO-ABox (`current-abox`) wird verwendet. Die so ermittelte ABox wird die *Anfrage-ABox* genannt.

Jede Anfrage kann durch optionale Argumente für `prepare-query` den *Operationsmodus* der Maschine bestimmen (Grad der Vollständigkeit, tupel- oder mengenbasiert, zweiphasig, etc.). Andernfalls wird der *aktuelle Operationsmodus* verwendet.

2. Für die Anfrage-ABox wird mittels `find-substrate` ein passendes Substrat vom Substrat-Verwalter (`Substrate-Manager`) angefordert. Das so bestimmte Substrat wird das *Anfrage-Substrat* genannt. Der *Typ des geforderten Substrates* ergibt sich anhand des von der Anfrage geforderten Operationsmodus.

Wird kein entspr. Substrat gefunden, so wird ein neues Substrat (mit den geforderten Eigenschaften) erzeugt. Das neu erzeugte Substrat muss für die Nutzung initialisiert werden; die Cache- und Indexstrukturen müssen angelegt werden, die Anreicherungsprozesse für die unvollständigen Modi müssen ablaufen, etc. Diese Aufbereitung eines Substrates ist also *modusspezifisch*. Dies geschieht jedoch erst, wenn die erste Anfrage dieses Substrat befragt; sollte sich die ABox nämlich bis zum *Ausführungszeitpunkt der Anfrage geändert haben*, so müssten diese Strukturen wieder invalidiert werden (wie beschrieben sind das NRQL- und RACERPRO-API im inkrementellen Modus zwischenzeitlich verfügbar, sodass in dieser Zeit Zustandsänderungen möglich sind). Die Cache-Einträge sind natürlich vom Zustand der Anfrage-ABox und ihrer assoziierten TBox abhängig.

Zur Lösung dieses *Cache-Synchronisationsproblems* wäre es ideal, wenn RACERPRO ein entspr. *Publish-Subscribe*-Protokoll zur Notifikation bei erfolgten ABox- bzw. TBox-Änderungen bieten würde, sodass die Strukturen inkrementell miteinander konsistent gehalten werden könnten. Dieses Protokoll ist (mit den hier geforderten Eigenschaften) leider nicht vorhanden. Wir können daher nur eine pragmatische Lösung des Problems anbieten. RACERPRO bietet eine Funktion, mit der Zustandsänderungen der Anfrage-ABox und -TBox entdeckt werden können – jede ABox- bzw. TBox-Änderung führt zu einer Veränderung des Rückgabewertes der API-Funktion *get-abox-version* bzw. *get-tbox-version*. Durch Speicherung und Vergleich dieses *WB-Fingerabdrucks* können also Veränderungen der WB entdeckt werden. In diesem Fall sind entsprechende Maßnahmen zu ergreifen (Invalidierung der Caches etc.).

Ein NRQL-Substrat (NRQL-Substrate) ist eine schwergewichtige Repräsentation. Daher wird pro ABox nur ein Substrat angelegt. Es kann also pro ABox höchstens ein NRQL-ABox-Substrat (NRQL-ABox-Substrate) bzw. ein NRQL-Hybrid-Substrat (NRQL-Hybrid-Substrate) geben. Anfragen, die die gleiche Anfrage-ABox verwenden, müssen sich daher ein Substrat „teilen“. Dies hat Konsequenzen:

- Ein *positiver Seiteneffekt* der gemeinsamen Verwendung ist, dass die Anfragen wechselseitig von den gemeinsam genutzten und erweiterten Caches profitieren können (eine entsprechende Nebenläufigkeitskontrolle vorausgesetzt).
- Ein *negativer Seiteneffekt* ergibt sich daraus, dass die Substrat-Aufbereitung und somit der Inhalt der Caches modusspezifisch ist. Eine Anfrage benötigt ein Substrat, welches den von der Anfrage geforderten Zustand hat.

Es ist daher möglich, dass ein bereits für einen anderen Modus vorbereitetes Substrat reinitialisiert werden muss, bevor es von einer

5.3 Vier konkrete Substrat-Anfragesprachen

Anfrage, die einen anderen Modus fordert, verwendet werden kann. Dies schränkt auch die Nebenläufigkeit ein, da Anfragen, die miteinander nicht vereinbare Zustände eines Substrates fordern, das Substrat nicht gleichzeitig verwenden können. Die NRQL-Maschine sorgt für eine entsprechende Serialisierung.

Prinzipiell könnte das Verhalten von *prepare-query* natürlich auch so geändert werden, dass in diesem Fall ein zusätzliches Substrat erzeugt wird. Aus Speicherplatzgründen haben wir uns dagegen entschieden; durch ein optionales Argument für *prepare-query* kann diese Option leicht nachgepflegt werden.

3. Der durch das Anfrage-Substrat gegebene Kontext wird verwendet, um die Anfrage vorzubereiten (Übersetzen, Optimieren, Compilieren), also Schritte 2 bis 5 in der Anfragebearbeitungskette zu durchlaufen, wie in Kap. 5.2.2 beschrieben.

Der von der Anfrage geforderte *Operationsmodus der NRQL-Maschine* bildet zusammen mit dem aktuellen WB-Fingerabdruck (*get-tbox-version*, *get-abox-version*) den sog. *Zustandsvektor*. Dieser Vektor wird in der Anfrage-Instanz als Wert des Attributes *state-vector* (s. Klasse *Anfrage* (*Query*)) gesichert.

Wird nun eine Anfrage aus der Liste der vorbereiteten Anfragen mittels *execute-query* zur Ausführung gebracht, so passiert folgendes:

1. Ein *COMMON LISP-Bearbeitungsprozess* wird aus einem Prozess-Pool akquiriert.²³ Wie erläutert, handelt es sich hierbei um einen Kontrollfaden (Thread). Die erste Aktion dieses *COMMON LISP-Prozesses* ist nun, das Anfrage-Substrat für die Anfragebearbeitung *vorbereiten*. Dazu wird der Zustandsvektor der Anfrage (*state-vector*) inspiziert, da durch ihn die Repräsentationsanforderungen an das Substrat (Modus etc.) formuliert sind.

Hat sich die assoziierte TBox der Anfrage-ABox nun seit der Anfrage-Vorbereitung verändert, was durch Vergleich des aktuellen TBox-Fingerabdruckes mit dem im Zustandsvektor der Anfrage gespeicherten TBox-Fingerabdruck festgestellt werden kann, so kann die Anfrage nicht unmittelbar gestartet werden, da sich u.U. die Eigenschaften bzw. Definitionen der Rollen geändert haben. NRQL gibt dann eine Warnung aus und bereitet die Anfrage erneut vor.

Bzgl. des Zustands des Anfrage-Substrates sind nun zwei Fälle möglich:

- a) Kann das Substrat die durch die Anfrage geforderten Repräsentati-

²³Der Pool kann dynamisch vergrößert werden, sollte die Anzahl der vorgesehenen Prozesse nicht ausreichen.

ansforderungen erfüllen, so kann die Anfrage direkt zur Ausführung auf dem Substrat gebracht werden. Dies ist der häufigste Fall (i.d.R. werden eine ganze Reihe von Anfragen im gleichen NRQL-Modus hintereinander abgesetzt).

- b) Andernfalls muss das Substrat reinitialisiert werden, sodass es die durch die Anfrage geforderten Repräsentationsanforderungen erfüllt. Dies kann jedoch nicht geschehen, bevor nicht alle anderen Anfragen, die dieses Substrat verwenden, terminiert haben.

Wir hatten erläutert, dass eine Anfrage u.U. nicht von selbst terminiert, nämlich wenn sie im *faulen (inkrementellen) Modus* (s. S. 305) abläuft. In diesem Falle wird eine Warnung ausgegeben, die den Nutzer darüber informiert, dass seine Anfrage nicht gestartet werden kann, solange die im faulen Modus ablaufende Anfrage nicht terminiert hat; insbesondere ist hier auch Verklemmungspotential gegeben. NRQL stellt sicher, dass keine Verklemmungen (Dead Locks) auftreten; eine entspr. Deadlock-Erkennung wird vorgesehen, sodass das API stets verfügbar bleibt.

2. Die eigentliche Anfragebeantwortung wird nun durch Aufruf der ABF der Anfrage-Instanz vom Prozess angestoßen.

Die Zugriffe auf RACERPRO müssen (bei mehreren konkurrent ablaufenden Prozessen bzw. Anfragen) serialisiert werden. Hierzu werden Sperren (Locks) verwendet. Bereits die von der Klasse RACERPRO-BAA-Substrat (RACERPRO-BAA-Substrate) ererbten Methoden zur Inanspruchnahme der RACERPRO-Dienste (wie `dl-prover-get-concept-instances`) sind derartig synchronisiert.

Die Berechnung von Ergebnistupeln kann dann mengenbasiert oder tupelbasiert erfolgen. Mengenbasierte Tupelberechnung wird auf tupelbasierte Einzelberechnungen mittels wiederholter Aufrufe von `get-next-tuple` realisiert, wie bereits in der BAA. Der ABF-Prozess kann *faul* (lazy) oder *fleißig* (eager) arbeiten, wie erläutert. Im mengenbasierten Modus wird stets eine fleißige Tupelberechnung verwendet.

Für den Fall, dass eine *zweiphasige Tupelberechnung* gefordert wurde, wird die Anfrage einfach ein zweites Mal mit geändertem Bearbeitungsmodus der Maschine ausgeführt (der modusspezifische Zustand des Substrates muss hierfür nicht geändert werden).

Die KPOs sind im bestehenden Rahmenwerk ebenfalls einfach zu realisieren. Im Rahmenwerk wird, bevor ein neues Tupel auf dem Ergebnisstapel des Anfrage-Objektes bzw. ASG-Objektes abgelegt wird, die vorgesehene Notifikationsmethode `register-tupel` gerufen. Wir können diese überschreiben, sodass die Methode dann die Projektionsoperatoren kom-

5.3 Vier konkrete Substrat-Anfragesprachen

ponentenweise auf das berechnete Ergebnistupel anwendet, bevor das Tupel auf dem Stapel abgelegt wird. Mehrfachregistrierung von Ergebnistupeln (insbesondere in den zweiphasigen Modi) wird von `register-tupel` ebenfalls unterdrückt.

Desweiteren gilt für NRQL-Anfragen der normale Lebenszyklus, s. Kap. 5.2.2.6.

Prinzipiell kann sich die Anfrage-WB nicht nur in der Zwischenzeit zwischen Vorbereitung und Ausführung der Anfrage ändern (während die Anfrage-Instanz also im Zustand *vorbereitet* ist), sondern auch, wenn die Anfrage-Instanz bereits im Zustand

1. *beendet bzw. terminiert*, oder im Zustand
2. *aktiv* ist

(s. Abb. 5.17, S. 301). Wir verfahren dann wie folgt:

1. Das Anfrageergebnis einer terminierten Anfrage ist als Wert des Attributes `result` (Klasse `Anfrage (Query)`) in der Anfrage-Instanz gespeichert (sofern die Bearbeitung nicht mittels `abort-query` abgebrochen wurde). Dieses gespeicherte Ergebnis kann mittels `get-answer` beliebig oft ausgelesen bzw. angefordert werden.

Sollte sich der Fingerabdruck der Anfrage-WB seit Berechnung des Ergebnisses geändert haben, so wird ein Notifikationssymbol `warning-kb-has-changed` als „zusätzliches Tupel“ der mittels `get-answer` angeforderten Ergebnismenge ausgeliefert, um den Nutzer zu informieren, dass diese u.U. nicht mehr aktuell ist. In diesem Fall steht es ihm frei, die Anfrage mittels `reexecute-query` erneut zur Ausführung zu bringen.

2. Ändert sich der Fingerabdruck der Anfrage-WB während die Anfrage noch im Zustand *aktiv* ist, so wird ebenfalls das Symbol `warning-kb-has-changed` als Ergebnistupel von `get-next-tuple` geliefert. Wie geschildert, wird `get-next-tuple` sowohl im mengenbasierten als natürlich auch im tupelbasierten Modus verwendet. Somit wird das Benachrichtigungssymbol stets den Nutzer erreichen. Der Nutzer hat dann die Option, die Anfragebearbeitung abubrechen und erneut zu starten, etc.

5.3.3.6. Implementation von NRQL

Die Implementation von NRQL ist mit mehr als 40.000 Programmzeilen (Lines of Code, LOC) recht umfangreich (nach einigen LOC-Klassifikationen handelt es sich nicht mehr um ein „kleines“, sondern bereits ein „mittelgroßes Projekt“).

Die Darstellung beschränkt sich (in jeweils wenigen Worten) auf folgende Punkte:

1. Implementation der Cache- und Indexstrukturen;
2. Implementation des Parsers, der ASG-Klassen, und der ABFen;
3. Implementation verfeinerter Bewertungsmethoden für den Optimierer;
4. Vermeidung von Ergebnistupel-Doppelberechnungen durch RACERPRO in den zweiphasigen Modi;
5. von NRQL gebotene Zusatzfunktionalität (welche nicht entwurfsrelevant und daher erst hier als eher implementationsspezifisch dargestellt wird);
6. Vorstellung der bereits erwähnten OWL TREE-Webanwendung.

5.3.3.6.1. Implementation der Cache- und Indexstrukturen Wir hatten dargestellt, dass die Verfügbarkeit von RACERPRO und auch die Performanz von NRQL durch die Vermeidung *repetitiver* und *unnötiger* Anfragen (s. Kap. 5.3.3.2.1, S. 341) erhöht wird. Im NRQL-ABox-Substrat (NRQL-ABox-Substrate) sind daher eine ganze Reihe von Cache- und Indexstrukturen implementiert. Wir unterscheiden im Folgenden nicht streng zwischen Cache und/oder Index, da die meisten unserer Caches indiziert ausgelegt sind.

Obwohl eine erste Version von NRQL nach Abschluss der Implementationsarbeiten (im Sommer 2003) auf Quellcode-Ebene direkt in RACERPRO integriert wurde, wodurch eine sehr viel breitbandigere lokale Nutzung der RACERPRO-API-Funktion ermöglicht wurde (keine RPC-artigen entfernten Aufrufe mehr erforderlich), sind die lokalen Cache- und Indexstrukturen für eine effiziente Anfragebeantwortung immer noch erforderlich. Gleichwohl die API-Funktionen von RACERPRO inzwischen ebenfalls entsprechende Indexstrukturen nutzen, so war dies jedoch zu Anfang der NRQL-Integrationsarbeiten noch nicht der Fall. Die NRQL-Indexstrukturen sind zudem besser für unsere Zwecke angepasst bzw. dediziert für NRQL entworfen.

Prinzipiell kann die Notwendigkeit von Caches jedoch auch als ein *Nachteil* angesehen werden, da

1. Informationen kopiert bzw. redundant gehalten werden müssen, und
2. das Cache-Synchronisationsproblem gelöst werden muss. Mangels einer entspr. Schnittstelle können wir das Problem unter Verwendung der WB-Fingerabdrücke (s.o.) hier nicht wirklich lösen.

Aus der Perspektive des Autors können diese Probleme nur durch Schaffung eines neuen DL-Systems wie MIDELOLA gelöst werden, bei dessen Entwurf diese Punkte bereits berücksichtigt werden.

Im NRQL-ABox-Substrat (NRQL-ABox-Substrate) sind nun drei Arten von Caches implementiert:

5.3 Vier konkrete Substrat-Anfragesprachen

1. Caches für die unterschiedlichen ABox-Assertionen (Konzept-Assertionen, Rollen-Assertionen, Attributs- und Constraint-Assertionen, die Menge der Individuen und KD-Individuen). Diese Caches sind als einfache Listen implementiert. Die Assertionen werden u.a. von den Anreicherungs- bzw. Aufbereitungsprozeduren für die unvollständigen Modi benötigt; das lokale Vorhalten dieser Assertionen vermeidet daher Netzwerkkommunikation und wiederholte Anfragen.
2. Caches, die die ABFen der *Konzept-Atome* in den verschiedenen Rollen unterstützen:

- a) Zur Unterstützung der *Tester-ABF* wird ein *Instanzenprüfungs-Cache* verwaltet. Dieser vermerkt für ein gegebenes Individuum i und Konzept C , ob i eine C -Instanz ist. Der Cache ist *dreiwertig*: Für gegebenes (i, C) liefern der Cache TRUE, FALSE oder UNKNOWN; der Cache ist als einfache Hashtabelle mit Schlüssel (i, C) implementiert.

Dieser Cache wird in der Substrat-Vorbereitungsphase mit TRUE-Einträgen anhand der Konzept-Assertionen angefüllt: Eine Assertion $i : C$ führt zu einem positiven (i, C) -Eintrag; je nach verwendetem Modus werden, den beschriebenen Anreicherungsprozeduren entsprechend, noch weitere TRUE-Einträge pro Assertion gemacht.

Der Cache ist zunächst natürlich unvollständig. Liefert der Cache auf Anfrage für (i, C) das Ergebnis UNKNOWN („Cache Fail“), so wird ausschließlich im vollständigen Modus RACERPRO mittels `dl-prover-individual-instance-p` kontaktiert, und der Cache anhand des erhaltenen Ergebnisses aktualisiert (der Eintrag (i, C) wird von UNKNOWN auf TRUE oder FALSE geändert). Im unvollständigen Modus wird das UNKNOWN-Ergebnis hingegen als FALSE ausgeliefert; aufgrund der OWA interpretieren die ABFen ein geliefertes FALSE stets als UNKNOWN.

- b) Zur Unterstützung der *Generator-ABF* wird ein *Konzeptinstanzen-Cache* verwaltet. Dieser liefert für ein Konzept C die Menge der *bereits bekannten* C -Instanzen; hierbei handelt es sich offensichtlich stets um eine Teilmenge der C -Instanzen. Der Cache ist durch eine einfache listenwertige Hashtabelle implementiert (wobei das Konzept als Schlüssel verwendet wird). Ist der Cache-Eintrag für C vollständig, so wird dies durch ein spezielles Markierungssymbol an erster Position des listenwertigen Eintrages vermerkt (ein Eintrag wie `(:complete a b)` stellt also eine vollständige, und `(a b)` eine unvollständige Instanzenmenge dar).

Wie auch der Instanzenprüfungs-Cache wird der Konzeptinstanzen-Cache zunächst dem eingestellten NRQL-Modus entsprechend von

den Anreicherungsprozeduren (unvollständig) vorgefüllt, und dann im Laufe der Anfragebeantwortung nach und nach für die in den Generator-Konzept-Atomen verwendeten Anfragekonzepte C vervollständigt.

Der Cache ist ganz wesentlich, um eine asynchrone Entkoppelung des Laufzeitverhaltens von RACERPRO und damit eine befriedigende Performanz zu erhalten. Ohne den Cache würde jeder Generator letztlich eine (synchrone) Instanzenretrieval-Anfrage an RACERPRO absetzen; der ABF-Prozess wäre bis zum Eintreffen des Ergebnisses blockiert. Obwohl RACERPRO lokal ebenfalls entsprechende Caches führt, ist die wiederholte Übermittlung des (u.U. umfangreichen) Anfrageergebnisses über eine Netzwerkschnittstelle ungünstig.

Der Cache ist nicht zuletzt auch unter dem Gesichtspunkt bzw. der Forderung, wann immer möglich (auch aus interner Perspektive) *informierte und inkrementelle Generatoren* bereitzustellen (s. Kap. 5.2.2.3.3, S. 298) erforderlich, und realisiert diese Forderung somit zumindest für die Menge der *bereits bekannten* C -Instanzen (diese Menge wird durch Seiteneffekte im Laufe der Anfragebeantwortung automatisch vervollständigt, sodass ab einem gewissen Zeitpunkt vollständig von RACERPRO entkoppelte asynchrone Anfragebeantwortung ermöglicht wird).

Der Cache stellt gleichzeitig einen Index bereit. Gegeben ein Konzept C , so liefert er unmittelbar die bereits bekannten C -Instanzen. Ein informierter inkrementeller Generator kann unter Verwendung eines über diese Menge laufenden Iterators unmittelbar realisiert werden.

- c) Beide Caches werden in der Vorbereitungsphase durch die Anreicherungsprozeduren vorgefüllt. Zur Anfragebeantwortungszeit werden diese Caches gemeinsam erweitert, sodass positive Instanzen tests nicht nur zu Einträgen im Instanzenprüfungs-Cache, sondern auch zu zusätzlichen Einträgen in den entsprechenden Konzeptinstanzen-Caches führen (sollten diese noch nicht vollständig sein), und andersrum.
3. Caches, die die ABFen der Rollen-Atome unterstützen. Diese Caches sind wesentlich zur Realisierung eines effizienten Navigierens durch die relationale Struktur einer ABox, s. Lemma 5, S. 323. Wir können für die Cache- und Indexstrukturen prinzipiell die gleichen Aussagen wie im Falle der Konzept-Atome machen (Realisierung informierter inkrementeller Generatoren, Realisierung asynchroner Entkoppelung von RACERPRO, etc.).
- a) Ein Cache zur Unterstützung der *Tester-ABF*. Dieser funktioniert analog zum Instanzenprüfungs-Cache. Der *Rollenprüfungs-Cache* gibt für

5.3 Vier konkrete Substrat-Anfragesprachen

ein Tripel (i, j, R) (i und j sind Individuen, R ist die Rolle) also TRUE, FALSE, oder UNKNOWN zurück. Es handelt sich um eine Hashtabelle mit Schlüssel (i, j, R) .

- b) Zur Unterstützung der ABFen der *Nachfolger- und Vorgänger-Generatoren* wird ein Cache verwaltet, der die bekannten Nachfolger und die bekannten Vorgänger für (i, R) verwaltet – der *Nachfolger- und Vorgänger-Cache*.

Diese Caches stellen also die leichtgewichtige Graphrepräsentation bereit; hier werden zwei einfache Hashtabellen mit Schlüssel (i, R) verwendet. Die Nachfolger bzw. Vorgänger von (i, R) werden wieder als einfache Listen gespeichert; wiederum wird ein spezielles Markierungssymbol (wie :complete) verwendet, um die Mengen als vollständig auszuzeichnen.

- c) Um Speicher zu sparen, wird kein dedizierter Cache für *Generatoren* vorgesehen. Die *Nachfolger- und Vorgänger-Caches* sind für unsere Zwecke ausreichend.
- d) Wie auch im Falle der Caches für die Konzept-Atome werden die Rollenprüfungs- und Nachfolger- und Vorgänger-Caches gemeinsam und konsistent um dynamisch durch RACERPRO-Aufrufe erworbenes „Zusatzwissen“ erweitert.

In dem durch Lemma 5 implizierten Traversierungsalgorithmus müssen auch die Eigenschaften der Rollen berücksichtigt werden. Um effizientes Traversieren zu ermöglichen, ist es mit einer lokalen, leichtgewichtigen Graphrepräsentation (wie durch die obigen Caches realisiert) daher noch nicht getan. Offensichtlich müssen wir auch für die Rolleneigenschaften (Transitivität, inverse Rolle, Ober- und Unterrollen) entsprechende Caches vorsehen, um entspr. Anfragen an RACERPRO während der Anfragebeantwortung zu vermeiden.

- 4. Caches, die die effiziente Auswertung der KPOs ermöglichen. Die KPOs sind Funktionen, die zum Zugriff auf TVs anhand von KD-Individuen, oder zum Zugriff von KD-Individuen anhand von Individuen benötigt werden. Natürlich ist auch hier ein lineares Suchen durch eine Liste der entspr. ABox-Assertionen (beispielsweise zur Ermittlung eines TVs) während der Anfragebeantwortung nicht vertretbar.

Bereits in der Substrat-Vorbereitungsphase werden daher die TVs durch Analyse der Assertionen (s. obige Caches) identifiziert und auch maximiert (s. Kap. 5.3.3.1.1, S. 335), und werden dann in entspr. Hashtabellen abgelegt, die einen indizierten Zugriff ermöglichen.

5.3.3.6.2. Implementation des Parsers und der ASG-Klassen und der ABFen Die Implementation der DSA für NRQL konnte vollständig ererbt werden. Die Implementation der BAA musste für NRQL jedoch entsprechend erweitert werden. Für NRQL wurden daher entsprechende Unterklassen der BAA-ASG-Klassen Konzept-Atom (Concept-Query), Rollen-Atom (Role-Query), sowie das neue Constraint-Atom (Constraint-Query) erforderlich.

Der NRQL-Parser (Klasse NRQL-Parser (NRQL-Parser)) wurde als gemeinsame Unterklasse des Parsers für die BAA (ABox-Anfrage-Parser (ABox-Query-Parser)) und des Parsers für die DSA abgeleitet, sodass er die konkrete Syntax beider Sprachen kennt. Erkennungsmethoden (get-expression-type) für NRQL-Anfragekonzepte (s. Def. 95, S. 347), Anfragerollen (s. Def. 94, S. 323) etc. mussten implementiert werden. Erweiterte Anfragekonzepte wurden durch syntaktisches Umschreiben (s. Kap. 5.3.3.3.1, S. 347) realisiert (Methode rewrite-query-atom). Bereits beim Übersetzen wird hier Kommunikation mit RACERPRO erforderlich, damit entschieden werden kann, ob es sich bei R um ein Datatype Property, Annotation Property, ein Attribut, oder um eine normale Rolle handelt. Die Anfrageketten in Constraint-Atomen werden ebenfalls durch einen syntaktischen Ersetzungsprozess wie in Kap. 5.3.3.3.2, S. 349 beschrieben aufgebrochen.

Die ABFen für das Constraint-Atom wurde mittels defquery-code implementiert. Für die anderen Atome sind hingegen die von der BAA ererbten Implementationen ausreichend. Sie verwenden die generischen Adapter-Methoden zur Inanspruchnahme der Dienste des DL-Systems (s. Kap. 5.3.2.2, S. 331). Diese Methoden wurden für die Klasse NRQL-ABox-Substrat (NRQL-ABox-Substrate) daher so redefiniert, dass sie zum einen die bereitgestellten Cache- und Indexstrukturen nutzen, und zum anderen den geforderten Modus (0 - 3, zweiphasig, ...) berücksichtigen. Aus der Perspektive der ABF-Implementationen ist all dies transparent.

5.3.3.6.3. Heuristische Verfeinerung der Bewertungsmethoden für den Optimierer Für NRQL wurde die h -Bewertungsfunktion bzw. die get-score-Bewertungsmethode für Knoten-Atome verfeinert (s. Kap. 5.2.2.2.3, S. 291). Die Standard-Implementation von get-score für Knoten-Atome zieht lediglich die Rolle des Atoms im Plan zur Berechnung der Bewertungsheuristik heran. Wie dargestellt, werden Knoten-Tester mit 99 (also maximal), und Knoten-Enumeratoren mit 50 bewertet.

Prinzipiell wollen wir sicherstellen, dass Nachfolger-Enumeratoren (für Rollen-Atome) immer höher bewertet werden als Knoten-Enumeratoren (für Konzept-Atome), um die navigierende Auswertungsstrategie sicherzustellen, und insbesondere auch, um Lemma 5, S. 323 zur Anwendung zu bringen. Da die Bewertung für Nachfolger-Generatoren bei 55 beginnt, können die Werte im Intervall

5.3 Vier konkrete Substrat-Anfragesprachen

[50; 55] zur unterschiedlichen Wertung konkurrierender Knoten-Generatoren genutzt werden.

Offensichtlich ist es schwierig, die „Komplexität eines Anfragekonzeptes“ zur Präferenzierung zu nutzen. Einfach kann hingegen eventuell vorhandenes Wissen über die *Kardinalitäten der Extensionen der Atome* genutzt werden. Diese Information ist letztlich (u.U. unvollständig) in den Cache- und Indexstrukturen, aber auch durch die gespeicherten Extensionen bereits ausgeführter Rümpfe gegeben.

Nehmen wir an, dass die Konzept-Atome (x, C) und (y, D) im momentan betrachteten Teil-Plan die Rolle eines Generators bekommen würden. Sollte als nächstes Atom im Plan nun (x, C) oder (y, D) gewählt werden? Anhand der Caches seien nun die Mengen $\mathcal{C}$ und $\mathcal{D}$ mit $\mathcal{C} \subseteq (x, C)^\varepsilon$ und $\mathcal{D} \subseteq (y, D)^\varepsilon$ bekannt (Gleichheit gilt, wenn die Caches bereits vollständig sind); $\mathcal{C}$ und $\mathcal{D}$ sind also die *bekannten C- bzw. D-Konzeptinstanzen*: $\mathcal{C} = \text{known\_concept\_instances}(C)$, $\mathcal{D} = \text{known\_concept\_instances}(D)$. Es drängen sich dann zwei konkurrierende Ideen auf:

1. Gilt $|\mathcal{C}| < |\mathcal{D}|$, so stellen wir sicher, dass $h((x, C)) > h((y, D))$ gilt; somit wird der Optimierer (x, C) wählen.

Die Motivation ist, dass beim Navigieren die *Selektivität* von (x, C) höher als die von (y, D) ist, und daher der Verzweigungsfaktor im Suchbaum bei der Wahl von (x, C) geringer ausfällt.

Allerdings kann der „Schein auch trügen“; letztlich sind $\mathcal{C}$ und $\mathcal{D}$ nur Approximationen. Die Bewertungsfunktion kann auch so gestaltet werden, dass nur im Falle *sicheren Wissens* (wenn also $\mathcal{C} = (x, C)^\varepsilon$ und $\mathcal{D} = (y, D)^\varepsilon$ gilt) diese Präferenzierung vorgenommen wird.

2. Wenn $\mathcal{C}$ und $\mathcal{D}$ hingegen nur approximativ bekannt sind, wissen wir in Wirklichkeit nichts über die „echten“ Extensionen $(x, C)^\varepsilon$ und $(y, D)^\varepsilon$. Es ist daher genauso plausibel, $|(x, C)^\varepsilon| \approx |(y, D)^\varepsilon|$ anzunehmen.

Liegen nun $\mathcal{C}$ und $\mathcal{D}$ (in den Cache- und Indexstrukturen) vor, so können sie im Rahmen einer *zweiphasigen Tupelberechnung* zur Bereitstellung der „billigen Tupel“ in *Phase 1* genutzt werden. In *Phase 2* müssten daher (unter der Annahme $|(x, C)^\varepsilon| \approx |(y, D)^\varepsilon|$) weniger „teure Tupel“ zur Vervollständigung für das Atom (y, D) als für das Atom (x, C) errechnet werden. Daher wäre (y, D) zu bevorzugen und somit $h((y, D)) > h((x, C))$ sicherzustellen, denn die Wahrscheinlichkeit, dass ein solcher Kandidat genutzt werden kann, nimmt mit der Position des Atoms im Plan ab.

Wir nehmen vorweg, dass sich 1. als besser herausgestellt hat, da die durch 1. erzielte Reduzierung des Verzweigungsfaktors für die Suche selbst zur Berechnung der Tupel in Phase 1 außerordentlich wichtig ist; zudem ist die Annahme $|(x, C)^\varepsilon| \approx |(y, D)^\varepsilon|$ für die meisten WBs bzw. Anfragen eher unrealistisch.

Die Wahl von 1. führt uns dann zu folgender Bewertungsfunktion, welche wie gewünscht Werte im Intervall $[50; 55[$ annimmt:

$$h((a, C)) =_{def} 50 + 5 * \frac{|V| - |\text{known\_concept\_instances}(C)|}{|V| + 1}$$

Eine ähnliche Bewertungsabstufung ist auch für die Rollen-Atome denkbar; hier könnte die *mittlere Anzahl der (bekannten) R-Nachfolger pro Individuum* bzw. die *mittlere Anzahl der (bekannten) R-Vorgänger pro Individuum* Verwendung finden. Diese Idee wurde im Rahmen der Arbeit nicht weiter verfolgt.

Rollen-Atome, die *negierte Rollen* verwenden, werden durch die Implementation hochgradig negativ bewertet, da für ihre Auswertung die ABox verändert werden muss (s. Lemma 4, S. 322), was zur Invalidierung aller RACERPRO-Indexstrukturen führt. Die so veränderte ABox muss dann auf Erfüllbarkeit getestet werden, und zudem ist Lemma 5, S. 323 für negierte Rollen nicht anwendbar. Rollenatome mit negierten Rollen sind also in NRQL nicht effizient auswertbar – ihre Auswertung sollte soweit als möglich „nach hinten“ im Plan verschoben werden, sodass sie möglichst nur als Tester auftreten und ihre Auswertung zur Anfragebeantwortungszeit so selten wie möglich vorkommt.

Schließlich wurde die `get-score`-Methode auch für das Constraint-Atom definiert.

5.3.3.6.4. Unterstützung der zweiphasigen Modi durch RACERPRO Betrachten wir das Beispiel aus Kap. 5.3.3.5.3. Da bereits in Phase 1 die Individuen $\{doris, betty\}$ als Instanzen von *spouse* erkannt wurden, ist es in Phase 2 ausreichend, lediglich die verbliebenen Instanzen $\{doris, betty, eve\} \setminus \{doris, betty\} = \{eve\}$ auf *spouse*-Mitgliedschaft zu überprüfen. RACERPRO unterstützt die *explizite Spezifikation von Kandidatenmengen* beim Instanzen-Retrieval mittels eines `:candidates`-Arguments in Basis-Retrievalfunktionen wie `concept-instances`. Hier würde also ein Aufruf (`concept-instances spouse :candidates '(eve)`) abgesetzt werden. Die für die Klasse `NRQL-ABox-Substrat` (`NRQL-ABox-Substrate`) überschriebene Methode `dl-prover-get-concept-instances` akzeptiert nun auch dieses optionale Argument.

5.3.3.6.5. Zusätzliche Funktionalität Die NRQL-Maschine bietet noch umfangreiche Zusatzfunktionalität, die hier nicht ausführlich diskutiert werden soll. Für Details sei auf das Handbuch [Racer Systems 2006, Kap. 6] verwiesen. Wir erwähnen hier nur kurz

1. die QBox,
2. Anfragerealisierung,

5.3 Vier konkrete Substrat-Anfragesprachen

3. NRQL-Regeln,
4. definierte Anfragen, und
5. komplexe TBox-Anfragen,

und beschreiben jedes Merkmal mit ein paar Worten (in dieser Reihenfolge).

Für NRQL wurde die QBox implementiert. Der verwendete Anfrage-Subsumptionsalgorithmus ist der bereits für die BAA in Kap. 5.3.2.1.3 vorgestellte, mit entsprechenden Erweiterungen für die Constraint-Atome. Ist die NRQL-Anfrage hybrid, so werden die DSA-Atome ignoriert (wir werden später anhand einer eingeschränkten DISKQL zeigen, wie Anfragesubsumption auch für eine hybride Anfragesprache im Rahmenwerk entschieden werden kann). Zudem wurde Anfragerealisierung für NRQL-Anfragen durch ABox-Realisierung implementiert (s. Kap. 5.2.2.2.1, S. 285).

NRQL bietet einfache Regeln. Diese stellen einen nutzerfreundlichen, deklarativen Mechanismus zur Erweiterung von ABoxen um zusätzliche Assertionen bereit. Eine Regel hat eine *Prämisse (Vorbedingung)* und eine *Konklusion (Nachbedingung, Konsequenz)*. Die Prämisse einer NRQL-Regel ist ein gewöhnlicher NRQL-Rumpf, und die Konklusion besteht aus einer Liste von *generalisierten ABox-Assertionen* (s.u.).

In einer *generalisierten ABox-Assertion* lassen wir statt einfacher ABox-Individuen auch NRQL-Variablen zu; die generalisierte Assertion wird durch Substitution der Variablen gegen die aktuelle Bindung in eine gewöhnliche ABox-Assertion überführt, die dann zur ABox hinzugefügt wird. Tatsächlich können diese Regeln auch genutzt werden, um ABox-Assertionen aus der ABox zu entfernen (s.u.).

Es werden die generalisierten Assertionen *instance*, *related*, *constrained*, *constraints* unterstützt (s. Kap. 2.2.8.1). In einer Konklusion können zudem neue ABox-Individuen erzeugt werden; hierfür wird der *new-ind*-Operator auf ein Objekt (Individuum oder Variable) in der generalisierten Assertion angewendet. Wir geben weiter unten ein Beispiel an. Zum Entfernen von Assertionen mittels Regeln werden die generalisierten „Assertionen“ *forget-concept-assertion*, *forget-role-assertion*, *forget-constrained-assertion*, *forget-constraint-assertion* genutzt.

Die NRQL-Maschine kann zwar verwendet werden, um eine Reihe von Regeln zu definieren, anzuwenden und zu verwalten, jedoch ist zur *Sicherstellung der Terminierung* keine feste *Regelanwendungsstrategie* implementiert. Stattdessen wird eine API-Funktionen zur Prüfung der Anwendbarkeit einer Regel bereitgestellt (*rule-applicable-p*), sowie eine Funktion, die alle anwendbaren Regeln zurückliefert (*applicable-rules*). Der Nutzer bzw. Client kann dann eine oder mehrere dieser anwendbaren Regeln explizit mittels *execute-rule* anwenden bzw. „feuern“. Natürlich kann das Vorbereiten und Feuern einer Re-

gel auch in einem Schritt geschehen (mittels `firerule`). Wiederholtes Regelanwenden muss daher durch den Client bzw. Nutzer veranlasst und kontrolliert werden. Etwaige Terminierungsprobleme werden daher nicht durch NRQL verursacht.

Die folgende Beispiel-Sitzung am NRQL-Terminal demonstriert das Regel-API anhand der RACERPRO-Beispielwissensbasis aus Kap. 2.2.8.1:

```
> (define-primitive-role has-sister :parent has-sibling)

HAS-SISTER

> (all-role-assertions)

(((BIRGIT ALINA) HAS-DAUGHTER)
 ((BIRGIT MARA) HAS-DAUGHTER)
 ((MARGRIT BIRGIT) HAS-DAUGHTER)
 ((RUDOLF BIRGIT) HAS-DAUGHTER)
 ((MARGRIT MICHAEL) HAS-SON)
 ((RUDOLF MICHAEL) HAS-SON))

> (firerule (and (?x ?y has-parent) (?y ?z has-daughter)
                (neg (?x ?z has-sister)))
            ((related ?x ?z has-sister)))

(((RELATED MICHAEL BIRGIT HAS-SISTER))
 ((RELATED MARA ALINA HAS-SISTER))
 ((RELATED ALINA MARA HAS-SISTER)))

> (instance michael father)

> (prepare-abox-rule (and (?x father) (?x nil has-child))
                    ((related ?x (new-ind child-of ?x) has-child)
                     (instance (new-ind child-of ?x) person)))

(:RULE-2 :READY-TO-RUN)

> (applicable-rules)

(:RULE-2)

> (execute-rule :rule-2)

(((RELATED MICHAEL CHILD-OF-MICHAEL HAS-CHILD)
 (INSTANCE CHILD-OF-MICHAEL PERSON)))
```

5.3 Vier konkrete Substrat-Anfragesprachen

> (applicable-rules)

NIL

Im obigen Beispiel ist insbesondere die Verwendung der NAF-Atome in den Prämissen interessant. Die Regeln zeigen aufgrund dieser ein *nichtmonotones Verhalten* und sind daher nur einmal anwendbar. Mit ihrer Hilfe können daher auch Terminierungsprobleme adressiert werden, da die Mehrfachanwendung einer solchen Regel mittels NAF-Atomen verhindert werden kann. In aktuellen Vorschlägen für SEMANTIC WEB-Regelsprachen wie SWRL [Horrocks u. a. 2004a] werden keine entsprechenden nichtmonotonen Operatoren angeboten.

Sollen mit Hilfe von Regeln jedoch constraint-Assertionen zur ABox hinzugefügt werden, so gibt es hier zunächst das Problem, das constraint-Assertionen ja KD-Individuen referenzieren, und NRQL-Variablen nicht an KD-Individuen gebunden werden können (s.o.). Wir erlauben die bereits bekannten Attribut-KPOs daher auch in Regel-Konklusionen:

```
> (firerule (and (?x (an age)) (same-as ?x michael))
  ((constrained ?x (new-ind age-of ?x) age)))

(((CONSTRAINED CHILD-OF-MICHAEL AGE-OF-CHILD-OF-MICHAEL AGE)))

> (firerule (and (michael top) (child-of-michael top))
  ((constraints (= (age michael) (+ 37 (age child-of-michael))))))

(((CONSTRAINTS (= AGE-OF-MICHAEL (+ 37 AGE-OF-CHILD-OF-MICHAEL))))))

> (retrieve (?x) (?x (= age -2)))

(((?X CHILD-OF-MICHAEL)))
```

NRQL bietet *definierte Anfragen*. Hierbei handelt es sich um einen einfachen Makro-Mechanismus. Zyklische Definitionen sind nicht möglich; prinzipiell ist in NRQL daher ein einfaches nichtrekursives Datalog mit (stratifizierter) Negation verfügbar:

```
> (defquery uncle-of (?x ?z)
  (and (?x male-person) (?x ?y has-sibling) (?y ?z has-child)))

UNCLE-OF

> (retrieve (?x ?y) (?x ?y uncle-of))
```

```

(((?X MICHAEL) (?Y ALINA)) ((?X MICHAEL) (?Y MARA)))

> (defquery uncle-of-female-person (?x)
    (project-to (?x) (and (?x ?y uncle-of) (?y female-person))))

UNCLE-OF-FEMALE-PERSON

> (retrieve (?x) (?x uncle-of-female-person))

(((?X MICHAEL)))

> (retrieve (?x) (neg (?x uncle-of-female-person)))

(((?X ALINA)) ((?X MARA)) ((?X BIRGIT)) ((?X MARGRIT)) ((?X RUDOLF)))

```

Prinzipiell sind die Atome $(?x ?y \text{ uncle-of})$ und $(?x \text{ uncle-of-female-person})$ nun mehrdeutig: Ist mit *uncle-of* nun eine evtl. existierende gleichnamige Rolle, oder die definierten Anfrage gemeint? Ein analoges Problem entsteht auch für $(?x \text{ uncle-of-female-person})$ bzgl. evtl. vorhandener gleichnamiger Konzeptnamen. Bei Mehrdeutigkeiten wird immer zu Gunsten des Konzeptnamens bzw. der Rolle entschieden. Die Verwendung der definierten Anfrage kann jedoch mittels der speziellen Syntax $(\text{substitute}(\text{uncle-of-female-person } ?x))$ *erzwungen* werden. Sowohl *substitute*-Atome als auch definierte Anfragen werden durch einfaches syntaktisches Umschreiben bzw. Makro-Expansion implementiert (insofern sind keine entspr. Anfrage (Query)-Unterklassen erforderlich).

Abschließend seien noch die *komplexen TBox-Anfragen* erwähnt. Hier wird die *Taxonomie* als *relationale Struktur* gedeutet, und die Knoten-Kanten-Struktur des GAGs wird in ein Daten-Substrat (Data-Substrate) überführt. Das sog. TBox-Substrat (TBox-Substrate) ist daher isomorph zur Taxonomie der TBox.

Ein Knoten im TBox-Substrat entspricht einem Knoten in der Taxonomie. Ein Taxonomie-Knoten repräsentiert eine Menge von synonymen Konzeptnamen. Pro Äquivalenzklasse wird ein Stellvertreter i als Name des Knotens ausgewählt; alle Elemente der Äquivalenzklasse werden als Konjunkte in $L_V(i)$ aufgenommen.

Eine Kante, die eine direkte Subsumptionsbeziehung repräsentiert, wird mit *has_child* beschriftet; der transitive Abschluss ist als *has_descendant*-Kanten verfügbar. Zusätzlich stehen die Inversen zur Verfügung (*has_parent*, *has_ancestor*).

Einfache Atome (s. Kap. 5.2.2.8.1, S. 309) bzw. entspr. eingeschränkte DSA-Atome sind nun ausreichend, um diese Struktur zu befragen. Die spezielle Syntax der Atome wird durch syntaktisches Umschreiben versteckt; für die eingeschränkten DSA-Atome gilt

5.3 Vier konkrete Substrat-Anfragesprachen

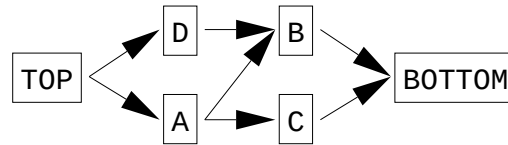


Abbildung 5.21.: Beispiel-Taxonomie zur Demonstration von tbox-retrieve.

$\mathcal{L}_{QE} =_{def} \{has\_child, has\_parent, has\_ancestor, has\_descendant\}$ und $\mathcal{L}_{QV} =_{def} \mathcal{N}_C$.

Zur Illustration sei die Beispiel-TBox $\{b \sqsubseteq a, c \sqsubseteq a, b \sqsubseteq d\}$ betrachtet. Die Taxonomie der TBox ist in Abb. 5.21 dargestellt. Die Anfrage

```
(tbox-retrieve
  (?x $?u)
  (and (?x ?y has-descendant)
        (?x ?z has-descendant)
        (?x $?u has-child)))
```

liefert dann das Ergebnis:

```
(( (?X *TOP*) ($?U A) )
  (( ?X *TOP*) ($?U D) )
  (( ?X A) ($?U C) )
  (( ?X A) ($?U B) ) ).
```

5.3.3.6.6. Ein prototypisches SWIR-Szenario und der OWL TREE Ein prototypischer SWIR-Dienst wurde von Matthias Flege, Thomas Hirsch, Torben Rebhan, und Kamil Sokolski im Rahmen eines Projektseminars an der TUHH geschaffen und vom Autor betreut – die OWL TREE-Webanwendung. Das Szenario ist das eines Universität-ISs; das Vokabular und die Beispiel-Daten werden durch den LUBM (s. Kap. 3.2.4.1) bereitgestellt.

Die OWL TREE-Webanwendung ermöglicht die endnutzerfreundliche grafisch-interaktive Erstellung von *baumstrukturierten* konjunktiven NRQL-Anfragen (s. Abb. 5.22a), das Absetzen dieser an eine RACERPRO-Server-Instanz, und die anschließende Ergebnispräsentation und -inspektion (s. Abb. 5.22b). Da es sich um eine Webanwendung handelt, wird aus Endnutzerperspektive lediglich ein Web-Browser zur Nutzung des OWL TREE-Dienstes benötigt.

Prinzipiell können mit dem OWL TREE beliebige OWL-WBs befragt werden; primär wurde er jedoch für die LUBM-Ontologie (s. Anhang A.1 und A.2) entworfen. Die NRQL-Ergebnistupel werden in einer HTML-Tabelle präsentiert. Da die Tupelkomponenten nun RDF-Instanzen bzw. URIRefs sind, werden sie als anklickbare Hyperlinks präsentiert. Das OWL TREE-Team hat den

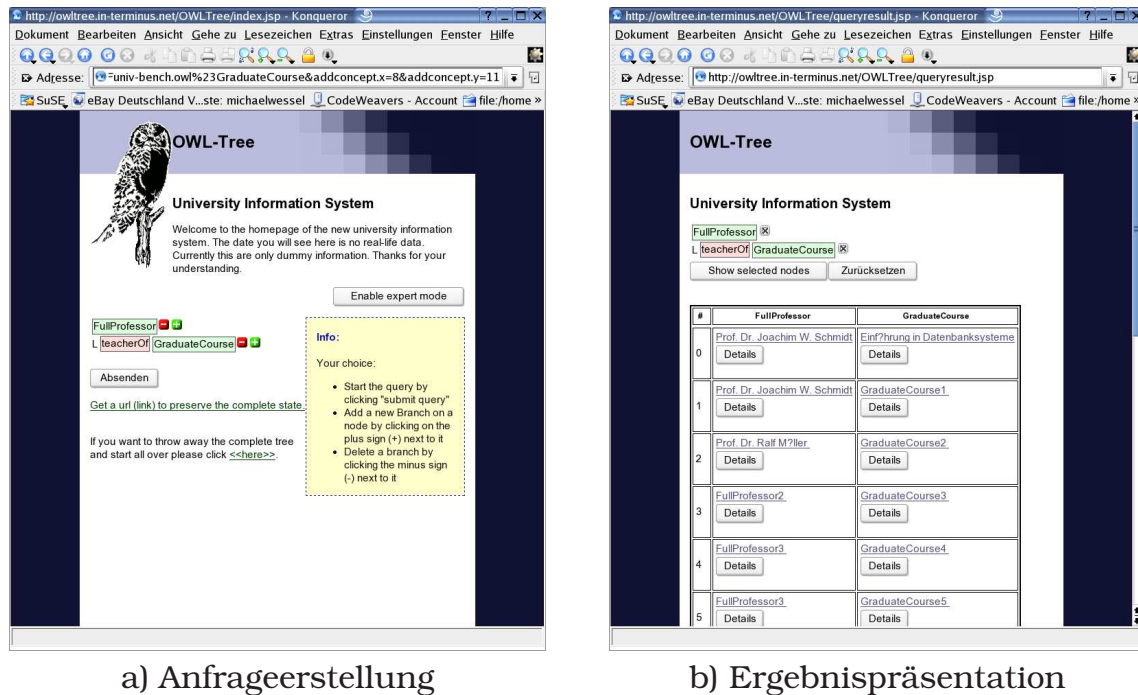


Abbildung 5.22.: Oberfläche eines prototypischen SWIR-Dienstes – Die OWL TREE-Webanwendung.

LUBM-Generator zur Erzeugung der Benchmark-Daten zudem so erweitert, dass nicht nur artifizielle synthetische URIRefs als RDF-Instanzen der LUBM-OWL-Klassen generiert, sondern existierende “reale” Web-Ressourcen für diesen Zweck genutzt werden. Beispielsweise wurde die vom LUBM-Generator erzeugte Instanzenmenge der Klasse `employee` um URIRefs erweitert, die Homepages von Mitarbeitern der TUHH referenzieren. Für die Klasse `course` wurden Homepages von TUHH-Vorlesungen verwendet, URIRefs auf PDF-Publikationen des Arbeitsbereiches STS als Instanzen der Klasse `publication`, etc. Ist eine solche Instanz nun in einem `NRQL`-Ergebnistupel auf der Ergebnis-Präsentationsseite enthalten, so wird durch Klick auf den entspr. Hyperlink die real existierende Web-Ressource aufgesucht und im Webbrowser geöffnet (die Homepage des Professors, die PDF-Publikation im Adobe Reader, etc.).

In Abb. 5.22a) ist eine grafische `NRQL`-Anfrage zur Ermittlung von Vorlesungen im Hauptstudium (Klasse `GraduateCourse`) gezeigt, die von festangestellten Professoren (Klasse `FullProfessor`) gehalten werden. Das Anfrageergebnis ist in Abb. 5.22b) zu sehen. Durch Klick auf Prof. Dr. Ralf Moeller wird die Homepage von Ralf Möller geöffnet.

5.3.4. (Leitdomäne 1) Das DLMAPS-System und die DISKQL

Das DLMAPS-System als Instanz eines ISs in Leitdomäne 1 nutzt als extensionale Komponente die entworfenen Karten-Substrate. Der MAP VIEWER (s. Abb. 5.13) ist die Inspektionskomponente des DLMAPS-Systems. Zur „Vervollständigung“ des DLMAPS-Systems reichen wir nun noch die Anfragesprache DISKQL („DISK Query Language“) nach, die z.T. durch [Meyer 1994a;b] inspiriert ist. Wir zeigen, wie die DISK-Referenzanfragen (s. Kap. 3.1.3.3) mittels DISKQL formuliert werden können. Wir stellen auch einen Algorithmus zur Entscheidung von Subsumption zwischen eingeschränkten DISKQL-Anfragen vor.

Wir beschreiben zunächst die Nutzung der Karten-Substrate im DLMAPS-System, schildern dann weitere Anforderungen sowie den Entwurf der DISKQL. Wie in Kap. 3.1.6 gefordert, entwerfen wir DISKQL als hybride räumlich-thematische Anfragesprache. Wir geben die DISK-Referenzanfragen in konkreter DISKQL-Syntax an und stellen dann den Algorithmus zur Entscheidung von Subsumption zwischen eingeschränkten DISKQL-Anfragen vor. Schließlich erläutern wir einige Implementationsdetails. Wir erlauben uns hier eine im Vergleich zu NRQL bzw. im Vergleich zum SWIR-Dienst kürzere Darstellung (und verzichten auch auf UML-Diagramme etc.), da das prinzipielle Vorgehen zur Erweiterung und Anpassung des Gesamtrahmens zur Unterstützung spezieller IS-Domänen bzw. -Szenarien inzwischen deutlich geworden sein dürfte.

5.3.4.1. Nutzung der Karten-Substrate im DLMAPS-System

In Kap. 5.1.3.1 hatten wir die (größtenteils abstrakten) Karten-Substrate zur Repräsentation der DISK vorgestellt (s. Unterklassen von Karten-Substrat (Map-Substrate) in Abb. 5.11). Die räumlichen und thematischen Aspekte der DISK können mit ihnen auf unterschiedliche Weise repräsentiert werden.

Als *externes* ABox-DL-System nutzen wir den RACERPRO-Server. Dementsprechend spezialisieren wir die Klassen ABox-DLS-Explizite-Relationen-Karten-Substrat (ABox-DLS-Explicit-Relations-Map-Substrate) und ABox-DLS-Implizite-Relationen-Karten-Substrat (ABox-DLS-Implicit-Relations-Map-Substrate) und erhalten somit die konkreten Klassen RACERPRO-Explizite-Relationen-Karten-Substrat (RACERPRO-Explicit-Relations-Map-Substrate) und RACERPRO-Implizite-Relationen-Karten-Substrat (RACERPRO-Implicit-Relations-Map-Substrate).

Für die *hybriden* Karten-Substrate (Klasse Hybrides-Karten-Substrat (Hybrid-Map-Substrate)) hatten wir S_1 in Abb. 5.11 auf eine gemeinsame Unterklasse der Klassen Einfache-Beschreibungen-Substrat (Simple-Descriptions-Substrate) und Qualitatives-Geometrisches-RCC-Substrat (Qualitative-Geometric-RCC-Substrate) eingestellt, und S_2 auf eine Unterklasse von ABox-Adapter-Substrat (ABox-Adapter-

Substrate). Da wir für S_2 die BAA nutzen wollen, legen wir S_2 auf die Klasse RACERPRO-BAA-Substrat (RACERPRO-BAA-Substrate). Zur Erinnerung sei erwähnt, dass Knoten in S_1 somit geometrische Objekte (Instanzen von SDTs) sind, zwischen denen RCC-Relationen anhand der Geometrie errechnet und optional als Kanten materialisiert werden können, und deren Knoten mit beliebigen einfachen Beschreibungen annotiert werden können. Knoten in S_2 sind hingegen stets (virtuelle) RACERPRO-ABox-Individuen.

Die *nichthybriden* Karten-Substrate erweitern entweder die MIDeLoRA-ABox (Klasse ABox) oder verwenden einfache Beschreibungen oder DL-Beschreibungen. Wie in Kap. 5.1.3.1 dargestellt, ist die Nutzung eines ABox-DL-Systems nicht zwingend, da durch die Klassen Einfache-Beschreibungen-Explizite-Relationen-Karten-Substrat (Simple-Descriptions-Explicit-Relations-Map-Substrate) und DL-Beschreibungen-Explizite-Relationen-Karten-Substrat (DL-Descriptions-Explicit-Relations-Map-Substrate) (wenn auch ausdruckschwächere) Alternativen vorhanden sind. In diesem Fall muss statt der BAA die DSA verwendet werden. Die Klassen zur Bereitstellung einer räumlichen MIDeLoRA-ABox sind bereits in Abb. 5.11 dargestellt: MIDeLoRA-Explizite-Relationen-Karten-Substrat (MIDeLoRA-Explicit-Relations-Map-Substrate) und MIDeLoRA-Implicite-Relationen-Karten-Substrat (MIDeLoRA-Implicit-Relations-Map-Substrate). Die operationalen Vorteile, die sich aus dieser Repräsentation ergeben, diskutieren wir erst in Kap. 6. Die BAA ist auch für MIDeLoRA definiert; wir benötigen daher ein MIDeLoRA-BAA-Substrat (MIDeLoRA-BAA-Substrate) statt einer ABox.

5.3.4.2. Verfeinerte Anforderungen an die DISKQL

Im Vergleich zu nRQL muss die initiale Anforderungsanalyse aus Kap. 3.1.6 nicht sehr viel weiter verfeinert bzw. erweitert werden. Wir möchten hier nur eine weitere Anforderung mit aufnehmen: *Erhöhung der Nutzerfreundlichkeit*.

Die DISKQL muss Atome zur Adressierung thematischer und räumlicher Aspekte bereitstellen. Wir werden diese Atome weiter unten vorstellen. Je nach verwendetem Karten-Substrat sind nun verschiedene Arten dieser räumlichen und thematischen Atome zur Formulierung (u.U. hybrider) DISKQL-Anfragen kombinier- und verwendbar. Idealerweise sollte es für den Nutzer des DL-MAPS-Systems jedoch transparent sein, welche Art von Repräsentation intern verwendet wird.

Durch entsprechendes *syntaktisches Umschreiben* kann eine DISKQL-*Endnutzersyntax* bereitgestellt werden, die von der unterschiedlichen Syntax der jeweils anwendbaren Atome und von der u.U. vorhandenen Hybridizität abstrahiert und die Abhängigkeit der Anfragesprache von der logischen Datenrepräsentation daher minimiert.

5.3 Vier konkrete Substrat-Anfragesprachen

In einem ersten Schritt zur Erhöhung der Nutzerfreundlichkeit erlauben wir die Verwendung natürlichsprachlicher räumlicher Präpositionen in räumlichen Atomen. Diese werden einfach als Synonyme für bestimmte RCC-Relationen definiert. Die Liste der Synonym-Relationen kann bei Bedarf einfach erweitert werden; beispielsweise gelten folgende Abbildungen:

```
{:contains, :inside-of}  $\mapsto$  PPI,  
{:inside, :within, :contained-in}  $\mapsto$  PP,  
{:adjacent, :externally-connected, :touched, :touches,  
:touched-by, :connects, :connects-from, :connects-to,  
:borders, :leads-from, :leads-to}  $\mapsto$  EC,
```

etc.

5.3.4.3. Entwurf der DISKQL

Für die DISKQL nutzen wir nun einfach das vorhandene Repertoire an bereits vorgestellten Atomen. Aufgrund der vorausgegangenen Schilderungen ist klar, dass bestimmte Atome nur auf bestimmten Karten-Substraten anwend- bzw. auswertbar sind.

Zur Adressierung der *thematischen Aspekte* sind nun, je nachdem welche Art von Karten-Substrat aus Abb. 5.11 verwendet wird, die BAA-Atome, die einfachen Atome (Klasse Einfaches-Beschreibungs-Atom (Simple-Description-Query)) bzw. die DSA-Atome, aber auch die RACERPRO-Beschreibungs-Atome (Klasse RACERPRO-Beschreibungs-Atom (RACERPRO-Description-Query)) nutzbar.

Zur Adressierung der *räumlichen Aspekte* (bzw. der *räumlichen Eigenschaften* und *räumlichen Relationen*) sind diese Atome prinzipiell genauso nutzbar, doch gilt dies nur für räumliche Aspekte, die *explizit* repräsentiert sind.

Liegt beispielsweise ein Karten-Substrat mit expliziten RCC-Kanten vor, so können natürlich einfache Atome (z.B. mit $\mathcal{L}_{QE} = \text{RCC8}$) als räumliche Atome verwendet werden. Sind die RCC-Kanten jedoch lediglich implizit vorhanden, so sind diese Atome nicht anwendbar. Die Situation ist analog wie für die *:satisfies- λ* -Atome (s. Kap. 5.2.2.8.3, S. 312). Diese λ -Atome können zur Implementierung räumlicher Atome (mit den bereits diskutierten Nachteilen) verwendet werden.

Betrachten wir beispielsweise die räumliche Eigenschaft „Fläche“. Natürlich könnten wir für jedes Objekt seine Fläche im Voraus berechnen und entweder numerisch in der konkreten Domäne der ABox, oder als Konjunkt in $L_V(i)$ des Knotens in S_1 explizit repräsentieren. Im ersten Fall könnten wir ein BAA-Atom nutzen

```
(?*x (and (min size-of-area 10000) (max size-of-area 20000))),
```

und im zweiten Fall das DSA-Atom

(?x (:and (:predicate (>= 10000)) (:predicate (<= 20000))))).

Ist die Fläche jedoch lediglich implizit (durch die Geometrie des Knotens) repräsentiert, so müssen wir entweder ein λ -Atom (mit den diskutierten Nachteilen) zur Berechnung nutzen, oder wir benötigen ein neues spezialisiertes räumliches Knoten-Atom. Sobald also das Substrat die fragliche räumliche Eigenschaft bzw. Relation implizit lässt, benötigen wir ein entspr. räumliches Atom zur Adressierung dieses Aspektes. Das räumliche Atom kann die Räumlichkeit bzw. Reichhaltigkeit der geometrischen Repräsentation nutzen und wird auch *informiert* sein (im Gegensatz zum λ -Atom).

Warum sollte das Substrat nun nicht alle (jemals benötigten) räumlichen Eigenschaften und Relationen im Voraus explizit machen? Dies ist zum einen aufgrund der *Reichhaltigkeit* einer geometrischen Repräsentation nicht in jedem Fall möglich. Oft wird erst durch die Anfrage bestimmt, welche Eigenschaften und Relationen erforderlich sind. Sicherlich könnte das Substrat dann speziell zur Befriedigung der Anfrage aufbereitet werden, sodass lediglich die geforderten Eigenschaften und Relationen explizit gemacht würden. Doch ist dies mit offensichtlichen Laufzeit- und Speicherplatznachteilen versehen (wir hatten dies bereits anhand der n^2 -Platzkomplexität der RCC-Netze gesehen). Wir können es als charakteristisch für diese Domäne betrachten, dass eine einfache Art von *räumlicher Inferenz* zur Beantwortung räumlicher Atome erforderlich wird, denn es müssen räumliche Aspekte angefragt werden, die lediglich implizit bzw. intrinsisch repräsentiert sind. Räumliche Datenbanken (und die hier vorgestellten Karten-Substrate) unterstützen diese eingeschränkte räumliche Inferenz durch räumliche Indexstrukturen und spezielle geometrische Auswertungsfunktionalität.

Wir unterscheiden nun räumliche Knoten- und Kanten-Atome:

- *Räumliche Knoten-Atome* beziehen sich auf *räumliche Eigenschaften* von Karten-Objekten (Fläche eines Polygons, Länge einer Kette, etc.).
- *Räumliche Kanten-Atome* beziehen sich auf *räumliche Relationen* zwischen Karten-Objekten. Hier sind wiederum die RCC-Relationen zu nennen, die nun jedoch nicht vorweg berechnet und materialisiert werden, sondern erst während der Anfragebeantwortung auszuwerten sind. Zum anderen sind hier auch Relationen zu nennen, deren Charakter ein Materialisieren im Voraus nicht sinnvoll erscheinen lässt. Hierzu gehören z.B. (metrische, quantitative) Abstandsrelationen (s.u.).

5.3.4.3.1. Räumliche Knoten-Atome Die einfachsten räumlichen Knoten-Atome sind Atome zur Selektion von Knoten anhand ihres *räumlichen Typs* (SDTs) („Zeige mir alle Flächen in der Karte!“), sog. *SDT-Selektions-Atome*:

- Die Extension von (?x :polygon) ist die Menge aller Polygon-Knoten.

5.3 Vier konkrete Substrat-Anfragesprachen

- Statt `:polygon` können auch `:chain`, `:line`, `line-or-chain`, `:symbol` und `:point` verwendet werden (`:symbol` wird für kartografische Symbole verwendet, die spezielle Punkte sind).

Zur Selektion von Knoten anhand bestimmter numerischer räumlicher Eigenschaften (wie Länge, Fläche etc.) sehen wir ebenfalls neue Atome vor. Als *Selektionsbedingung* können beliebige arithmetische COMMON LISP-Ausdrücke über einer freien Variablen verwendet werden, die an den numerischen Wert der entspr. räumlichen Eigenschaft des gerade betrachteten Knotens gebunden wird:

- Die Extension von `(?x (:area (>= area 10000) (<= area 20000)))` ist die Menge aller Polygon-Knoten, deren Fläche im Intervall $[10000; 20000]$ liegt.
- Statt `:area` können auch `:length` und `:circumference` für die räumlichen Eigenschaften Länge und Umfang verwendet werden; die Liste kann bei Bedarf einfach erweitert werden.

5.3.4.3.2. Räumliche Kanten-Atome Wir stellen hier drei Arten neuer Atome bereit:

1. *Virtuelle* RCC-Atome,
2. Δ -Atome, und
3. ϵ -Atome.

Virtuelle RCC-Atome werden benötigt, wenn das Substrat keine expliziten RCC-Kanten enthält. Als RCC-Relationen sind alle diskutierten RCC-Basis-Relationen und beliebige Disjunktionen dieser zulässig: so ist `(?x ?y (:ntppi :tppi))` ein virtuelles RCC-Atom. Es ist rein syntaktisch nicht von einem *einfachen Atom* zu unterscheiden (s.u.).

Zudem halten wir die Δ - und ϵ -Atome im GIS-Kontext für relevant, s. Abb. 5.17. Diese Atome haben folgende Semantik:

Δ -Atom: „In einer Entfernung von $[min; max[$ “ – diese Relation gilt zwischen zwei Objekten a und b , wenn die Distanz zwischen a und b im Intervall $[min; max[$ liegt. Die Entfernung wird zwischen den Schwerpunkten von a und b berechnet.

Letztlich bedeutet dies, dass der Schwerpunkt von b in einem Ring der Breite $max - min$ mit Radius $\frac{max+min}{2}$ um den Schwerpunkt von a liegen muss. Die Relation ist in Abb. 5.23a) skizziert.

ϵ -Atom: „In einer Epsilon-Umgebung von $[min; max[$ enthalten“ – diese Relation gilt zwischen zwei Objekten a und b , wenn b *innerhalb* der max -Epsilon-Umgebung und *außerhalb* der min -Epsilon-Umgebung von a liegt.

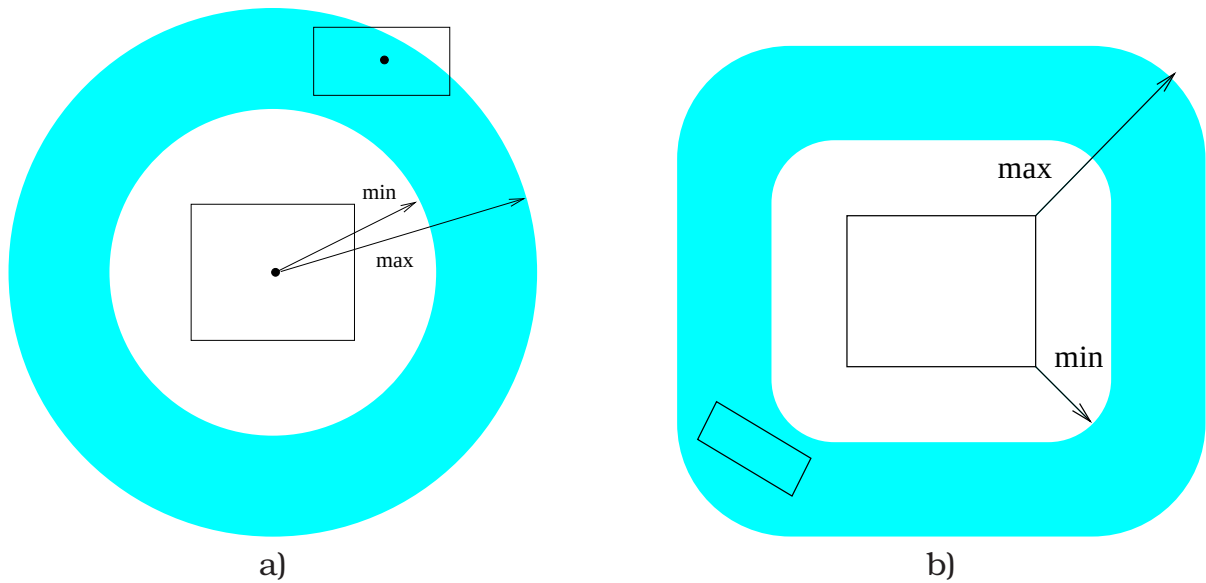


Abbildung 5.23.: Illustration der $\Delta(\min, \max)$ und $\epsilon(\min, \max)$ -Relationen.

Letztlich bedeutet dies, dass das Objekt b in einem Epsilon-Ring der Breite $\max - \min$ mit Radius $\frac{\max + \min}{2}$ um a herum liegen muss. Die Relation ist in Abb. 5.23b) skizziert.

Die Extension des Atoms `(?x ?y (:ring-epsilon 100 200))` ist also durch die Menge aller Paare (i, j) gegeben, sodass sich j in einem ϵ -Ring der Abmessung $[100; 200[$ um i herum befindet.

Für viele GIS-Analysetätigkeiten werden sog. *Puffer-Zonen* (*Buffer Zones*) und *Umkreissuchen* (*Range Queries*) benötigt. Die Δ - und ϵ -Atome stellen diese Funktionalität im DLMAPS-System bereit. Beide Relationen lassen unspezifiziertes $\min$ oder $\max$ zu. Dementsprechend nennen wir die Δ -Relationen dann

- `:inside-distance` für Intervalle $[0; \max[$,
- `:outside-distance` für Intervalle $[\min; \infty[$, und
- `:ring-distance` für Intervalle $[\min; \max[$,

und die ϵ -Relationen

- `:inside-epsilon` für Intervalle $[0; \max[$,
- `:outside-epsilon` für Intervalle $[\min; \infty[$, und
- `:ring-epsilon` für Intervalle $[\min; \max[$.

Die Relationen können aufgrund der benötigten Intervalle nicht mehr qualitativ genannt werden. Es gibt also eine überabzählbare Anzahl dieser Relationen, und sie haben natürlich auch nicht die JEPD-Eigenschaft. Daher erscheint

5.3 Vier konkrete Substrat-Anfragesprachen

auch ein Vorausberechnen und Explizieren mittels expliziter Kanten nicht sinnvoll.

5.3.4.4. Konkrete Syntax der DISKQL am Beispiel der DISK-Referenzanfragen

Zur Illustration der konkreten Syntax der Anfragesprache geben wir nun die DISK-Referenzanfragen **Q1** bis **Q7** (s. Kap. 3.1.3.3, S. 147) in konkreter Syntax an.

Wir geben die Anfragen in der DISKQL-*Endnutzersyntax* an. Wir erläutern weiter unten, wie die Endnutzersyntax auf (u.U. hybride) DISKQL-Anfragen abgebildet wird. Die Art der Abbildung hängt vom verwendeten Karten-Substrat ab.

Die Köpfe der Anfragen sind eigentlich irrelevant. Natürlich können hier mal mehr, mal weniger Variablen enthalten sein, wie gewünscht. Wesentlich ist jedoch, dass mittels *tupelwertiger* DISKQL-Anfrageergebnisse stets *Konstellationen* von Karten-Objekten erhalten werden können, die in den durch die Anfrage geforderten räumlich-thematischen Beziehungen zueinander stehen, und dass diese Ergebnis-Konstellationen auch adäquat in der Karte visualisiert werden können (s. Kap. 3.1.6).

Q1 „Zeige mir alle Wohngebiete!“

```
(dlmaps-retrieve (?x) (?x wohngebiet))
```

Q2 „Zeige mir alle Wohngebiete, die an einer Autobahn liegen!“

```
(dlmaps-retrieve (?x ?y)
  (and (?x wohngebiet) (?x ?y :adjacent) (?y autobahn)))
```

Q3 „Zeige mir alle Wohngebiete, die an einer Grünfläche liegen, welche einen Teich oder See enthält! Das Wohngebiet soll zudem an einem Gewerbegebiet liegen (zum Einkaufen).“

```
(dlmaps-retrieve (?w)
  (and (?w wohngebiet) (?g gruenflaeche)
    (?e gewerbegebiet) (?t (or teich see))
    (?w ?g :adjacent) (?w ?e :adjacent) (?g ?t :contains)))
```

Q4 „Zeige mir alle Wohngebiete, die mindestens 8 Gebäude enthalten, und deren Fläche größer als 30000 qm ist!“

Für diese Anfrage benötigen wir zwingenderweise ein BAA-Konzept-Atom (*at-least 8 ppi gebaeude*) und somit ein Karten-Substrat „mit ABox“:²⁴

²⁴Die Rolle PPI ist eine gemeinsame Oberrolle der Rollen NTTPI und TPPI. Für die verwendete ABox muss zudem die UNA verwendet werden, da ansonsten (*at-least 8 ppi gebaeude*) nicht abgeleitet werden kann.

```
(dlmaps-retrieve (?x)
  (and (?x (and wohngebiet (at-least 8 ppi gebaeude)))
    (?x (:area (> area 30000))))))
```

Q5 „Zeige mir alle Wohngebiete, die NICHT an Industrie- oder Gewerbegebiete angrenzen!“

Wir hatten in Kap. 3.1.3.3, S. 147 zwei Semantiken für „nicht“ diskutiert:

1. Interpretieren wir „nicht“ als „nicht bekannt“, so können wir die NAF-Negation verwenden:

```
(dlmaps-retrieve (?x)
  (and (?x wohnen)
    (neg (project-to (?x)
      (and (?x ?y :adjacent)
        (?y (or industrie gewerbe)))))))
```

(Q5, Variante 1).

2. Andernfalls können wir „nicht“ als „beweisbar nicht“ interpretieren, und somit die klassische Negation verwenden:

```
(dlmaps-retrieve (?x)
  (and (?x (and wohnen
    (all ec (not (or industrie gewerbe)))))))
```

(Q5, Variante 2).

Q6 „Welche Gewässer in Naturschutzgebieten werden von Bächen gespeist, die durch oder an Gewerbegebieten mit Fabriken vorbei fließen?“

```
(dlmaps-retrieve (?x ?y ?z ?u ?f)
  (and (?x gewaesser)
    (?x ?y :inside)
    (?y naturschutzgebiet)
    (?z ?x :flows-in)
    (?z bach)
    (?z ?u (:or :touches :crosses))
    (?u gewerbegebiet)
    (?u ?f :contains)
    (?f fabrik)))
```

Q7 „Welche Wohngebiete enthalten in einer Umgebung von 750 m. Radius eine UBahn-Station?“

```
(dlmaps-retrieve (?x ?y)
  (and (?x wohngebiet)
    (?y ?x (:inside-distance 750))
    (?y ubahn-station)))
```

5.3 Vier konkrete Substrat-Anfragesprachen

5.3.4.5. Ein Algorithmus zur Entscheidung von Subsumption zwischen eingeschränkten DISKQL-Anfragen

Im DLMAPS-System wird die QBox für *eingeschränkte* DISKQL-Anfragen genutzt. Der Algorithmus wurde in [Wessel 2003b] vorgestellt.

Wie auch im Subsumptionsalgorithmus für die BAA-Anfragen, s. Kap. 5.3.2.1.3, S. 325, müssen eingeschränkte DISKQL-Anfragen in Normalform sein, und dürfen zudem keine NAF- und keine Referenz-Atome enthalten. Eine weitere Einschränkung kommt hinzu, um Vollständigkeit zu gewährleisten:

Definition 100 (Eingeschränkte DISKQL-Anfrage) Eine eingeschränkte DISKQL-Anfrage ist eine DISKQL-Anfrage, die

1. in Normalform ist, keine NAF- und Referenz-Atome enthält, und
2. ausschließlich BAA-Atome und RCC-Atome enthält. Wir können o.B.d.A. annehmen, dass diese stets die Form $(a, b, R_1 \vee \dots \vee R_n)$ mit $R_i \in \text{RCC8}$ haben.

BAA-Atome dürfen zudem *ausschließlich* zur Adressierung thematischer Aspekte verwendet werden, und zur Adressierung räumlicher Aspekte dürfen *ausschließlich* RCC-Atome verwendet werden. ■

Eine solche Anfrage kann, da ausschließlich BAA-Atome zur Adressierung der thematischen Aspekte verwendet werden dürfen, lediglich an ein Karten-Substrat mit assoziierter ABox gestellt werden. Räumlichen Aspekte sind zudem nicht in der ABox repräsentiert, da BAA-Atome ausschließlich thematische Aspekte adressieren dürfen. Der Algorithmus kann nicht für Anfragen mit Δ - oder ϵ -Atomen verwendet werden. Die RCC-Atome stellen jedoch die am häufigsten verwendete Klasse von räumlichen Atomen dar, wie anhand der Beispielanfragen zu sehen. Es ist sicherlich möglich, entsprechende Subsumptionsalgorithmen für andere (größere Klassen von) DISKQL-Anfragen zu entwickeln.

Unter Verwendung eines solchen Karten-Substrates wird dann die Anfrage **Q3** in Endnutzersyntax

```
(dlmaps-retrieve (?w)
  (and (?w wohngebiet) (?g gruenflaeche)
    (?e gewerbegebiet) (?t (or teich see))
    (?w ?g :adjacent) (?w ?e :adjacent) (?g ?t :contains)))
```

in eine hybride, vom Rahmenwerk verarbeitbare DLMAPS-Anfrage

```
(dlmaps-retrieve (?*w)
  (and (?*w wohngebiet) (?*g gruenflaeche)
    (?*e gewerbegebiet) (?*t (or teich see))
    (?w ?g EC) (?w ?e EC) (?g ?t (:OR NTPPI TPPI))))
```

umgeschrieben. Hier sind nun ($?*w$ wohngebiet), ($?*g$ gruenflaeche), als auch ($?*e$ gewerbegebiet) und ($?*t$ (or teich see)) BAA-Atome für S_2 ; die anderen Atome sind virtuelle RCC-Atome für S_1 .

Ein Beispiel für eine unzulässige Anfrage ist durch **94** gegeben, da das Konzept-Atom ($?x$ (and wohngebiet (at-least 8 ppi gebaeude))) einen räumlichen Aspekt adressiert. Die Definition verlangt jedoch, dass BAA-Atome *ausschließlich* zur Adressierung thematischer Aspekte verwendet werden.

Aufgrund der strikten Trennung thematischer und räumlicher Aspekte in eingeschränkten DISKQL-Anfragen kann ein einfacher Subsumptionsalgorithmus angegeben werden. Die Idee ist einfach: für den *thematischen Anteil* einer Anfrage, der ja ausschließlich durch BAA-Atome gegeben ist, wird der BAA-Subsumptionsalgorithmus verwendet (s. S. 325), und für den *räumlichen Anteil* entscheiden wir *Subsumption zwischen RCC-Netzen* (s.u.). Da räumliche und thematische Anteile der eingeschränkten Anfragen nicht interagieren, ist die Vollständigkeit des kombinierten Subsumptionsalgorithmus sichergestellt.

Für eine eingeschränkte DISKQL-Anfrage seien die Funktionen R und T definiert. Dabei liefert R den räumlichen Anteil einer solchen Anfrage, und T den thematischen Anteil. Offensichtlich erhalten wir für eine eingeschränkte DISKQL-Anfrage mit Rumpf k durch $R(k)$ stets eine Konjunktion von RCC-Atomen, und durch $T(k)$ eine BAA-Anfrage. Betrachten wir die Beispiel-Anfrage

$$k = (w, g, EC) \wedge (w, e, EC) \wedge (g, t, NTPPI \vee TPPI) \wedge$$

$$(*w, wohngebiet) \wedge (*g, gruenflaeche) \wedge (*e, gewerbegebiet) \wedge (*t, teich \sqcup see).$$

so ist

$$R(k) = (w, g, EC) \wedge (w, e, EC) \wedge (g, t, NTPPI \vee TPPI),$$

und

$$T(k) = (*w, wohngebiet) \wedge (*g, gruenflaeche) \wedge (*e, gewerbegebiet) \wedge (*t, teich \sqcup see);$$

die Funktionen geben jeweils $\top$ zurück für leere Konjunktionen. Wir sparen uns eine formale Definition von T und R .

Es ist offensichtlich, dass wir $R(k)$ als (prädikatenlogisch notiertes) RCC-Netz $EC(w, g) \wedge EC(w, e) \wedge (NTPPI(g, t) \vee TPPI(g, t))$ und daher als *RCC-Substrat* (s. Def. 62, S. 197) verstehen können. *Subsumption zwischen RCC-Netzen* bzw. *RCC-Substraten* ist nun wie folgt definiert:

5.3 Vier konkrete Substrat-Anfragesprachen

Definition 101 (Subsumption zwischen RCC-Netzen gleicher Knotenanzahl)

Seien S_1 und S_2 Instanzen der Klasse RCC8-Substrat (RCC8-Substrate) mit $|V_1| = |V_2|$ (also ist $S_i = (V_i, E_i, \emptyset, L_{E_i}, \emptyset, 2^{\text{RCC8}})$).

Wir sagen, dass S_1 von S_2 subsumiert wird, in Zeichen $S_1 \models S_2$ oder $S_1 \sqsubseteq S_2$, gdw. es eine Bijektion Π von V_2 auf V_1 gibt, sodass für alle $R \in 2^{\text{RCC8}}$ und für alle $i, j \in V_2 \times V_2$ aus $S_2 \models (i, j, R)$ stets $S_1 \models (\Pi(i), \Pi(j), R)$ gilt. ■

Gilt $S_1 \models S_2$, so gilt jede in S_2 geltende RCC-Relation (i, j, R) auch in S_1 . Dies bedeutet, dass die in S_1 zwischen $\Pi(i)$ und $\Pi(j)$ geltende RCC-Relation auch *spezieller* als die in S_2 geltende Relation (i, j, R) sein kann. Es gilt daher u.U. sogar $(\Pi(i), \Pi(j), R') \models (\Pi(i), \Pi(j), R)$ für $R' \subseteq R$ und daher $(\Pi(i), \Pi(j), R') \models (\Pi(i), \Pi(j), R)$ (man beachte, dass R, R' Disjunktionen sind).

Haben die Knotenmengen nun jedoch verschiedene Kardinalitäten, $|V_1| \neq |V_2|$, so kann natürlich keine Bijektion gefunden werden. Durch Hinzufügen neuer Knoten zu V_1 (im Fall $|V_1| < |V_2|$) bzw. zu V_2 (im Fall $|V_1| > |V_2|$) können wir $|V_1| = |V_2|$ sicherstellen. Aufgrund der JEPD-Eigenschaft wissen wir, dass eine nichtexistierende Kante äquivalent zu einer mit der Disjunktion aller Basis-Relationen beschrifteten Kante ist (der RCC8-Relation). Wir können dann auch Subsumption zwischen beliebigen RCC-Netzen bzw. -Substraten wie folgt definieren:

Definition 102 (Subsumption zwischen beliebigen RCC-Netzen) Seien S_1 und S_2 Instanzen der Klasse RCC8-Substrat (RCC8-Substrate).

Wir haben 3 Fälle zu betrachten:

Falls $|V_1| < |V_2|$, so definieren wir $S'_1 = (V'_1, E'_1, \emptyset, L'_{E_1}, \emptyset, 2^{\text{RCC8}})$ mit

- $V'_1 =_{\text{def}} V_1 \cup \{ \boxed{i} \mid i \in 1 \dots (|V_2| - |V_1|) \},$
- $E'_1 =_{\text{def}} E_1 \cup \{ (\boxed{i}, \boxed{j}) \mid i, j \in 1 \dots (|V_2| - |V_1|) \},$
- $L'_{E_1} =_{\text{def}} L_{E_1} \cup \{ ((\boxed{i}, \boxed{j}), \text{RCC8}) \mid i, j \in 1 \dots (|V_2| - |V_1|) \} \cup \{ ((\boxed{i}, \boxed{i}), \text{EQ}) \mid i \in 1 \dots (|V_2| - |V_1|) \}.$

Nun gelte $S_1 \models S_2$ gdw. $S'_1 \models S_2$.

Falls $|V_1| > |V_2|$, so definieren wir $S'_2 = (V'_2, E'_2, \emptyset, L'_{E_2}, \emptyset, 2^{\text{RCC8}})$ analog zu S'_1 .

Nun gelte $S_1 \models S_2$ gdw. $S_1 \models S'_2$.

Andernfalls gilt $|V_1| = |V_2|$, und $S_1 \models S_2$ ist bereits definiert. ■

Wir werden Subsumption zwischen RCC-Netzen durch Reduktion auf eine Reihe von RCC8-Unerfüllbarkeitstests entscheiden:

Lemma 7 (Folgerbarkeit für RCC-Atome) Sei $S = (V, E, \emptyset, L_E, \emptyset, 2^{\text{RCC8}})$ ein RCC8-Substrat (RCC8-Substrate), $R \in 2^{\text{RCC8}}$, und $i, j \in V \times V$.

Sei $S' = (V, E', \emptyset, L'_E, \emptyset, 2^{\text{RCC8}})$ mit $E' =_{\text{def}} E \cup \{(i, j)\}$ und $L'_E =_{\text{def}} L_E \cup \{((i, j), R')\}$, wobei $R' = (\text{RCC8} \setminus R) \cap L_E((i, j))$.

Dann gilt $S \models (i, j, R)$ gdw. S' inkonsistent ist (s. Faktum 7, S. 141). ■

Der benötigte Konsistenztest wird von der Klasse RCC8-Substrat (RCC8-Substrate) durch die Methode `consistent-p` bereitgestellt.

Zunächst benötigen wir eine Hilfsfunktion (deren Zweck gleich deutlich werden wird), die *Paare von korrespondierenden Objekten* einer hybriden eingeschränkten DISKQL-Anfrage bildet:

Definition 103 (Funktion objekt_paare) Sei k der Rumpf einer eingeschränkten DISKQL-Anfrage. Die Funktion `objekt_paare` ist dann wie folgt definiert:

$$\text{objekt\_paare}(k) =_{\text{def}} \{ (a, \ast(a)) \mid a \in \text{objekte}(\text{R}(k)) \} \cup \{ (\ast(a), a) \mid a \in \text{objekte}(\text{T}(k)) \},$$

wobei R und T die erwähnten *räumlichen* und *thematischen* Anteile einer Anfrage liefern, und $\ast$ wie folgt definiert ist:

$$\ast(x) = \begin{cases} x' & \text{falls } x' = \ast x, \quad x \in \text{objekte}(\text{R}(k)), \quad x' \in \text{objekte}(\text{T}(k)) \\ x' & \text{falls } x = \ast x', \quad x \in \text{objekte}(\text{T}(k)), \quad x' \in \text{objekte}(\text{R}(k)) \\ \perp & \text{sonst.} \end{cases}$$

Zudem setzen wir

$$\text{var\_paare}(k) =_{\text{def}} \text{objekt\_paare}(k) \cap (\text{Vars} \times \text{Vars})$$

und

$$\text{ind\_paare}(k) =_{\text{def}} \text{objekt\_paare}(k) \setminus \text{var\_paare}(k).$$

■

Beispielsweise gilt für $k = (\ast x, C) \wedge (\ast y, D) \wedge (x, z, EC)$ zunächst $\text{R}(k) = (x, z, EC)$, $\text{objekte}(\text{R}(k)) = \{x, z\}$, und $\text{T}(k) = (\ast x, C) \wedge (\ast y, D)$, $\text{objekte}(\text{T}(k)) = \{\ast x, \ast y\}$. Dann ist $\text{objekt\_paare}(k) = \{(x, \ast(x)), (z, \ast(z))\} \cup \{(\ast(\ast x), \ast x), (\ast(\ast y), \ast y)\} = \{(x, \ast x), (z, \perp)\} \cup \{(x, \ast x), (\perp, \ast y)\}$. Daher ist $\text{objekt\_paare}(k) = \{(x, \ast x), (z, \perp), (\perp, \ast y)\}$.

Wie auch im BAA-Subsumptionsalgorithmus aus Kap. 5.3.2.1.3, S. 325, muss zur Entscheidung von Subsumption zwischen zwei Rümpfen k_1 und k_2 aufgrund potentiell verschiedener Variablennamen in k_1 und k_2 nun eine Permutation Π geraten werden, sodass k_2 -Variablen in k_1 -Variablen umbenannt werden. Natürlich müssen Variablen durch Π nun $\ast$ -konsistent umbenannt werden,

5.3 Vier konkrete Substrat-Anfragesprachen

d.h., gilt $\Pi(x) = z$, so muss auch $\Pi(*x) = *z$ gelten. Wir definieren Π daher als eine Bijektion von $\text{var\_paare}(k_2)$ nach $\text{var\_paare}(k_1)$, wodurch $*$ -konsistentes Umbenennen automatisch sichergestellt wird. Wie auch im BAA-Algorithmus ist Π für Individuenpaare nicht definiert, sondern lediglich für Variablenpaare.

Fordert nun eine hybride Anfrage mit Rumpf k_1 ein Paar korrespondierender Objekte, $(x, *x) \in \text{var\_paare}(k_1)$, so soll intuitiv nur dann $k_2 \sqsubseteq k_1$ für einen Rumpf k_2 gelten können, wenn auch in k_2 ein entsprechendes Paar korrespondierender Objekte gefordert wird, also ebenfalls $(y, *y) \in \text{var\_paare}(k_2)$ für $\Pi((x, *x)) = (y, *y)$ gilt (die Existenz korrespondierender Knoten in hybriden Substraten ist ja nicht automatisch gewährleistet, da die $*$ -Funktion *partiell* ist). Analog fordern wir, dass für $(i, *i) \in \text{ind\_paare}(k_1)$ auch $(i, *i) \in \text{ind\_paare}(k_2)$ gelten muss.

Eine Verallgemeinerung der Idee, dass die Objekt-Existenzforderungen der *subsumierten Anfrage* k_2 mindestens genau so stark wie die Objekt-Existenzforderungen der *subsumierenden Anfrage* k_1 sein müssen, führt zur Definition der $\rightsquigarrow$ -Relation:

Definition 104 (*-konsistente Objekt-Paare) Seien (a, b) und (a', b') zwei Objekt-Paare. Ist die Formel

$$\begin{aligned} & (a \neq \perp \rightarrow a' \neq \perp) \wedge \\ & (b \neq \perp \rightarrow b' \neq \perp) \wedge \\ & ((a \notin \text{Vars} \wedge a \neq \perp) \rightarrow a' = a) \wedge \\ & ((b \notin \text{Vars} \wedge b \neq \perp) \rightarrow b' = b) \end{aligned}$$

wahr, so heißen (a, b) und (a', b') **-konsistent*, und wir schreiben dann $(a, b) \rightsquigarrow (a', b')$. ■

Dementsprechend definieren wir:

Definition 105 (*-konsistente Permutation) Eine Permutation $\Pi : \text{var\_paare}(k_2) \rightarrow \text{var\_paare}(k_1)$ heißt **-konsistent*, wenn $\forall (a, b) \in \text{var\_paare}(k_2). (a, b) \rightsquigarrow \Pi((a, b))$ gilt. ■

Nach diesen Vorarbeiten können wir den Algorithmus zur Anfragesubsumptionsbestimmung aus Kap. 5.3.2.1.3, S. 325 für eingeschränkte DISKQL-Anfragen wie folgt anpassen und erweitern:

1. Eingabe: Zwei eingeschränkte DISKQL-Anfragen k_1 und k_2 .
2. Ausgabe: Gilt $k_2 \sqsubseteq k_1$: TRUE, FALSE, UNKNOWN.
3. Wenn $|\text{var\_paare}(k_1)| \neq |\text{var\_paare}(k_2)|$, gib FALSE zurück.
4. Ist $\forall (i, j) \in \text{ind\_paare}(k_2). \exists (k, l) \in \text{ind\_paare}(k_1). (i, j) \rightsquigarrow (k, l)$ falsch, so gib FALSE zurück.

5. Solange es noch eine nichtprobierte, *-konsistente Permutation Π von $\text{var\_paare}(k_2)$ nach $\text{var\_paare}(k_1)$ gibt:

- Berechne $\text{DNF}(\Pi(k_2) \wedge \setminus k_1) = d_1 \vee \dots \vee d_l$.
- Prüfe, ob
 - $\text{SAT}(\text{T}(d_i)) = \text{FALSE}$ für alle $i \in 1 \dots l$ mit $\text{T}(d_i) \neq \top$ gilt, und ob
 - $\text{RCC\_SAT}(\text{R}(d_i)) = \text{FALSE}$ für alle $i \in 1 \dots l$ mit $\text{R}(d_i) \neq \top$ gilt.

Wenn Ja, dann gib TRUE zurück.²⁵

- Gilt $\text{SAT}(\text{T}(d_i)) = \text{UNKNOWN}$ für ein $i \in 1 \dots l$ mit $\text{T}(d_i) \neq \top$, so gib UNKNOWN zurück.

6. Gib FALSE zurück.

Die Funktion RCC_SAT testet eine Konjunktion $k = d_1 \wedge \dots \wedge d_n$ von RCC-Atomen auf Erfüllbarkeit, indem hieraus auf offensichtliche Art und Weise ein RCC-Substrat konstruiert und dieses auf relationale Konsistenz (s. Def. 53) geprüft wird. NAF-negierte RCC-Atome der Art $d_i = \setminus(x, y, R)$ werden durch $(x, y, \bigvee_{R_i \in \text{RCC8} \setminus \{R\}} R_i)$ -Atome ersetzt. Die Funktion SAT ist der normale BAA-Erfüllbarkeitstester (s. Kap. 5.3.2.1.3, S. 325).

5.3.4.5.1. Ein Beispiel zur DISKQL-Anfragensubsumption Betrachten wir die beiden eingeschränkten DISKQL-Anfragen

$$k_1 = (x, y, EC) \wedge (y, z, PPI) \wedge \\ (*x, \text{wohngebiet}) \wedge (*y, \text{gruenflaeche}) \wedge (*z, \text{wasserflaeche})$$

und

$$k_2 = (c, a, EC) \wedge (c, b, EC) \wedge (a, b, TPP) \wedge \\ (*c, \text{wohngebiet}) \wedge (*a, \text{teich}) \wedge (*b, \text{naturschutzgebiet})$$

mit $\text{inds}(k_1) = \emptyset$, $\text{inds}(k_2) = \emptyset$, so gilt $k_2 \sqsubseteq k_1$, denn für die Permutation

$$\Pi = \{((c, *c), (x, *x)), ((a, *a), (z, *z)), ((b, *b), (y, *y))\}$$

gilt, dass

$$(x, z, EC) \wedge (x, y, EC) \wedge (z, y, TPP) \wedge \\ (*x, \text{wohngebiet}) \wedge (*z, \text{teich}) \wedge (*y, \text{naturschutzgebiet}) \wedge \\ \setminus((x, y, EC) \wedge (y, z, PPI) \wedge \\ (*x, \text{wohngebiet}) \wedge (*y, \text{gruenflaeche}) \wedge (*z, \text{wasserflaeche}))$$

unerfüllbar ist, denn die einzelnen Disjunkte sind unerfüllbar:

- Das Disjunkt

²⁵Anmerkung: Die Funktionen R und T liefern $\top$ für leere Konjunktionen.

5.3 Vier konkrete Substrat-Anfragesprachen

$$(x, z, EC) \wedge (x, y, EC) \wedge (z, y, TPP) \wedge \\ (*x, \text{wohngebiet}) \wedge (*z, \text{teich}) \wedge (*y, \text{naturschutzgebiet}) \wedge \\ \neg(x, y, EC)$$

ist unerfüllbar, weil $\text{RCC\_SAT}((x, y, EC) \wedge \neg(x, y, EC)) = \text{FALSE}$, denn
 $(x, y, \{EC\}) \wedge (x, y, \{EQ, PO, DC, NTPP, TPP, NTPPI, TPPI\})$

ist unerfüllbar, weil $\{EC\} \cap \{EQ, PO, DC, NTPP, TPP, NTPPI, TPPI\} = \emptyset$.

- Das Disjunkt

$$(x, z, EC) \wedge (x, y, EC) \wedge (z, y, TPP) \wedge \\ (*x, \text{wohngebiet}) \wedge (*z, \text{teich}) \wedge (*y, \text{naturschutzgebiet}) \wedge \\ \neg(y, z, PPI)$$

ist unerfüllbar, weil $\text{RCC\_SAT}((z, y, \{TPP\}) \wedge \neg(z, y, PP)) = \text{FALSE}$, denn
 $(z, y, \{TPP\}) \wedge (z, y, \{EQ, PO, DC, NTPPI, TPPI\})$

ist unerfüllbar, weil $\{TPP\} \cap \{EC, EQ, PO, DC, NTPPI, TPPI\} = \emptyset$.

- Das Disjunkt

$$(x, z, EC) \wedge (x, y, EC) \wedge (z, y, TPP) \wedge \\ (*x, \text{wohngebiet}) \wedge (*z, \text{teich}) \wedge (*y, \text{naturschutzgebiet}) \wedge \\ \neg(*x, \text{wohngebiet})$$

ist unerfüllbar, weil $\text{SAT}((*x, \text{wohngebiet}) \wedge \neg(*x, \text{wohngebiet})) = \text{FALSE}$.

- Das Disjunkt

$$(x, z, EC) \wedge (x, y, EC) \wedge (z, y, TPP) \wedge \\ (*x, \text{wohngebiet}) \wedge (*z, \text{teich}) \wedge (*y, \text{naturschutzgebiet}) \wedge \\ \neg(*y, \text{gruenflaeche})$$

ist unerfüllbar, weil $\text{SAT}((*y, \text{naturschutzgebiet}) \wedge \neg(*y, \text{gruenflaeche})) = \text{FALSE}$,
da in der DISK-Ontologie $\text{naturschutzgebiet} \sqsubseteq \text{gruenflaeche}$ gilt.

- Das Disjunkt

$$(x, z, EC) \wedge (x, y, EC) \wedge (z, y, TPP) \wedge \\ (*x, \text{wohngebiet}) \wedge (*z, \text{teich}) \wedge (*y, \text{naturschutzgebiet}) \wedge \\ \neg(*z, \text{wasserflaeche})$$

ist unerfüllbar, weil $\text{SAT}((*z, \text{teich}) \wedge \neg(*z, \text{wasserflaeche})) = \text{FALSE}$, da auf-
grund der DISK-Ontologie $\text{teich} \sqsubseteq \text{wasserflaeche}$ gilt.

5.3.4.6. Implementation des DLMAPS-Systems

Das DLMAPS-System ist durch die verschiedenen Karten-Substrate, den MAP VIEWER und die DISKQL-Anfragesprache gegeben. Mit dem MAP VIEWER kann eine SQD-Datei eingelesen und aus dieser ein entspr. Karten-Substrat erzeugt werden. Die Klasse des zu erzeugenden Karten-Substrates wird im MAP VIEW-

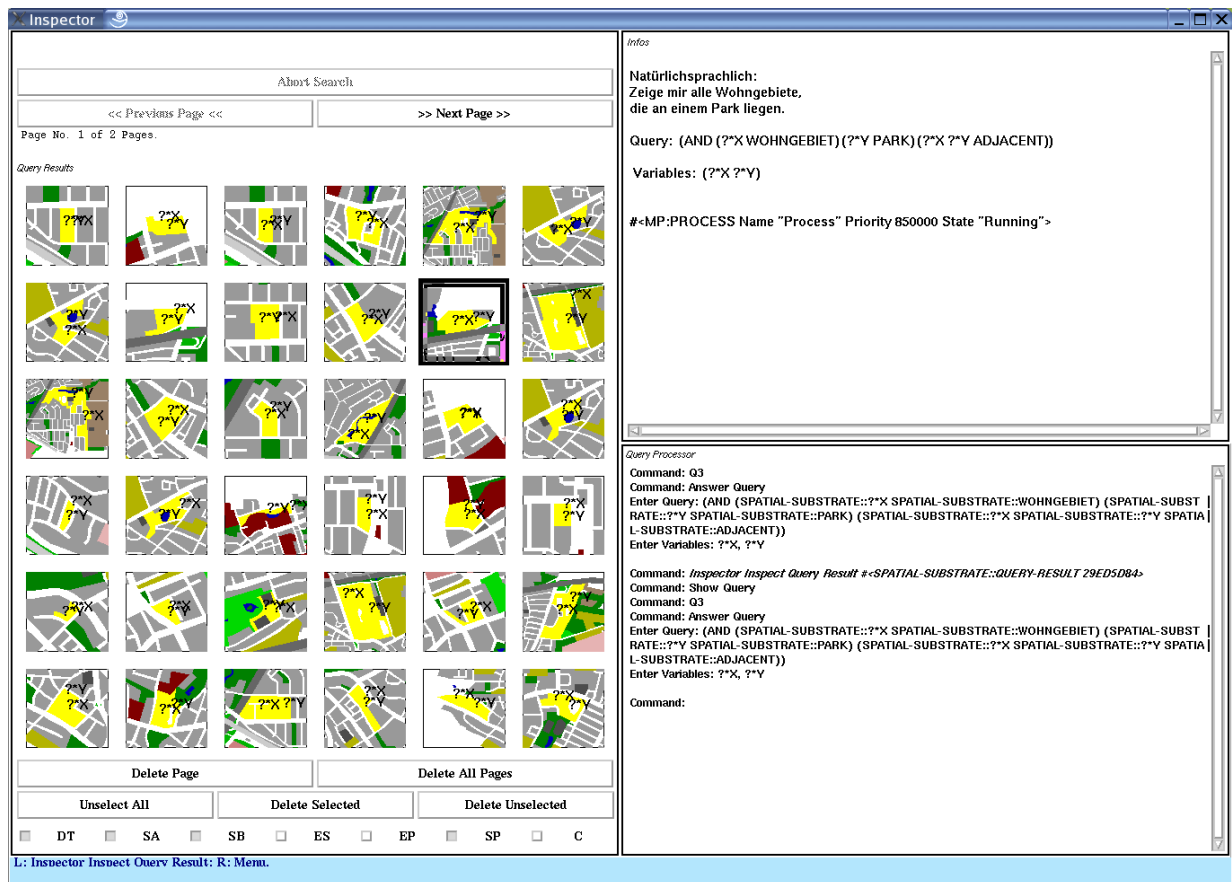


Abbildung 5.24.: Der QUERY INSPECTOR.

ER durch ein Menü bestimmt. Ein erzeugtes Karten-Substrat kann im MAP VIEWER visualisiert und grafisch-interaktiv inspiziert werden. Der sog. QUERY INSPECTOR (s. Abb. 5.24) ist eine weitere Anwendung zum Absetzen von DISKQL-Anfragen und zur Präsentation und -inspektion von Anfrageergebnissen. Der QUERY INSPECTOR vervollständigt das DLMAPS-System.

Wir schildern folgende implementationsspezifische Punkte in aller Kürze:

1. Implementation der DISKQL-Endnutzersyntax,
2. Bereitstellung der räumlichen Atome,
3. Anpassung der Bewertungsmethoden,
4. Vorstellung der QUERY INSPECTOR-Anwendung.

5.3.4.6.1. Bereitstellung der DISKQL-Endnutzersyntax Wir nutzen die vom Rahmenwerk bereitgestellte Funktionalität zum syntaktischen Umschreiben (Methoden syntactically-rewrite-query, rewrite-complex-query-body,

5.3 Vier konkrete Substrat-Anfragesprachen

rewrite-query-atom). Zudem variieren wir die *vom Parser vorgenommene Klassifikation eines Atoms* in Abhängigkeit (der Klasse, des Typs) des befragten Karten-Substrates. Wird nun beispielsweise ein Atom ($?x$ wohngebiet) in einer Anfrage verwendet, so wird die Methode *get-expression-type* je nach befragtem Substrat einen der folgenden drei Klassennamen zurückgeben (in dieser Reihenfolge):

- RACERPRO-Beschreibungs-Atom (RACERPRO-Description-Query), falls das Substrat (direkte oder indirekte) Instanz der Klasse RACERPRO-DL-Beschreibungs-Substrat (RACERPRO-DL-Description-Substrate) ist.
- Konzept-Atom (Concept-Query), falls das Substrat (direkte oder indirekte) Instanz einer der folgenden Klassen ist: RACERPRO-ABox-Adapter-Spiegel-Substrat (RACERPRO-ABox-Adapter-Mirror-Substrate), ABox, oder Hybrides-Karten-Substrat (Hybrid-Map-Substrate) mit S_2 vom Typ RACERPRO-BAA-Substrat (RACERPRO-BAA-Substrate). Im letzten Fall wird ($?*x$ wohngebiet) zurückgegeben, in den anderen Fällen jedoch ($?x$ wohngebiet).
- Einfaches-Beschreibungs-Atom (Simple-Description-Query) andernfalls.

Natürlich kann diese Automatik durch entspr. Syntax stets „überschrieben“ werden.

Bei der Initialisierung des Atoms wird die Anfragebedingung (das Wort aus $\mathcal{L}_{QV}$) entsprechend aufbereitet; die Atome sind ja selbstübersetzend, s. S. 239. Tatsächlich kann hierbei auch $?x$ gegen $?*x$ und andersrum ersetzt werden. Das Atom „weiß“ daher, ob der angefragte räumliche oder thematische Aspekt in einem evtl. hybriden Karten-Substrat nun in S_1 oder in S_2 repräsentiert ist.

Genauso wird auch für die RCC-Atome vorgegangen. Ein RCC-Atom hat aus Endnutzerperspektive stets die gleiche Syntax, z.B. ($?x ?y (:ntppi :tppi)$). In Abhängigkeit vom vorliegenden Karten-Substrat muss für dieses nun entweder ein *einfaches Atom* oder ein *virtuelles RCC-Atom* erzeugt werden. Letzteres ist der Fall, wenn das Substrat keine expliziten RCC-Kanten enthält. Zudem können in einer evtl. vorhandenen (assoziierten) ABox RCC-Kanten in Form von RCC-Rollen-Assertionen vorhanden sein, sodass prinzipiell sogar ein BAA-Rollenatom für das RCC-Atom eingesetzt werden könnte. In der Regel wird jedoch versucht, RCC-Atome anhand von S_1 auszuwerten (da S_1 ein geometrisches Substrat ist).

Die *natürlichsprachlichen Präpositionen* werden bereits vor dem Erzeugen der ASG-Knoten mittels *syntactically-rewrite-query* auf RCC-Relationen umgeschrieben. Die entspr. Ersetzungsregeln hatten wir oben aufgelistet; diese werden in einer Art „Wörterbuch“ gehalten, welches zum automatischen Umschreiben genutzt wird. Die Ersetzungsregeln im Wörterbuch können (ohne dass Programmierung erforderlich würde) deklarativ geändert werden.

5.3.4.6.2. Implementation der räumlichen Atome Für alle „neuen“ diskutierten räumlichen Atome wurden ASG-Klassen (Unterklassen von Unäres-Atom (Unary-Query) und Binäres-Atom (Binary-Query)) geschaffen.

Die Generatoren der virtuellen RCC-Atome sind informiert, da sie die *räumlichen Iteratoren* (s. Kap. 5.1.2.6.9, S. 269 und Kap. 5.1.4, S. 275) nutzen, welche die RCC8-Relationen „kennen“ und dementsprechend wissen, wie die von der Klasse Geometrisches-Substrat (Geometric-Substrate) bereitgestellten räumlichen Indexstrukturen z.B. zur effizienten räumlichen Selektion aller in einem Referenzpolygon enthaltener Objekte genutzt werden können (Errechnung der *PPI*-Nachfolger). Gleiches gilt auch für die Δ - und ϵ -Atome. Der räumliche Index unterstützt auch die räumliche Selektion anhand von Δ - oder ϵ -Spezifikationen.

5.3.4.6.3. Anpassung der Bewertungsmethoden für räumliche Kanten-Atome

Die getscore-Bewertungsmethoden wurden für die virtuellen RCC-, die Δ - und ϵ -Atome so angepasst, dass die Kosten einer räumlichen Selektion berücksichtigt werden. Der hier implementierte räumliche Index kann beispielsweise *PPI*- und *PO*-Iteratoren recht effizient unterstützen; teurer sind hingegen $\{TPPI, TPP, NTPP, EC, EQ, DC\}$ sowie ϵ -Atome.

5.3.4.6.4. Der QUERY INSPECTOR Speziell für Demonstrationszwecke wurde dem MAP VIEWER eine weitere grafische Oberfläche zum Absetzen vom DISKQL-Anfragen sowie zur Ergebnispräsentation und -inspektion zur Seite gestellt, der sog. QUERY INSPECTOR, s. Abb. 5.24.

Zur Visualisierung der Ergebnistupel werden Miniatur-Karten (Thumbnails) verwendet, die automatisch auf mehrere Seiten verteilt werden. Die Anfrageergebnisse können interaktiv mit der Maus selektiert und dann im MAP VIEWER detaillierter inspiziert werden; einzelne Anfrageergebnisse können auch gelöscht werden, etc. Der QUERY INSPECTOR bietet zudem umfangreiche Zusatzfunktionalität. Beispielsweise ist ein einfacher Taxonomie-Browser für die DISK-Ontologie integriert, ein QBox-Browser, ein komfortabler Anfrageneditor mit Vervollständigungsfunktionalität, usw. Zu Demonstrationszwecken können zudem eine Reihe von **Q**-Konsolenkommandos definiert werden, sodass beispielsweise die DISK-Referenzanfragen **q1** bis **q7** „ohne Tipparbeit“ verfügbar sind.

5.4. Zusammenfassung

In diesem Kapitel haben wir das formale Rahmenwerk in ein softwaretechnisches Rahmenwerk überführt. Wir haben den softwaretechnischen Entwurf und die Implementation dargestellt. Wir haben zunächst das Substrat-Datenmodell realisiert. Die implementierten Basis-Substratklassen können für die unterschiedlichsten prototypischen ontologiebasierten ISe als extensionale Komponenten dienen und erlauben u.a. die flexible Ankopplung der Dienste existierender DL-Systeme wie RACERPRO. Wir haben insbesondere Substrate für die Leitdomänen bereitgestellt, aber auch eine Reihe von Mehrzweck- bzw. „General Purpose“-Substraten „im Angebot“ des Rahmenwerks. Substrate können zudem hybrid sein.

Zur Nutzung des nun implementierten Substrat-Datenmodells für die ontologiebasierte Anfragebeantwortung haben wir die generische Substrat-Anfragebeantwortungsmaschine realisiert. Sie enthält einen heuristischen und einen semantischen Optimierer, bietet Funktionalität zur syntaktischen und semantischen Anfrageumschreibung, ist mehrbenutzerfähig, offeriert eine Lebenszyklusverwaltung von Anfragen, und bietet u.a. inkrementelle, mengenbasierte, unvollständige und zweiphasige Anfragebearbeitung. Zudem unterstützt sie hybride Anfragesprachen.

Als konkrete Ausprägungen des Rahmenwerks haben wir insbesondere NRQL und DISKQL sowie die „zugehörigen“ prototypischen ontologiebasierten ISe OWL TREE und DLMAPS vorgestellt. Die Flexibilität, Generizität und Erweiterbarkeit des softwaretechnischen Rahmens wurde durch die diskutierten Ausprägungen belegt.

Im Laufe des Kapitels haben wir an mehreren Stellen *Problempunkte* bei der Nutzung externer DL-Systeme wie RACERPRO identifiziert (s. z.B. Kap. 5.2.2.3.3, S. 298 und Kap. 5.3.3.6.1, S. 367, die die Schaffung eines eigenen DL-Systems namens MIDELORA motivieren. Das folgende Kapitel wird daher der Realisierung von MIDELORA gewidmet sein.

6. Bereitstellung offener und flexibler DL-Beweiser im Rahmenwerk

Inhaltsangabe

6.1	Motivation und Anforderungen	401
6.1.1	Implementierung von Beweisern für Spezial-DLs	401
6.1.2	Implementierung aufgabenspezifischer Beweiser	401
6.1.3	Flexible Adaption und Modifikation existierender Beweiser	401
6.1.4	Offene und erweiterbare ABox-Repräsentation	402
6.1.5	Anforderungen aus den Leitdomänen	405
6.1.6	Tableau-Repräsentation	409
6.1.7	Inkrementelle Basis-ABox-Retrieval-Funktionen	411
6.2	Entwurf von MiDeLoRA	412
6.2.1	Der MiDeLoRA-Raum	413
6.2.2	Basis-Entwurfsideen	414
6.2.3	Potentiale der MiDeLoRA-Architektur	430
6.2.4	Benötigte Basis-Funktionalität	447
6.2.5	Struktur des MiDeLoRA-Raums	454
6.2.6	Softwaretechnische Abstraktionen	458
6.2.7	Ein Arsenal von Tableau-Regeln	466
6.2.8	Berücksichtigung von Stand-der-Kunst-Optimierungen .	479
6.3	Implementation von MiDeLoRA	488
6.3.1	Repräsentation von Mengen	488
6.3.2	Implementation der Befehlshistorie und Befehlsobjekte .	488
6.3.3	Definition von Tableau-Regeln	488
6.3.4	Beweiser und Regelanwendungsketten	494
6.3.5	Repräsentation von Modellen	496
6.4	Vorstellung einiger MiDeLoRA-Instantiierungen	499
6.4.1	ABox-Erfüllbarkeitsbeweiser für Standard-DLs	500
6.4.2	Ein Instanzenretrieval-Beweiser	509
6.4.3	Die BAA für MiDeLoRA	513
6.4.4	ABox-Erfüllbarkeitsbeweiser für DLs mit Rollenboxen .	514
6.4.5	Ein Beweiser für räumliche ABoxen	518
6.4.6	Ein Beweiser für persistenten ABoxen	520
6.5	Zusammenfassung	522

Das in den letzten beiden Kapiteln aufgespannte formale und softwaretechnische Rahmenwerk stellt eine flexible und erweiterbare Basis-Architektur zur Anknüpfung, Nutzung und Erweiterung der Repräsentations- und Inferenzdienste existierender DL-Systeme (wie RACERPRO) zur Realisierung prototypischer ontologiebasierter ISe bereit. Wir haben in dieser Arbeit die Nutzungsmöglichkeiten externer DL-Systeme (wie RACERPRO) in unseren Anwendungsszenarien bzw. Leitdomänen analysiert und dabei auch diverse (die Nutzung betreffende) kritische Punkte identifiziert. Diese Kritik- bzw. Problemfelder motivieren die Schaffung eines eigenen DL-Systems namens MIDeLoRA. MIDeLoRA steht für „Michael’s Description Logic Reasoner“. Tatsächlich handelt es sich, wie deutlich werden wird, nicht primär um ein DL-System, sondern um ein *Baukastensystem* zur Konstruktion *offener, erweiterbarer, sprach- und aufgabenspezifischer DL-Beweiser* (vgl. auch [Cunis u. a. 1991]). Da der Schwerpunkt dieser Arbeit in der extensionalen Komponente ontologiebasierter ISe angesiedelt ist, soll MIDeLoRA ABoxen bieten und somit auch die Definition von „ABox-Beweisern“ (welcher Art auch immer) ermöglichen. Natürlich wird MIDeLoRA auch Standard-Implementationen in Form von „fertigen“ DL-Beweisern für Standard-DLs und Standard-Inferenzdienste anbieten.

Unter einem *Beweiser* verstehen wir im Folgenden die *Implementierung eines Inferenzverfahrens*. Ein Beweiser kann beispielsweise einen Tableau-Kalkül zur Lösung eines Entscheidungsproblems, aber auch ein höherwertiges Inferenzproblem implementieren (s. Kap. 2.2.2.5), z.B. `concept_instances` bereitstellen. Im Zuge der Problemreduktion wird dieser Beweiser seinerseits die Dienste anderer Beweiser in Anspruch nehmen. Auch die ABF-Implementierungen der BAA können als (zum Teil strukturelle) Inferenzverfahren und somit als „Beweiser“ im obigen Sinne angesehen werden. Tableau-Kalküle spielen jedoch die Hauptrolle in MIDeLoRA.

Da heutzutage sogar kommerzielle Systeme (wie RACERPRO) auf dem „DL-System-Markt“ existieren, müssen wir MIDeLoRA umso sorgfältiger motivieren und in Architektur und Eigenschaften von anderen DL-Systemen abgrenzen. MIDeLoRA wurde unter anderen Prämissen als vergleichbare Systeme entworfen; diese Prämissen sind in den im Rahmen dieser Arbeit analysierten DL-Anwendungsszenarien verwurzelt. Wir werden mit der MIDeLoRA-Architektur daher *wesentlich andere Schwerpunkte als vergleichbare DL-Systeme setzen*. Diese Schwerpunkte werden durch die Nutzungsprobleme und -defizite von Standard-DL-Systemen in unseren Anwendungskontexten motiviert.

Zunächst stellen wir die Anforderungen an MIDeLoRA zusammen und motivieren die Basis-Ideen, die wir im Entwurf umsetzen wollen. Es folgt dann der softwaretechnische Entwurf; als zentrale Leitmetapher für den Entwurf identifizieren wir den Begriff des MIDeLoRA-Raums. Wir widmen der Darstellung der Implementation ein weiteres Unterkapitel. Schließlich demonstrieren wir die Flexibilität und Generalität des MIDeLoRA-Baukastensystems durch Vorstellung einiger Ausprägungen.

6.1. Motivation und Anforderungen

Viele der im Folgenden dargestellten Punkte sind im Verlauf der Arbeit bereits hier und da erwähnt und diskutiert worden. Wir stellen aus Gründen der Übersichtlichkeit die Motivation und wesentlichsten Anforderungen an MiDELORA erneut zusammen; hierbei werden auch einige noch nicht diskutierte Anforderungen aufkommen.

6.1.1. Implementierung von Beweisern für Spezial-DLs

Eine offensichtliche Motivation für MiDELORA ist, dass wir im Rahmen dieser Arbeit *anwendungsmotivierte Spezial-DLs* wie die $\mathcal{ALCI}_{\mathcal{RCC}}$ -Familie (s. Kap. 3.1.5, S. 150) untersucht haben. Die (entscheidbaren) $\mathcal{ALCI}_{\mathcal{RCC}}$ -Logiken benötigen Spezialbeweiser, die wir mittels MiDELORA implementieren wollen (s. auch Kap. 3.1.6, Kap. 3.1.7, Kap. 3.3). MiDELORA soll jedoch auch Beweiser für Standard-DLs bereitstellen.

Wir sehen für MiDELORA daher *softwaretechnische Abstraktionen zur Definition von Beweisern für Spezial- und Standard-DLs* vor.

6.1.2. Implementierung aufgabenspezifischer Beweiser

DL-Beweiser unterscheiden sich nicht nur bzgl. der unterstützten DL, sondern auch bzgl. der *Aufgabe des Beweisers* voneinander. Offensichtlich kann ein *Instanzenretrieval-Beweiser* (Implementierung von `concept_instances`) andere Techniken und Optimierungen nutzen und zur Anwendung bringen, als ein allgemeiner ABox-Erfüllbarkeitsbeweiser (Implementierung von `abox_satisfiable?`).

Natürlich wird der Instanzenretrieval-Beweiser den ABox-Erfüllbarkeitsbeweiser verwenden, doch ist er nicht mit diesem identisch, sondern softwaretechnisch hinreichend kompliziert, um als dedizierter Beweiser zur Lösung eines speziellen Inferenzproblems betrachtet und ausgezeichnet zu werden.

Wir sehen für MiDELORA daher *softwaretechnische Abstraktionen zur Definition von Beweisern für spezifische Aufgaben* vor.

6.1.3. Flexible Adaption und Modifikation existierender Beweiser

Wir hatten in Kap. 3.1.5, S. 158 eine Situation vorgefunden, in der eine *Anpassungen bzw. Modifikationen eines existierenden Beweisers* wünschenswert erschien. Wir hatten erläutert, dass der *Domänenabschluss der ABox* bzgl.

6.1 Motivation und Anforderungen

der RCC-Rollen (s. Kap 3.1.5, S. 156) nicht nur durch Etablierung von RCC-Anzahlrestriktionen für die Individuen gleichsam „logisch implementiert“ werden kann, sondern dass ein alternatives Vorgehen darin bestünde, eine *spezialisierte $\exists$ -Regel* zu implementieren bzw. das Verhalten der existierenden $\exists$ -Regel entsprechend zu modifizieren, wodurch ebenfalls diskutierter RCC-Domänenabschluss erzielt würde. Diese interne Anpassbarkeit ist jedoch bei Verwendung eines externen, nicht offenen DL-Systems nicht gegeben.

In MIDeLoRA soll eine solche Regel-Anpassung hingegen prinzipiell möglich sein.¹ MIDeLoRA soll nicht nur die Anpassung von Regeln, sondern auch von *Regelanwendungsstrategien* und sogar von ganzen *Beweisern* ermöglichen. Wir wollen hier insbesondere Vererbung nutzen, sodass Standard-Verhalten ererbt und bei Bedarf durch Überschreibung und Parametrierung einzelner Methoden adaptiert werden kann.

MIDeLoRA wird daher ein umfangreiches Repertoire bzw. „Arsenal“ an Standard-Bausteinen zum Beweiserbau (Regeln und Strategien) als auch vollständige Beweiser bereitstellen. Dieses Repertoire kann dann für eigene Anwendungen genutzt, adaptiert und neu komponiert werden kann (ganz im Sinne eines „Baukastens“).

6.1.4. Offene und erweiterbare ABox-Repräsentation

MIDeLoRA wird als Baukastensystem zur Konstruktion von DL-Beweisern konzipiert und soll dementsprechend *offen, flexibel und erweiterbar* sein. Diese Eigenschaften sollen auch bzgl. der verwendeten *ABox-Repräsentation* gelten.

Das bisher vorgestellte Rahmenwerk ermöglicht bereits eine gewisse Flexibilität und Erweiterbarkeit. So können ABox-Individuen eines externen DL-Systems entweder mit Zusatzinformationen *assoziiert*, oder um Zusatzinformationen *erweitert* werden. Durch Zusatzinformation hatten wir u.a. *Repräsentationsdefizite* oder *Performanzprobleme* gelindert, aber auch *Zusatzdienste* ermöglicht.

Zur *Assoziation* von ABox-Individuen mit Zusatzinformationen wird ein hybrides Substrat verwendet, welches ein Komponenten-Substrat vom Typ ABox-Adapter-Substrat (ABox-Adapter-Substrate) hat; dieses stellt eine „virtuelle Substrat-Perspektive“ auf die externe ABox bereit. Zur (scheinbaren) *Erweiterung* der externen ABox-Individuen (um weitere Attribute) verwenden wir hingegen das ABox-Adapter-Spiegel-Substrat (ABox-Adapter-Mirror-Substrate), welches lokale Proxy-Substrat-Knoten für die externen ABox-Individuen anlegt. Es können dann anwendungsspezifische Spezialisierungen dieser Proxy-Knoten gebildet werden (um zusätzliche Information in zusätzlichen Attributen abzuspeichern, etc.).

¹Wir behaupten hier nicht, dass dies für *beliebige* Nutzer von MIDeLoRA der Fall sein muss.

Wir hatten für das DLMAPS-System beide Möglichkeiten durch entspr. Karten-Substrate realisiert.

Nachteilig an beiden Möglichkeiten (Assoziation und Erweiterung) ist jedoch, dass die Zusatzinformation aus der Perspektive des externen ABox-DL-Systems nicht sichtbar ist. Prinzipiell kann eine Sichtbarkeit dieser Zusatzinformation nur dann erzielt werden, wenn die Information in der vom DL-System unterstützten DL prinzipiell (evtl. mittels logischer Umkodierungen) abbildbar ist. Wir hatten in dieser Arbeit sowohl Szenarien betrachtet, in denen das möglich ist, als auch solche, in denen dies nicht möglich ist. Selbst wenn die Information abbildbar ist, führt dies zu einem hohen Kommunikationsaufwand über die APIs; die erforderliche Abbildung bedingt u.U. einen Modellbruch und u.U. auch einen „Impedance Mismatch“ an den Schnittstellen. Speziell beim Vorliegen einer Netzwerkschnittstelle muss der Kommunikationsaufwand mittels Caches etc. entschärft werden, was wie in Kap. 5.3.3.6.1, S. 367 kritisiert werden kann.

Prinzipiell wäre es also vorteilhaft, wenn die Erweiterung um Zusatzinformation *direkt auf der Ebene der intern vom ABox-DL-System verwendeten ABox-Repräsentation bzw. ABox-Speicherstrukturen* vorgenommen werden könnte. Wir benötigen also wieder eine *interne* Perspektive, bzw. ein *integriertes* (im Gegensatz zu einem externen) DL-System. Wir hatten in Kap. 1 bereits diskutiert, dass wir (entgegen dem aktuellen Trend zur losen Kopplung) hier eine *strikte Kopplung* und somit eine *hohe Kohäsion* erreichen wollen, um so „Impedance Mismatches“, Modellbrüche und Reibungsverluste an den Schnittstellen zu vermeiden. Aus der Perspektive des Autors ist nicht klar, wie die anvisierte Offenheit und Erweiterbarkeit mittels einer reinen Komponenten-Architektur erreicht werden kann.² Dem „Offen-Geschlossen-Prinzip“ folgend [Meyer 1990, Kap. 2.3], sehen wir daher vor, diese Eigenschaften auf programmiersprachlicher Ebene durch Nutzung von Vererbung und durch die Bereitstellung „domänenspezifischer Sprachen“ (COMMON LISP-Makrosprachen) zu realisieren. So wird die anvisierte hohe Kohäsion erzielt; es sollte klar sein, dass hierdurch alle oben diskutierten Probleme adressiert werden können. Wir werden dies im Verlauf dieses Kapitels detailliert demonstrieren. In [Wessel 2003b] hatten wir die Idee eines derartig *offenen und erweiterbaren ABox-DL-Systems* vorgestellt und motiviert.

Derartige offene und erweiterbare ABox-Speicherstrukturen sind nicht zuletzt auch zur Realisierung persistenter ABoxen wünschenswert. Im Substrat-Datenmodell können die diskutierten Aspekte berücksichtigt werden. Wir hatten dies an mehreren Stellen in Kap. 5 verdeutlicht. Eine MIDELORA-ABox wird daher als spezielles Substrat entworfen. Wir sehen für MIDELORA daher

²Aus dem Gesagten wird schon deutlich, dass wir hier nicht Black Box-, sondern White Box-Komponenten benötigen.

6.1 Motivation und Anforderungen

softwaretechnische Abstraktionen zur Offenlegung und Erweiterung der ABox-Repräsentation vor.

6.1.4.1. Abkehr von der Newell-Aufteilung?

Bedeutet MIDELORA nun eine Abkehr von der von Newell vorgeschlagenen Aufteilung eines wissensbasierten Systems (und somit auch eines DL-Systems) in

1. eine Wissensebene (Knowledge Level) und
2. eine Symbolebene (Symbol Level)?

Sind beide Ebenen in MIDELORA nun „vermischt“, da das Substrat-Datenmodell (auf dem MIDELORA aufbaut) sowohl als Datenmodell auf der Wissensebene (Kap. 4), als auch als Datenmodell auf der Symbolebene (Kap. 5) beschrieben und begriffen werden kann?

Wir meinen nein, denn aus der Perspektive des Autors beschreibt Newell mit seinen Ebenen weniger die „Softwareschichten“ eines WBS bzw. einer WBS-Grobarchitektur, als vielmehr konzeptionell verschiedene Perspektiven auf ein WBS. Wie anhand der bereits geführten Diskussionen deutlich wurde, erscheint uns ein fließender Übergang und eine enge Kopplung zwischen der Wissens- und der Symbol-Ebene oftmals wünschenswert. Über die Art der Schnittstellen zwischen Wissens- und Symbolebene wird von Newell nichts ausgesagt. Wir meinen, dass diesen Schnittstellen mehr Beachtung in der DL-Forschung gewidmet werden sollte. Die Newell-Ebenen sind zudem zu grob und verleiten dazu, ein DL-System primär auf der Wissensebene zu betrachten (diese Perspektive ist für die meisten Nutzer sicherlich auch adäquat). Hierdurch wird jedoch auch viel Potential verschenkt. Dies wird insbesondere beim Vergleich mit den *verfeinerten Schichtenarchitekturen* für Datenbanksysteme deutlich, s. z.B. [Lockemann u. Schmidt 1987, Kap. 2].

Heutige DL-Systeme bieten fast ausschließlich APIs auf der Wissensebene. Beispielsweise ist die KRSS-Tell-Anweisung (`instance michael person`) – womit die Konzept-Assertion *michael : person* zur ABox hinzugefügt wird – auf der Wissensebene anzusiedeln. Hier wird nichts darüber ausgesagt, wie oder mit welchen Mitteln diese Assertion auf der Symbolebene repräsentiert wird. Während in RACERPRO diese ABox-Assertion tatsächlich auch auf der Symbolebene als Satz bzw. symbolische Struktur gespeichert wird, so verwendet MIDELORA hier bereits eine graphbasierte Repräsentation (da es auf dem Substrat-Datenmodell aufbaut). Aus diesem Unterschied ergeben sich weitreichende Konsequenzen (s.u.), die die gesamte Architektur beeinflussen.

Wir haben in dieser Arbeit motiviert, dass es wünschenswert sein kann, einen Einfluss auf die Repräsentation auf der Symbolebene nehmen zu können. Derartige Aspekte werden von den primär auf der Wissensebene anzusiedelnden

APIs heutzutage noch nicht adressiert. Für MIDELORA sehen wir also Schnittstellen zur Adressierung von Aspekten auf der Symbolebene vor. Wir behaupten, dass MIDELORA das erste DL-System ist, welches diesen Aspekten bereits beim Entwurf (durch Verwendung des Substrat-Datenmodells) gebührend Rechnung trägt.

6.1.5. Anforderungen aus den Leitdomänen und reichhaltige ABoxen

Eine weitere Liste von Anforderungen bzw. Motivation für die MIDELORA-Architektur erhalten wir durch die Analyse der Nutzung von RACERPRO im DLMAPS-System. Diese kann aus folgenden Gründen nicht als unkritisch bezeichnet werden.

Beispielsweise wird zur Beantwortung der DISKQL-Anfrage **94** ein „zählendes“ Konzept-Atom ($?*x \text{ (at-least } 8 \text{ ppi top)}$) und somit eine ABox-Repräsentation des RCC-Netzes in Form von RCC-Rollen-Assertionen benötigt (andernfalls ist die Anfrage nicht beantwortbar). Das komplette RCC-Netzwerk muss daher zunächst vollständig berechnet und dann über die Netzwerkschnittstelle in Form von RCC-Rollen-Assertionen zum RACERPRO-Server transferiert werden. Das explizite Vorausberechnen und Materialisieren räumlicher Relationen in Form von Rollen-Assertionen ist zum einen aufgrund des hohen Speicherplatzbedarfs und speziell bei Vorliegen eines entfernten DL-Systems aufgrund des Kommunikationsaufwands nicht wünschenswert. Zum anderen gibt es auch Klassen räumlicher Relationen, für die ein explizites Vorausberechnen nicht möglich ist, da es von ihnen eine überabzählbare Anzahl gibt, z.B. die Klassen der ϵ - und Δ -Relationen, s. Kap. 5.3.4.3.2, S. 384 (zwischen zwei Objekten gelten u.U. eine überabzählbare Anzahl solcher Relationen).

Zur Beantwortung von Anfragen wird in vielen Fällen zudem nur eine *Teilmenge* der RCC-Assertionen benötigt werden. Beispielsweise werden zur Auswertung des BAA-Konzept-Atoms ($?*x \text{ (at-least } 8 \text{ ppi top)}$) lediglich die $\{PPI, PP\}$ - bzw. $\{NTTP, NTTPI, TPP, TPPI\}$ -Assertionen benötigt. Die „überflüssigen“ RCC-Rollen-Assertionen sind dann in gewisser Weise „Ballast“ und verlangsamen u.U. den Beweiser.

In MIDELORA soll es daher möglich sein, auf das explizite Vorausberechnen von RCC-Netzen zu verzichten, und lediglich die *für die Anfragebeantwortung bzw. zur Lösung des aktuellen Inferenzproblems benötigten* RCC-Rollen-Assertionen zu betrachten. Wir sprechen von *anfragefokussierten Beweisern* bzw. *anfragefokussierter Anfragebeantwortung*. Idealerweise wird diese Fokussierung zudem inkrementell während des Inferenzprozesses passieren. Eine MIDELORA-RCC-ABox wird daher initial gar keine expliziten RCC-Assertionen enthalten.

6.1 Motivation und Anforderungen

Werden nun im Rahmen des Beweisprozesses die RCC-Rollennachfolger eines Individuums aus der ABox angefordert, so führt dies zur inkrementellen Berechnung der angeforderten RCC-Nachfolger anhand der *Räumlichkeit der ABox* mittels *Inspektionsmethoden* (s. Kap. 3.1.1.1, S. 135), genauso wie dies zur Auswertung der *virtuellen* RCC-Atome in der DISKQL geschieht (s. Kap. 5.3.4.3.2, S. 384 und Kap. 5.3.4.6.2, S. 397). Dem Beweiser steht es dann frei, diese „virtuellen“ Rollen-Assertionen auch zu materialisieren (ähnlich einem RCC-Karten-Substrat mit expliziten Kanten).

Die Berücksichtigung solch virtueller Assertionen ist offensichtlich mit einem externen DL-System wie RACERPRO nicht möglich, da hier die Räumlichkeit der ABox nicht zur Verfügung steht, und auch keine entspr. Schnittstellen vorgesehen sind, die das beschriebene inkrementelle Anfordern von Inferenzleistungen eines externen (hier räumlichen) „Orakels“ während des Beweisprozesses ermöglichen würden. Eine alternative Möglichkeit wären *offene und nutzererweiterbare konkrete Domänen*.

Sicherlich könnte eine RACERPRO-ABox auch *anfragespezifisch so vorbereitet werden, dass sie lediglich die für die Anfragebeantwortung benötigten Assertionen enthält*. Dies ist im Gegensatz zum expliziten Vorausberechnen *aller* geltender Rollen-Assertionen prinzipiell auch für Relationen wie die ϵ - und Δ -Relationen möglich, da ja die *Anfrage* feststeht. Im Allgemeinen ist es jedoch (insbesondere aufgrund der TBox) oftmals nicht möglich, im Voraus zu entscheiden, welche Assertionen nun zur Lösung des Inferenzproblems benötigt werden, und welche nicht. Eine solche ABox wird daher u.U. zu viele (und daher auch nicht benötigte) Assertionen enthalten. Aufgrund des hohen Vorbereitungs-Aufwands und da eine solche ABox nur für diese und sehr ähnliche Anfragen genutzt werden kann (somit muss die Vorbereitungszeit letztlich der Anfrage-Beantwortungszeit zugeschlagen werden), ist dieses Vorgehen unbefriedigend. Letztlich führt jedes Anfrage zu einer neuen Anfrage-ABox; dies bedeutet aber auch, dass alle RACERPRO-internen Cache- und Indexstrukturen für jede neue Anfrage invalidiert werden, dass der initiale ABox-Erfüllbarkeitstest erneut anfällt, etc. Das Problem ist daher nicht befriedigend mit einem externen DL-System zu lösen.

MIDeLoRA bietet hingegen *reichhaltige ABox-Repräsentationen*. Da eine MIDeLoRA-ABox eine Instanz der Klasse ABox und daher letztlich ein spezielles Substrat (Substrate) ist, sind alle für das Substrat-Datenmodell bereits diskutierten Vor- und Nachteile auf MIDeLoRA-ABoxen übertragbar:

- Aspekte der *physischen Datenorganisation* können adäquat berücksichtigt werden. Für spezielle Anwendungsklassen können z.B. zusätzliche Indexstrukturen nachgepflegt werden. Sehr große ABoxen können zudem nicht mehr im Hauptspeicher gehalten werden (s. Kap. 3.2.5). Wir hatten diese Aspekte in Kap. 5.1.1 diskutiert.

- *Darstellungsunabhängigkeit von der ABox-Repräsentation* ist notwendig und wird im Substrat-Datenmodell durch generische Schnittstellen, Iteratoren etc. bereitgestellt, s. Kap. 5.1.2.3, S. 245.

Im Quelltext eines MiDeLoRA-Beweisers sollten keine Annahmen bzgl. physischer Darstellungsaspekte gemacht werden. So sollte ein MiDeLoRA-Beweiser ohne Änderungen am Quellcode sowohl auf einer Hauptspeicher-ABox, als auch einer DB-ABox lauffähig sein. MiDeLoRA ist (nach Wissen des Autors) das erste tableaubasierte DL-System, welches diese Darstellungsunabhängigkeit ermöglicht und konsequent „gegen entspr. Schnittstellen programmiert“ [Gamma u. a. 1996, S. 21] wurde.

MiDeLoRA bietet *räumliche ABoxen* (s. z.B. Klasse MiDeLoRA-Implicite-Relationen-Karten-Substrat (MiDeLoRA-Implicit-Relations-Map-Substrate), Abb. 5.11, S. 271), wobei es sich um spezielle reichhaltige ABoxen handelt.

Reichhaltigkeit definieren wir nun noch etwas genereller: Eine *reichhaltige ABox* ist eine ABox, in der während des Beweisprozesses zusätzlich zu den syntaktisch (im Hauptspeicher in Datenstrukturen) vorhandenen ABox-Assertionen (bei Bedarf) noch weitere *virtuelle Assertionen* berücksichtigt werden können, die *anhand der Reichhaltigkeit der Repräsentation* bei Bedarf errechnet (und optional materialisiert) werden können. Wir können die Situation auch so beschreiben:

- In einer Standard-ABox ist die Menge der ABox-Assertionen fest vorgegeben. Sie müssen *im Voraus* in die ABox eingetragen werden und sind im Hauptspeicher durch Datenstrukturen materialisiert.
- In einer *reichhaltigen ABox* sind im Hauptspeicher nur einige Assertionen (in Datenstrukturen) materialisiert. Andere Assertionen sind hingegen lediglich virtuell. Die virtuellen Assertionen werden erst während des Beweisprozesses berücksichtigt, wenn sie vom Beweiser *von der ABox angefordert werden*. Es steht dem Beweiser frei, die von der ABox gelieferten virtuellen Assertionen dann im Hauptspeicher zu materialisieren oder nicht. Das Anfordern virtueller Assertionen von der ABox wird letztlich durch Iteratoren realisiert.

Zur Illustration betrachten wir zwei Beispiele für reichhaltige ABoxen:

- In einer räumlichen RCC-ABox ist die Menge der virtuellen Assertionen durch die aufgrund der Räumlichkeit der ABox-Individuen implizierten RCC-Rollen-Assertionen gegeben. Das Anfordern einer solchen Assertion beinhaltet dann das Auswerten einer räumlichen RCC-Inspektionsmethode.³

³Tatsächlich kann dies auch als eine Art von „Inferenz“ angesehen werden, da implizit vorhandenes Wissen durch die Inspektionsmethode explizit gemacht wird; es handelt sich jedoch

6.1 Motivation und Anforderungen

- In einer *Datenbank-ABox* werden virtuelle Assertionen in der Datenbank gehalten. Das Anfordern einer solchen Assertion durch den Beweiser veranlasst das Absetzen einer Datenbankabfrage.

Man mag hier einwenden, dass das Absetzen von u.U. vielen elementaren Datenbankabfragen während des Beweisprozesses zu einer schlechten Performanz an der DB-Schnittstelle und somit u.U. zu einem „Impedance Mismatch“ führt. Letztlich sind hier die gleichen Argumente wie in Kap. 5.2.2.3.3, S. 298 zu wiederholen. Die Performanz muss nicht notwendigerweise schlecht sein, wenn diese Datenbankabfragen bzw. die diese Abfragen absetzenden Iteratoren hinreichend informiert sind, und zusätzlich entspr. Cache- und Speicherverwaltungsstrategien implementiert werden. Wir geben jedoch zu, dass das Hauptproblem eben in der oftmals mangelhaften Informiertheit besteht (s. auch Diskussion in Kap. 1).

Trotz dieser Probleme halten wir an der Idee der virtuellen Assertionen fest und bauen die Idee im weiteren Verlauf des Kapitels entsprechend aus, da die durch sie eröffneten Potentiale die Problempunkte überwiegen.

Betrachtet man die Menge aller ABox-Assertionen einer ABox nun als *partitioniert in die Menge der virtuellen Assertionen und die Menge der (normalen) Assertionen*, so kann *Anfragefokussierung* nun *konzeptionell* durch zwei Techniken erreicht werden:

- Eine Teilmenge der (normalen) Assertionen kann *markiert* werden; nur markierte Assertionen werden vom Beweiser betrachtet.
- Die virtuellen Assertionen sind nicht explizit und werden wie beschrieben erst bei Bedarf vom Beweiser von der ABox angefordert. Es kann somit individuell und dynamisch für jede einzelne angeforderte virtuelle Assertion entschieden werden, ob die Assertion geliefert und somit berücksichtigt werden soll oder nicht. All dies obliegt letztlich den entspr. Iteratoren.

Damit der Beweiser nicht unvollständig wird, muss garantiert werden, dass tatsächlich alle für das aktuelle Inferenzproblem benötigten Assertionen betrachtet bzw. im Falle der virtuellen Assertionen auch angefordert werden. Wir kommen auf das Problem in Kap. 6.2.3.2 zurück.

um eine Inferenz auf Datenstruktur- bzw. Symbol-Ebene, die somit nicht mit der durch das Beweisverfahren implementierten Inferenz auf logischer bzw. Wissens-Ebene zu vergleichen ist.

6.1.6. Tableau-Repräsentation und Repräsentation von Assertionen

Da MIDELORA ein Substrat als ABox-Repräsentation nutzen soll, wird letztlich auch das Tableau (die Arbeitsdatenstruktur des Beweisers) ein Substrat sein, welches im Verlauf des Beweisprozesses vom Tableau-Beweiser anhand der Eingabe-ABox um zusätzliche Assertionen erweitert wird, wie in Kap. 2.2.7.2, S. 102 dargestellt. Dementsprechend unterscheiden wir im Folgenden *ABox-Assertionen* von *Tableau-Assertionen*.

In MIDELORA existieren Assertionen nicht als eigenständige Objekte bzw. Datenstrukturen. Die Konzept-Assertionen sind letztlich durch Elemente in L_V , und die Rollen-Assertionen durch Elemente in L_E gegeben. Assertionen sind somit automatisch bzgl. i und (i, j) indiziert. Mittels $L_V(i)$ erhalten wir alle Konzept-Assertionen bzgl. i , und mittels $L_E((i, j))$ alle Rollen-Assertionen, die (i, j) betreffen. Wir hatten diese „Substrat-Perspektive“ auf eine ABox bzw. ein Tableau in Prop. 2, S. 197 formuliert. Die Existenz einer (nichtvirtuellen) Konzept-Assertion ist daher an die Existenz eines explizit vorhandenen Knotens, und die Existenz einer (nichtvirtuellen) Rollen-Assertion an die Existenz einer expliziten Kante im Substrat geknüpft.

Nehmen wir an, die ABox enthält die Konzept-Assertion $i : C$. Gestehen wir zu, dass $i : C$ nun auch eine *virtuelle* Assertion sein kann, so ergeben sich drei Möglichkeiten (**V1**, **V2**, **V3**), weshalb $i : C \in \mathcal{A}$ gelten kann. Sei V wieder die Knotenmenge des ABox-Substrates:

V1 $C \in L_V(i)$ und daher $i \in V$: Es handelt sich um eine *normale* Assertion.

V2 $C \notin L_V(i)$ und $i \in V$: Es handelt sich um eine *virtuelle* Assertion.

Betrachten wir den Fall einer räumlichen ABox. Dann könnte beispielsweise $C = \exists(\text{area}). >_{10.000}$ sein, und $i : C$ könnte aufgrund der Geometrie des Knotens i gelten. Die Gültigkeit von C für i wird dann bei Bedarf mittels einer Inspektionsmethode verifiziert. Dabei wird C i.d.R. vom Beweiser (durch die Anfrage) vorgegeben.

Da der Knoten i bereits explizit ist, muss C lediglich zu $L_V(i)$ hinzugefügt werden und kann dann im Beweisprozess wie eine normale Assertion berücksichtigt werden.

V3 $C \notin L_V(i)$ und $i \notin V$: Es handelt sich um eine *virtuelle* Assertion.

Im Unterschied zu **V2** ist i hier *nicht* explizit, d.h., es existiert kein Knoten i in V . Beispielsweise kann i lediglich in einer Datenbank vorhanden sein. Ein solches i nennen wir auch einen *virtuellen Knoten* bzw. ein *virtuelles Individuum*. Offensichtlich müssen dann auch alle Konzept- und Rollen-Assertionen, die sich auf i beziehen, virtuell sein.

6.1 Motivation und Anforderungen

Wir schicken voraus, dass eine solche virtuelle $i : C$ -Assertion nur dann von MIDeLoRA berücksichtigt werden kann, wenn i materialisiert wird und somit ein neuer Knoten $i \in V$ erzeugt wird, für den $L_V(i) := \{C\}$ gesetzt werden kann. Der Grund hierfür ist, dass i letztlich *als Träger weiterer Verwaltungsinformationen (z.B. von Abhängigkeitsinformationen, s.u.) benötigt wird*.

Da in der ABox bis zum Zeitpunkt der Materialisierung von i kein *expliziter Hinweis* auf die $i : C$ -Assertion vorhanden ist, ist dafür Sorge zu tragen, dass die $i : C$ -Assertion während des Beweisprozesses nicht vergessen bzw. übersehen wird.

Für eine *in der ABox geltende Rollen-Assertion* $(i, j) : R$ ergeben sich nun fünf Möglichkeiten (**E1 – E5**), weshalb $(i, j) : R \in \mathcal{A}$ gelten kann:

- E1** $R \in L_E((i, j))$ und daher $(i, j) \in E$, $i, j \in V$: Es handelt sich um eine normale Assertion. Dieser Fall ist analog zu **V1**.
- E2** $R \notin L_E((i, j))$ und $(i, j) \in E$, $i, j \in V$: Es handelt sich um eine virtuelle Assertion. Dieser Fall ist analog zu **V2**.

Betrachten wir wieder den Fall einer räumlichen ABox. Gilt beispielsweise bereits $DC \in L_E((i, j))$, so kann bei Bedarf anhand der Räumlichkeit von i und j mittels einer Inspektionsmethode z.B. $\Delta(i, j) \leq 10.000$ verifiziert und symbolisch als qualitative Rollen-Assertion repräsentiert werden, indem $\Delta_{\leq 10.000}$ zu $L_E((i, j))$ hinzugefügt wird. Dies ist möglich, da (i, j) ja explizit ist.

- E3** $R \notin L_E((i, j))$ und $(i, j) \notin E$, aber $i, j \in V$.

Die Kante ist nun nicht mehr explizit, im Gegensatz zu den Knoten i, j . Anders als für Assertionen vom Typ **V3** ist MIDeLoRA u.U. nicht auf explizite Kanten angewiesen, damit virtuelle Rollen-Assertionen im Beweisprozess Berücksichtigung finden können. Der Grund hierfür ist, dass für Rollen-Assertionen oftmals keine zusätzlichen Verwaltungsinformationen kanten- und somit (i, j) -lokal abgespeichert werden müssen, s.u.

Wie auch für Assertionen vom Typ **V3** muss sichergestellt werden, dass sie nicht übersehen werden. Die explizit vorhandenen Knoten können als Anhaltspunkt für die Erfragung solch virtueller Rollen-Assertionen dienen.

- E4** $R \notin L_E((i, j))$ und $(i, j) \notin E$, sowie $\{i, j\} \cap V \neq \emptyset$, aber $\{i, j\} \not\subseteq V$. Prinzipiell wie **E3**, doch ist nun zusätzlich auch noch einer der Knoten implizit.
- E5** Schließlich gibt es noch den Fall $R \notin L_E((i, j))$ und $(i, j) \notin E$ und $\{i, j\} \cap V = \emptyset$. Prinzipiell wie **E3**, doch sind nun beide Knoten implizit.

Wir hatten erläutert, dass zur Berücksichtigung einer virtuellen $i : C$ -Assertion vom Typ **V3** zwingend ein neuer expliziter Knoten i mit $L_V(i) := \{C\}$ erzeugt

werden muss. Zur Berücksichtigung einer virtuellen Rollen-Assertion vom Typ **E3**, **E4** oder **E5** ist dies u.U. nicht notwendig. Dies hängt letztlich von der Art der die Assertion anfordernden Algorithmuses bzw. Beweisers ab. Solche materialisierten virtuellen Objekte können bei Bedarf auch wieder dematerialisiert werden, doch muss dann entweder sichergestellt werden, dass die Objekte im weiteren Verlauf des Algorithmuses nicht mehr benötigt werden (s. Diskussion in Kap. 6.2.3.2 für hiermit verbundene Probleme), oder die zugehörigen Verwaltungsinformationen müssen im Falle einer nach dieser Dematerialisierung *erneut* erfolgten Materialisierung ebenfalls im vorherigen Zustand wieder hergestellt werden. Dies erfordert entweder, dass auch die Verwaltungsinformationen in die Datenbank geschrieben werden, oder das *zusätzliche dedizierte Hauptspeicher-Datenstrukturen* hierfür verwendet werden. Letzteres führt jedoch zu einer unschönen Separierung von eigentlich zusammengehörigen Informationen, die zudem miteinander konsistent gehalten werden müssten. Zumal ergibt sich hier ein prinzipielles Problem, sobald die Menge der virtuellen DB-ABox-Assertionen nicht mehr zu einem Zeitpunkt in den Hauptspeicher passt. Es werden dann dedizierte und beweiserspezifische *Speicherverwaltungsregeln* zur Adressierung dieser Probleme benötigt.

6.1.7. Inkrementelle Basis-ABox-Retrieval-Funktionen

Wir hatten mehrfach motiviert, dass für die ABF-Generatoren der BAA informierte und inkrementelle Generatoren wünschenswert sind, die auch aus *interner Perspektive* diese Eigenschaften aufweisen (s. Kap. 3.2.4, S. 182 und Kap. 5.2.2.3.3, S. 298). In MIDELORA wollen wir daher inkrementelle Versionen der ABox-Basis-Retrieval-Probleme (*concept_instances*, *individual_role_successors* etc.), s. Kap. 2.2.2.5, S. 78 bereitstellen. MIDELORA wird die Ergebnisse dieser API-Funktionen nicht nur *inkrementell über die Schnittstelle anbieten*, sondern auch *intern inkrementell* und daher bei Bedarf errechnen.

6.2. Entwurf von MiDELORa

Wie auch in Kap. 4 beschreiben wir zunächst die Basis-Ideen des Entwurfs und diskutieren dann zunehmend speziellere Aspekte.

MiDELORa ermöglicht die Definition von DL-, aufgaben-, und ABox-spezifischen Beweisern, den Anforderungen aus Kap. 6.1.1, 6.1.2, 6.1.4 entsprechend. Die zentrale Leitmetapher für unseren Entwurf ist der dreidimensionale MiDELORa-Raum. Ein Beweiser kann als ein Punkt in diesem Raum angesehen werden (tatsächlich kann er jedoch i.d.R. *Regionen* in diesem Raum abdecken). Die zentralen Beweiser sind Beweiser zur Entscheidung des ABox-Erfüllbarkeitsproblems. Die zugrundeliegenden Kalküle sind Tableau-Verfahren. Wir hatten in Kap. 2.2.7.2, S. 102 einen Kalkül zur Entscheidung von *ALC*-Konzepterfüllbarkeit diskutiert und gleichzeitig die erforderlichen Grundlagen eingeführt. Wie dort erwähnt, kann dieser Kalkül auch zur Entscheidung von *ALC*-ABox-Erfüllbarkeit (mit leerer TBox bzw. bzgl. einfacher TBoxen) verwendet werden.

Es stellt sich also die Frage, wie MiDELORa die Definition bzw. Implementation von Tableau-Beweisern unterstützt; es werden softwaretechnische Abstraktionen zur Definition von MiDELORa-Tableau-Beweisern im MiDELORa-Raum benötigt, da MiDELORa ja ein Baukastensystem ist. Zunächst ist jedoch zu klären, wie die *Tableau-Repräsentation* im Substrat-Datenmodell aussehen soll. Die Tableau-Repräsentation ist zum einen so zu gestalten, dass die in der Analyse ermittelten Anforderungen umgesetzt werden. Zum anderen eröffnet die Tableau-Repräsentation auch Potentiale und Risiken.

Dieser Abschnitt ist wie folgt strukturiert:

- Wir definieren zunächst den MiDELORa-Raum. Der MiDELORa-Raum ist die zentrale Entwurfs-Metapher.
- Wir schildern dann die dem Entwurf zugrundeliegenden Basis-Ideen und geben das Gerüst bzw. den Basis-Entwurf an. Hier ist insbesondere die Tableau-Repräsentation zu entwerfen. Der MiDELORa-Kern ist für die Tableau-Verwaltung verantwortlich.
- Wir identifizieren die Potentiale, die sich durch die Architektur ergeben. Wir begründen, warum die Anforderungen durch diesen Entwurf umgesetzt werden können, und welche Probleme und Risiken sich ergeben. In der Liste der eröffneten Potentiale werden wir auch Punkte finden, die für die *nächste Generation von DL-Systemen* relevant werden (z.B. Transaktionalität und Persistenz).
- Es wird ein umfangreiches Arsenal an Basis-Funktionalität benötigt; u.a. müssen Konzepte und TBoxen repräsentiert und verwaltet werden. Wir entwerfen die benötigte Basis-Funktionalität. Der Schwerpunkt der Arbeit

liegt jedoch (der Substrat-Idee folgend) auf der ABox.

- Wir schildern, wie der MIDELORA-Raum *strukturiert* wird. Hierzu muss insbesondere geklärt werden, wie die dynamische Auswahl eines geeigneten Beweisers funktioniert. Damit nicht für jeden Punkt im Raum ein eigener Beweiser erforderlich wird, stellt sich die Frage, wie Vererbung zur Definition genutzt werden kann, sodass Beweiser wie erwähnt Regionen, und nicht nur Punkte, in diesem Raum abdecken können.
- Schließlich werden wir die für MIDELORA zentralen softwaretechnischen Abstraktionen zur Definition von Tableau-Beweisern vorstellen. Diese werden später als COMMON LISP-Makrosprachen implementiert, wodurch sie zu programmiersprachlichen Abstraktionen werden.
- Wir kommen erneut auf das Thema „Flexibilität“ zurück und begründen erneut, warum MIDELORA als flexibel anzusehen ist.
- Da MIDELORA ein Baukastensystem ist, muss es ein umfangreiches Arsenal funktionsfähiger Standard-Bausteine bieten („muss ein Angebot bereitstellen“). Hierzu gehören in erster Linie Tableau-Regeln für die Standard-Konzeptkonstruktoren bis zur Sprache $\mathcal{ALCHIf}\mathcal{N}_{\mathcal{R}+}$.⁴ Diese Regeln sollen natürlich *wiederverwendbar und benutzerspezifisch komponierbar und kombinierbar* sein.
- In MIDELORA sollen *Stand der Kunst-Optimierungsverfahren* Anwendung finden. Wir stellen die verwendeten Optimierungsverfahren steckbriefartig vor. Prinzipiell werden diese oder ähnliche Verfahren heutzutage in allen reiferen DL-Systemen verwendet.

6.2.1. Der MIDELORA-Raum

Ein MIDELORA-Beweiser kann als ein Punkt bzw. Tripel in einem dreidimensionalen Raum zueinander orthogonaler Dimensionen betrachtet werden. Die erste Achse ist durch die *Aufgabe* des Beweisers, die zweite durch die *unterstützte DL*, und die dritte Achse durch die *Klasse der ABox*, für die der Beweiser definiert ist, gegeben:

Definition 106 (MIDELORA-Raum, MIDELORA-Beweiser) Der MIDELORA-Raum ist definiert als dreidimensionaler Raum

$$\mathcal{T} \times \mathcal{DL} \times \mathcal{K}.$$

Hierbei ist

⁴ Es ist klar, dass f hier letztlich redundant (da schon durch $\mathcal{N}$ impliziert), und somit $\mathcal{ALCHIN}_{\mathcal{R}+}$ ausreichend wäre, doch halten wir an dem f im Namen fest, um deutlich zu machen, dass f üblicherweise anders als $\mathcal{N}$ implementiert wird.

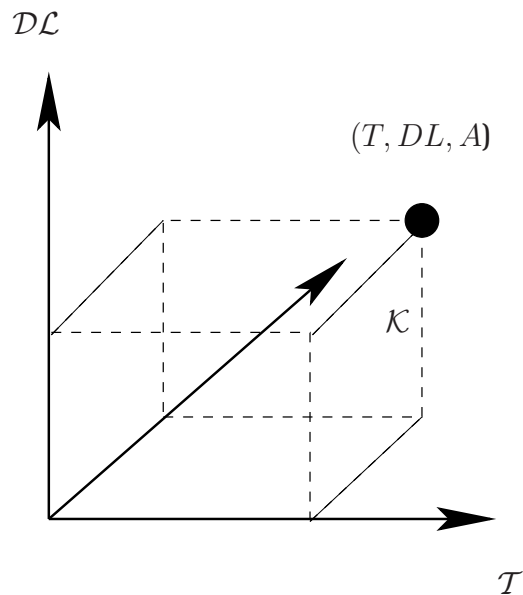


Abbildung 6.1.: Der MIDeLoRA-Raum und ein Beweiser in diesem Raum.

$\mathcal{T}$ eine Menge von *Tasks* bzw. *Namen von ABox-Inferenzdiensten*

(z.B. $\mathcal{T} = \{\text{abox\_consistent?}, \text{concept\_instances}, \text{individual\_role\_successors}, \dots\}$),

$\mathcal{DL}$ eine Menge von Namen für DLs (z.B. $\mathcal{DL} = \{\mathcal{ALC}, \mathcal{ALCI}_{\mathcal{RCC}}, \dots\}$), und

$\mathcal{K}$ eine Menge von ABox-Klassennamen (also ABox inkl. der Menge aller Unterklassen von ABox).

Ein MIDeLoRA-Beweiser wird für ein Tripel

$$(T, DL, K) \in \mathcal{T} \times \mathcal{DL} \times \mathcal{K}$$

definiert und stellt daher einen Beweiser für ABoxen der Klasse K zur Lösung des Inferenzproblems T in der DL DL bereit. ■

Die Menge $\mathcal{T}$ enthält lediglich (Namen für) *ABox-Inferenzdienste*. TBox-Inferenzdienste (z.B. *taxonomy* zur Berechnung der Taxonomie) werden hier nicht aufgenommen (s. auch Kap. 6.2.4.4).

6.2.2. Basis-Entwurfsideen für einen substratbasierten ABox-Erfüllbarkeitsbeweiser

Die Basis-Entwurfsideen betreffen zunächst die objektorientierte Tableau-Repräsentation und die sich hieraus ergebenden Konsequenzen.

6.2.2.1. Eine ABox ist ein Substrat, das Tableau ist die ABox

Wir hatten die Grundidee ja schon in der Analyse diskutiert: Eine ABox ist nun ein spezielles Substrat, eine Instanz der Klasse ABox. Ein und dieselbe Instanz stellt sowohl die ABox als auch das Tableau dar.

Wie in Kap. 2.2.7.2 erläutert, entscheidet ein Tableau-Beweiser die Erfüllbarkeit einer Eingabe-ABox durch Anwendung von Tableau-Regeln auf die ABox anhand einer vorgegebenen globalen Regelanwendungsstrategie. Kann eine widerspruchsfreie Vervollständigung der ABox hergeleitet werden, so ist die ABox erfüllbar. Führt andererseits jede Regelanwendungsreihenfolge (die der Strategie gehorcht) zu einem Widerspruch, so ist die ABox unerfüllbar. Prinzipiell handelt es sich hierbei um einen nichtdeterministischen Regelanwendungsprozess. Der Nichtdeterminismus muss in einem deterministischen Computer durch ein *Suchverfahren* aufgelöst bzw. implementiert werden.

Die Klasse ABox ist daher so zu entwerfen, dass eine Instanz sowohl als Eingabe-ABox als auch als Tableau-Arbeitsdatenstruktur verwendet werden kann. Zur Verwaltung des Tableau-Graphen werden die Dienste des Substrat-Rahmenwerks in Anspruch genommen. Die gewünschte physische Darstellungsunabhängigkeit der Beweiser erreichen wir wieder durch entspr. Iteratoren (Beweiser werden ausschließlich mittels der Iteratoren und Zugriffsmethoden auf die Knoten und Kanten des Tableaus zugreifen).

Da nun dieselbe Substrat-Instanz zur Repräsentation der ABox als auch des Tableaus verwendet wird, müssen Substrat-Knoten, die ABox-Individuen repräsentieren (sog. *alte Individuen*) von erst während der Tableau-Expansion erzeugten Substrat-Knoten (sog. *neuen Individuen*) unterschieden werden können. Betrachten wir die Eingabe-ABox $\{i : \exists R.C\}$, so enthält jede Vervollständigung bzw. jedes vollständige Tableau, das aus dieser ABox hervorgeht, ein neues Individuum, hier $temp_1$ genannt: $\{i : \exists R.C, (i, temp_1) : R, temp_1 : C\}$. Lediglich i ist ein ABox-Individuum, aber sowohl i als auch $temp_1$ sind Knoten der ABox; i ist also alt, und $temp_1$ ist neu.

Eine Unterscheidung von neuen und alten Individuen ist zum einen zur Steuerung der Tableau-Expansion durch die Tableau-Regelanwendungsstrategie erforderlich. Zum anderen können in MIDELORA ABox-Individuen auch von einer anderen *Qualität* sein als neue Individuen. Betrachten wir erneut das Szenario einer räumlichen ABox. Hier sind die ABox-Individuen zugleich auch Instanzen räumlicher Datentypen in einem räumlichen Index. Hingegen wird die Geometrie eines während des Beweises neu erzeugten Individuums i.d.R. unspezifiziert bleiben. Somit kann keine Instanz eines räumlichen Datentyps angelegt werden. Dies spricht dafür, dass lediglich die alten Individuen Instanzen (einer Unterklasse) der Klasse Geometrischer-Substrat-Knoten (Geometric-Substrate-Node) (s. Abb. 5.10, S. 267) sind, nicht jedoch die neuen Individuen.

Diese Betrachtung führt uns zu den offensichtlich erforderlichen Klassen ABox, ABox-Knoten (ABox-Node) (Unterklasse von Knoten (Node)), ABox-Kante (ABox-Edge) (Unterklasse von Kante (Edge)) zur Darstellung von $\mathcal{A}$, V und E , und den Klassen ABox-Knoten-Label (ABox-Node-Label) (Unterklasse von Knoten-Beschreibung (Node-Description)) und ABox-Kanten-Label (ABox-Edge-Label) (Unterklasse von Kanten-Beschreibung (Edge-Description)) zur Darstellung von Beschreibungsinstanzen aus $\mathcal{L}_V$ und $\mathcal{L}_E$. Instanzen der Unterklasse ABox-Individuum (ABox-Individual) der Klasse ABox-Knoten (ABox-Node) repräsentieren ABox-Individuen; für diese können wir beispielsweise Räumlichkeit durch Nennung einer weiteren Oberklasse (in diesem Falle Geometrischer-Substrat-Knoten (Geometric-Substrate-Node)) implementieren. Neue Individuen sind hingegen lediglich Instanzen von ABox-Knoten (ABox-Node).

Wie in Kap. 4 zeichnen wir die Standard-Klassen für die Konstruktoren mittels überschriebener Methoden `get-standard-node-class`, `get-standard-edge-class`, `get-standard-node-description-class` und `get-standard-edge-description-class` aus.

Es ist als großer Vorteil der objektorientierten Tableau-Repräsentation zu sehen, dass beliebige Hilfs- und Verwaltungsinformationen stets objektlokal und somit *objektindiziert* abgespeichert und angefragt werden können. So ist beispielsweise das Ermitteln aller Konzept-Assertionen, die das Individuum i betreffen, eine zeitkonstante ($O(c)$) Operation. In einfacheren Implementationen würden Assertionen vielleicht durch Listen implementiert; dann hätte diese Operation bereits eine $O(n)$ -Komplexität. Ist die Tableau-Repräsentation nicht objektorientiert, und können Zusatz- und Verwaltungsinformation somit nicht objektlokal abgespeichert werden, so müssen diese Informationen in separaten Datenstrukturen konsistent mitverwaltet werden, was oftmals fehlerträchtig, speicheraufwendig und insbesondere auch unter dem Gesichtspunkt der Software-Wartbarkeit erfahrungsgemäß problematisch ist.

Natürlich hat die *schwergewichtige Repräsentation* auch Nachteile; hier ist zum einen der Speicherbedarf zu nennen. Zum anderen müssen Tableau-Datenstrukturen durch das Suchverfahren des Tableau-Beweisers nun entweder *destruktiv modifiziert* oder *kopiert* werden, wenn verschiedene Alternativen durch den Tableau-Beweiser zu betrachten sind (s. z.B. die $\sqcup$ -Regel aus Abb. 2.5, S. 105). Wird Rücksetzen (Backtracking) während der Suche erforderlich, so muss der frühere Zustand der zwischenzeitlich destruktiv modifizierten Objekte wieder hergestellt werden, was ebenfalls wieder destruktive Operationen erfordert. MIDeLoRA führt hierfür in der ABox eine sog. *Befehls-Historie*. Jede den Tableauezustand verändernde Aktion wird protokolliert. Wird Rücksetzen in einen früheren Zustand erforderlich, so wird die Historie zurückgerollt, und entsprechende *Kompensationsoperationen* für die Einträge in der Historie werden durchgeführt (sog. *Undo- oder Rollback-Operationen*). Wir kommen darauf

in Kap. 6.2.2.3 zurück.

Wichtige objektlokale Fokussierungs- und (vom Tableau-Beweiser benötigte) Steuerungsinformationen sind in Instanzen der Klasse ABox-Knoten (ABox-Node) durch Attribute gegeben:

- Zum einen erben diese Knoten die für die Klasse Knoten (Node) vorgesehenen Attribute und die mit ihnen verbundene Funktionalität: Wir hatten in Kap. 5.1.2.3 beschrieben, dass Knoten und Kanten beispielsweise als deaktiviert oder als gelöscht *markiert* werden können, und dass beliebige (auch rekursive) Markierungskontexte für Knoten und Kanten etabliert werden können, die dann von den Iteratoren genutzt werden.

Gelöschte, deaktivierte oder nicht dem aktuellen Markierungskontext entsprechende Knoten- und Kanten werden von den Iteratoren nicht mehr erfasst. So wird eine Fokussierung auf Teile bzw. Selektion von Teilen des Substrat-Graphen realisiert.

- Wie in Kap. 2.2.7.3, S. 108 erläutert, werden in Tableau-Verfahren für DLs mit transitiven Rollen (oder allgemeinen TBoxen) Blockierungsbedingungen erforderlich - ein Knoten kann bei Vorliegen einer Blockierungssituation einen anderen Knoten *blockieren*.

Wir sehen daher ein Attribut zur Referenzierung des *blockierenden Knotens*, sowie ein (listenwertiges) Attribut zur Referenzierung der durch diesen Knoten *blockierten Knoten* vor (evtl. blockiert ein Knoten mehr als einen anderen Knoten).

- Weitere knoten- bzw. kantenlokal günstig verwaltbare Informationen, z.B. die später diskutierten *Individuums-Modelle*, können ebenfalls stets automatisch objektindiziert in weiteren Attributen abgespeichert bzw. durch diese referenziert werden.

Für die Kanten bzw. Instanzen von ABox-Kante (ABox-Edge) sind sehr viel weniger Attribute erforderlich; dies liegt vor allem daran, dass eine Unterscheidung von expandierten und unexpandierten Rollen-Assertionen für die meisten DLs nicht erforderlich ist. Wir sehen aber auch hier vor, dass eine Kante als gelöscht oder deaktiviert markiert werden kann.

Die meiste Information über einen Knoten i bzw. eine Kante (i, j) ist in $L_V(i)$ bzw. $L_E((i, j))$ und somit durch die Instanz der entspr. Beschreibungsklasse ABox-Knoten-Label (Abox-Node-Label) bzw. ABox-Kanten-Label (ABox-Edge-Label) gegeben, die wir im Folgenden beschreiben.

6.2.2.2. Entwurf der Knoten- und Kanten-Label

Die Beschreibungsinstanz $L_V(i)$ nennen wir im Folgenden Knoten-Label, und $L_E((i, j))$ Kanten-Label. Das Knoten-Label ist eine Beschreibungsinstanz der Klasse ABox-Knoten-Label (Abox-Node-Label), und das Kanten-Label ist Beschreibungsinstanz der Klasse ABox-Kanten-Label (ABox-Edge-Label).

Konzept-Assertionen sind durch Elemente des Knoten-Labels, und Rollen-Assertionen durch Elemente des Kanten-Labels gegeben. Diese Elemente sind Konzepte bzw. Rollen. Wir schicken voraus, dass es sich hierbei ebenfalls um CLOS-Instanzen handelt. Dementsprechend existieren Klassen wie Konzept (Concept) und Rolle (Role), und entspr. Unterklassen wie Und-Konzept (And-Concept), etc.

Die initial in der ABox vorhandenen Assertionen hatten wir ABox-Assertionen genannt; mit Tableau-Assertionen sind hingegen stets alle Assertionen gemeint. Die Klassen ABox-Knoten-Label (Abox-Node-Label) und ABox-Kanten-Label (ABox-Edge-Label) sind so zu entwerfen, dass die Tableau-Regeln effizient ihre Anwendbarkeit prüfen und die durch sie herbeigeführten Tableau-Modifikationen effizient vornehmen können. Hierbei sind folgende Punkte zu berücksichtigen.

Die Tableau-Regeln müssen teilweise komplexe Bedingungen auswerten, um ihre Anwendbarkeit zu überprüfen. Die meisten Regeln können ihre Anwendbarkeit rein knotenlokal, also durch lokale Betrachtung des Labels und evtl. Betrachtung der Label von Nach- und Vorgängern überprüfen. Eine Regel darf nur dann angewendet werden, wenn ihre Nachbedingung nicht bereits erfüllt ist (denn andererseits könnte sie unendlich oft angewendet werden, und Terminierung wäre nicht mehr sichergestellt). So ist z.B. die $\sqcap$ -Regel aus Abb. 2.5, S. 105 für $i : C \sqcap D \in \mathcal{A}$ und daher $C \sqcap D \in L_V(i)$ nur dann anwendbar, wenn nicht bereits $\{i : C, i : D\} \subseteq \mathcal{A}$ bzw. $\{C, D\} \subseteq L_V(i)$ gilt. Andernfalls *expandiert* die Regel die Konjunktion, und setzt somit $\mathcal{A} := \mathcal{A} \cup \{i : C, i : D\}$ bzw. $L_V(i) := L_V(i) \cup \{C, D\}$.

Offensichtlich kann ein einmal expandiertes $\sqcap$ -Konzept als *expandiert* gekennzeichnet werden, und muss somit niemals wieder betrachtet werden, da die Regel auf diese Assertion nicht mehr angewendet werden kann. Sie wird erst wieder anwendbar, wenn $i : C$ oder $i : D$ aus dem Tableau entfernt wird, was nur durch ein Rücksetzen geschehen kann. Um die vergebliche erneute Überprüfung der Regelanwendbarkeit einzusparen, kann $i : C \sqcap D$ nach $\sqcap$ -Regelanwendung daher als *expandiert* gekennzeichnet werden. Tableau-Regeln werden daher nur *unexpandierte* Assertionen betrachten. Zur Überprüfung ihrer Anwendbarkeit muss eine Regel zudem das Elementproblem, hier $x : C \in \mathcal{A}$, $x : D \in \mathcal{A}$, effizient entscheiden können.

Zudem müssen die ABox-Assertionen von den Tableau-Assertionen unterschied-

den werden können. Konzeptionell ist daher L_V (für einen beliebigen Knoten i) wie folgt *partitioniert*: $L_V(i) = L_{V/told}(i) \cup L_{V/expanded}(i) \cup L_{V/unexpanded}(i)$. Diese Dreiteilung für ein Knoten-Label wird durch separate Attribute zur Repräsentation dieser drei Mengen umgesetzt; für ein Kanten-Label ist dies i.d.R. nicht notwendig (s.u.).

Da eine Tableau-Regel i.d.R. für einen DL-Konzeptkonstruktor definiert wird (z.B. die $\sqcap$ -Regel zur Behandlung von $\sqcap$ -Konzepten) und diese Regel daher lediglich sensitiv auf $\sqcap$ -Konzepte reagiert, steigert es die Performanz, wenn die Menge $L_{V/unexpanded}(i)$ weiter in *konzeptspezifische* Mengen partitioniert wird. So kann z.B. die $\sqcap$ -Regel ihr Anwendbarkeit ausschließlich durch Betrachtung von Konzepten in der Menge $L_{V/unexpanded,\sqcap}(i)$ überprüfen.

Zur Illustration der im Laufe eines Beweises durch Regel-Anwendungen vorgenommenen Label-Veränderungen sei wieder die Eingabe-ABox $\{i : C \sqcap D\}$ betrachtet:

1. Die initiale ABox entspricht einer Instanz der Klasse ABox mit einem Knoten i , sodass $L_V(i) = L_{V/told}(i) = \{C \sqcap D\}$, $L_{V/expanded}(i) = \emptyset$, und $L_{V/unexpanded}(i) = \emptyset$ gilt. Die Assertion $i : C \sqcap D$ ist also eine ABox-Assertion und somit „Told Information“ bzw. „Asserted Information“.
2. Wird nun der ABox-Erfüllbarkeitsbeweiser gestartet, so wird in einer *Vorbereitungsphase* zunächst für alle vorhandenen Knoten $i \in V$: $L_{V/unexpanded}(i) := L_{V/told}(i)$ gesetzt. Die Assertion $i : C \sqcap D$ wird somit zur unexpandierten Tableau-Assertion.

Das Hinzufügen eines Konzeptes zu $L_{V/unexpanded}(i)$ führt automatisch zum Einfügen in die „richtige Partition“ von unexpandierten Konzepten; daher gilt $L_{V/unexpanded,\sqcap}(i) = \{C \sqcap D\}$ nach der Vorbereitungsphase.

Dann werden die Regeln ihre Anwendbarkeit stets bzgl. der Menge $L_{V/unexpanded}(i)$ prüfen, solange, bis $L_{V/unexpanded}(i) = \emptyset$ für alle i .

3. Das Anwenden einer Regel führt i.d.R. zum Überführen eines Konzeptes C aus $L_{V/unexpanded}(i)$ in $L_{V/expanded}(i)$, wobei die entsprechende Partition anhand der Klasse von C gewählt wird. Wie $L_{V/unexpanded}$, so ist auch $L_{V/expanded}$ nach Konzepttyp partitioniert. Einzige Ausnahme bilden die universellen Rollenrestriktionen ($\forall R.C$ -Konzepte) und die nach oben begrenzenden Anzahlrestriktionen ($(\leq_n R)$ - und $(\leq_n R.C)$ -Konzepte), die stets unexpandiert bleiben.

Durch Anwendung der $\sqcap$ -Regel wird $L_{V/unexpanded,\sqcap}(i) := L_{V/unexpanded,\sqcap}(i) \setminus \{C \sqcap D\}$, $L_{V/expanded,\sqcap}(i) := L_{V/expanded,\sqcap}(i) \cup \{C \sqcap D\}$ gesetzt. Die $\sqcap$ -Regel fügt zudem die beiden Konjunkte als unexpandierte Konzepte zum Label hinzu – sind C und D beispielsweise atomare Konzepte, so wird $L_{V/unexpanded,positive\_atoms}(i) := L_{V/unexpanded,positive\_atoms}(i) \cup \{C, D\}$ gesetzt.

Nun gilt $L_{V/unexpanded}(i) = \emptyset$. Da kein Widerspruch aufgetreten ist, haben wir eine Vervollständigung konstruiert, und der Beweiser terminiert mit der Rückgabe von TRUE; die ABox $\{i : C \sqcap D\}$ ist also erfüllbar.

Für die Klasse ABox-Knoten-Label (Abox-Node-Label) sehen wir daher Attribute zur Speicherung der told-concepts, expanded-concepts und unexpanded-concepts vor. Wir wollen die Mengenrepräsentation hier nicht festlegen; daher werden Zugriffsmethoden und -iteratoren benötigt (has-unexpanded-concepts-p, loop-over-unexpanded-concepts). Um die Mengen wie besprochen klein zu halten, werden die entspr. Mengen jeweils noch nach Konzeptklasse partitioniert; dies führt uns zu den Attributen expanded-and-concepts, expanded-or-concepts, etc., und analog für unexpanded-concepts. Zur *Verwaltung* der entspr. Mengen, also zum Hinzufügen und Löschen von Elementen, werden ebenfalls entspr. Methoden erforderlich: add-to-unexpanded, move-from-unexpanded-to-expanded, usw.

Eine weitere Gruppe von Attributen wird für Verwaltungsinformationen für das *dependenzgesteuerte Rücksetzen* (s.u.) benötigt. Für jede Assertion müssen *Dependenzinformationen* verwaltet werden. Anhand dieser können für eine Assertion all die Assertionen identifiziert werden, die für das Vorhandensein der Assertion auf dem Tableau (mit)verantwortlich sind. Konzeptionell handelt es sich um eine Funktion $\text{deps} : \mathcal{A} \mapsto 2^{\mathcal{A}}$. Da Assertionen nicht als eigenständige Datenstrukturen bzw. Objekte repräsentiert werden, kann diese Information nicht assertionslokal gespeichert werden; stattdessen muss sie knoten- bzw. kantenlokal gespeichert werden.⁵

Diese Dependenzinformation ist in erster Linie erforderlich, damit im Falle eines im Tableau erschienenen Widerspruchs entschieden werden kann, durch welche Assertionen dieser (mit)verursacht wurde. Um den Widerspruch zu beseitigen, muss zu einem Verzweigungspunkt (einem sog. „Oder-Knoten“) im Suchbaum zurückgesetzt werden, in dem eine den Widerspruch (mit)verursachende Assertion auf das Tableau gelegt wurde. Daher ist es zur Realisierung des dependenzgesteuerten Rücksetzens ausreichend, wenn lediglich die sog. *Verzweigungspunkt-IDs* (*Choice Point IDs*) einer Assertion verwaltet und identifiziert werden können. Eine Verzweigungspunkt-ID identifiziert eindeutig einen Verzweigungspunkt im Suchbaum. Mittels der Funktion $\text{choice\_points} : \mathcal{A} \mapsto 2^{\mathbb{N}}$ können die Verzweigungspunkte-IDs einer Assertion ermittelt werden, sodass ein Zurücksetzen zu einem entspr. Verzweigungspunkt veranlasst werden kann.

⁵Es ist nicht möglich, diese Dependenzinformation in den Instanzen der Klasse Konzept (Concept) abzulegen, da Konzept-Instanzen eine *Wertsemantik* haben. Dementsprechend existiert für jedes DL-Konzept nur eine Instanz. Eine best. Konzept-Instanz kann daher in mehreren Knoten-Labeln vorkommen, sodass die Instanz letztlich verschiedene Assertionen mit unterschiedlichen Abhängigkeiten repräsentiert.

Die Abhängigkeitsinformation ist prinzipiell auch durch die Befehlshistorie explizit; letztlich identifiziert auch ein Befehlsobjekt in der Historie eindeutig einen Zustand im Suchbaum und somit einen Tableau-Zustand.

Die Klasse ABox-Kanten-Label (ABox-Edge-Label) ist wesentlich einfacher als die Klasse ABox-Knoten-Label (ABox-Node-Label), da hier nur die zusätzliche Abhängigkeitsinformation mitrepräsentiert werden muss.

Auf den ersten Blick erscheint es nachteilig, dass in der MIDELoRA-Architektur Knoten (bzw. Kanten) und Beschreibungen voneinander getrennt sind. Schließlich werden so für jeden Knoten (bzw. jede Kante) zwei CLOS-Instanzen erforderlich: Eine für i (bzw. (i, j)), und eine für $L_V(i)$ (bzw. $L_E((i, j))$). Zum einen respektiert diese Trennung jedoch die Basis-Prinzipien des Substrat-Datenmodells, auf dem wir hier aufbauen. Zum anderen kann durch die Separierung prinzipiell auch ein *Speicherspareffekt durch „Structure Sharing“* ermöglicht werden, wenn Knoten-Label mit einer Wert-Semantik (im Unterschied zu einer Objekt-Semantik) betrachtet werden. So wäre es möglich, dass Knoten mit gleichen Labeln *die selbe ABox-Knoten-Label (ABox-Node-Label)-Instanz referenzieren*. Dies erfordert dann aber, dass die *knotenspezifischen Anteile aus den Label-Instanzen* in die Knoten-Instanzen verlagert werden (insbesondere die ja knotenlokalen Abhängigkeitsinformationen). Aus diesem Grund sind die Abhängigkeitsinformationen als Attribute der Klasse ABox-Knoten (ABox-Node) und nicht als Attribute der Klasse ABox-Knoten-Label (ABox-Node-Label) vorzusehen (und analog für die Kanten).

6.2.2.3. Entwurf der Befehlshistorie

Die Zustände der Substrat-Instanzen (zur Darstellung von Knoten, Kanten, sowie von Knoten- und Kanten-Labeln) werden während des Beweises destruktiv durch Regelanwendungen verändert. Instanzen werden zudem dynamisch erzeugt, miteinander in Beziehung gesetzt, und auch wieder gelöscht. Die Indexstrukturen der ABox müssen dementsprechend mitverwaltet werden. Das *Wiederherstellen eines früheren Zustands*, wie es für das Rücksetzen im Rahmen der Suche erforderlich ist, wird somit nur durch entsprechende *Kompensationsoperationen* ermöglicht, die letztlich die *inversen Operationen* ausführen. Die alternative Idee, den Beweiser mit *Kopien von Teilen des Objektgraphen* arbeiten zu lassen, wurde durch erste Experimente schnell als hoffnungslos speicher- und laufzeitineffizient enttarnt.

Wir verwenden daher das *Befehlshistorie*-Entwurfsmuster [Gamma u. a. 1996]. Hier wird jede die ABox modifizierende Operation in einem *Befehlsobjekt* mitprotokolliert und in eine *Befehlshistorie* aufgenommen. Wesentlich ist, dass *alle* Zustandsveränderungen adäquat verzeichnet werden können. Wird beispielsweise das Konzept C zu $L_V/unexpanded(i)$ hinzugefügt, so muss dies durch ein

Befehlsobjekt in der Historie protokolliert werden.

Ein Befehlsobjekt ist Instanz der (abstrakten und „konzeptuellen“, s.u.) Klasse Aktion (Action). Die Befehlshistorie sehen wir als doppelt verkettete Liste vor, um effizientes Navigieren von einem Befehlsobjekt aus in beide Richtungen zu ermöglichen. Das Befehlsobjekt protokolliert die ausgeführte Operation und sollte leichtgewichtig sein, da Beweise (und somit Historien) sehr lang werden können (daher handelt es sich bei Aktion (Action) auch nur um eine „konzeptuelle“ Klasse). Da es verschiedene Arten von ABox-verändernden Operationen gibt, sehen wir zudem *Befehlsklassen* vor, die Unterklassen von Aktion (Action) sind. Eine ABox referenziert mittels *first-action* das erste, sowie mittels *current-action* das aktuelle Befehlsobjekt der Befehlshistorie.

Soll nun (aus welchen Gründen auch immer) ein früherer Zustand der ABox wiederhergestellt werden, so wird die Befehlshistorie dieser ABox beim aktuellen Zustand (*current-action*) beginnend bis zu dem gewünschten Zustand mittels der Methode *rollback-to* „zurückgerollt“. Der gewünschte wiederherzustellende Zustand bzw. das diesem Zustand entsprechende Befehlsobjekt wird der Methode als Argument übergeben. Die *rollback-to*-Methode verfolgt dann die Befehlshistorie rückwärts bis zu diesem Befehlsobjekt und wendet auf jedes so besuchte Befehlsobjekt die *undo*-Methode an. Die *undo*-Methode wird befehlsklassenspezifisch implementiert.

Somit wird nicht nur das Tableau vollständig explizit durch Datenstrukturen repräsentiert, sondern auch der Berechnungs- bzw. Beweisprozess durch die Befehlshistorie – die Befehlshistorie kann als vollständige, explizite Repräsentation des bisher abgelaufenen Beweises angesehen werden.

6.2.2.4. Der MiDeLoRA-Kern

Die korrekte Verwaltung des Tableaus ist offensichtlich kompliziert. Beispielsweise erfordert das Expandieren von $C \sqcap D \in L_V / \text{unexpanded}(i)$ das „Verschieben“ von $C \sqcap D$ von *unexpanded-and-concepts* nach *expanded-and-concepts*, und zudem muss ein Befehlsobjekt in die Befehlshistorie eingetragen werden, das diese Operation protokolliert. Noch komplexer sind die Tableau-Verwaltungsoperationen, die beispielsweise bei der Expansion von $\exists R.C \in L_V / \text{unexpanded}(i)$ erforderlich werden, da hier sogar ein neuer Knoten und eine neuen Kante erzeugt werden müssen. Die Expansion löst eine ganze Kaskade von Tableau-Veränderungen aus und erzeugt mehrere Befehlsobjekte.

Um von dieser Komplexität zu abstrahieren, stellt der MiDeLoRA-Kern *semantisch höherwertige Tableau-Verwaltungsoperationen* bereit. Der MiDeLoRA-Kern ist somit eine Art „virtuelle Maschine“ zur Verwaltung des Tableaus. Durch diese Abstraktionen können die Tableau-Regel-Implementationen konzeptuell einfach und nahe an ihren mathematischen Vorlagen gehalten werden

– MiDeLoRA-Tableau-Regel sollen möglichst deklarativ formulierbar sein (dies gilt auch für die anderen durch MiDeLoRA bereitgestellten Abstraktionen).

Der Kern soll daher eine assertionsbezogene Schnittstelle (auf der Wissens- bzw. „Tell & Ask“-Ebene) bereitstellen, da eine solche Schnittstelle den mathematischen Tableau-Regeldefinitionen am ehesten entspricht. Zum Hinzufügen einer Assertion zum Tableau durch eine Tableau-Regel sollte es aus Sicht der Tableau-Regel ausreichend sein, lediglich eine Methode wie `register-as-unexpanded` zu rufen, der im Falle einer Konzept-Assertion ein Konzept und ein Knoten, und im Falle einer Rollen-Assertion eine Rolle und eine Kante übergeben wird, plus optionaler Abhängigkeitsinformationen. Ist die Wahl der Assertion zudem nichtdeterministisch, so muss ein entspr. Verzweigungspunkt („Choice Point“) übergeben werden. Die entspr. Methoden haben daher folgende Signaturen:

```
(defmethod register-as-unexpanded ((node abox-node) (concept concept)
                                   &key new-choice-point depends-on ...)
  ...)

(defmethod register-as-unexpanded ((edge abox-edge) (role role)
                                   &key new-choice-point depends-on ...)
  ...)
```

Der Kern kümmert sich dann um alles Weitere (korrekte Verwaltung der Label, der Abhängigkeiten, der Befehlshistorie). Zudem sind die Methoden

- `register-as-expanded`,
- `register-additional-dependencies`

definiert. Diese höherwertigen Methoden stellen somit die konsistente Verwaltung des Tableaus sicher. Zur Verwaltung des Tableaus muss der Kern verschiedene Aufgaben erfüllen, die sich in vier Gruppen aufteilen lassen:

1. Verwaltung und Zugriff auf Knoten und Kanten,
2. Verwaltung der Knoten- und Kantenlabel,
3. Verwaltung der Abhängigkeiten,
4. Verwaltung der Befehlshistorie.

Diese Aufgabengruppen beschreiben und entwerfen wir nun detaillierter, auch wenn einiges davon schon klar wurde.

6.2.2.4.1. Verwaltung und Zugriff auf Knoten- und Kanten Die prinzipielle Nutzung der Dienste des Substrat-Rahmenwerks wurde hinreichend deutlich.

Die *Verwaltung* umfasst die Erzeugung und Löschung von Tableau-Knoten und -Kanten. Damit die ABox- bzw. Tableau-Perspektive auf den Substrat-Graphen im Quellcode der MIDeLoRA-Beweiser explizit gemacht wird, stellen wir adäquat benannte Methoden bereit, beispielsweise `create-abox-node`, `delete-abox-node` statt `create-node`, `delete-node`, und analog für die Kanten. Diese Methoden erzeugen protokollierende Befehlsobjekte und verwalten die Abhängigkeitsinformationen entsprechend. So wird `create-abox-node` nicht nur ein Tableau-Individuum mittels `create-node` erzeugen, sondern auch ein entspr. Befehlsobjekt anlegen.

Zum Zugriff auf Knoten und Kanten nutzen wir die bereitgestellten Iteratoren und mengenwertigen Schnittstellen, die wir wiederum adäquat umbenennen und zudem so verfeinern, dass sie die Spezifika der Tableau-Individuen berücksichtigen:

- Ein Knoten kann einen anderen Knoten *blockieren*: diese Assoziation wird durch entsprechende Attribute verwaltet. Entsprechende Prädikate `is-blocking-node-p`, `blocking-node-for`, `is-blocked-node-p` und Methoden `get-blocking-node`, `get-blocked-nodes` sind vorgesehen. Ein blockierter Knoten wird dann normalerweise nicht mehr von den Iteratoren betrachtet.
- Knoten können miteinander *verschmolzen* werden (Methode `merge-nodes`) und wieder entschmolzen werden (`unmerge-nodes`). Der Kern kann die in einem Cluster verschmolzenen Knoten so behandeln, als würde es sich um einen einzelnen Knoten handeln. Ein einzelner Knoten im Cluster wird als *Repräsentant* des Clusters gewählt (s. Attribut `representative-node-p`); die anderen Knoten im Cluster werden dann durch die Zugriffsmethoden und -Iteratoren nicht mehr geliefert.

Die entspr. Iteratoren heißen statt `loop-over-nodes` dann `loop-over-abox-nodes`, statt `loop-over-node-successors` dann `loop-over-role-successors`, und analog für die Kanten. Die mengenwertigen Methoden der Schnittstelle werden ebenfalls entspr. umbenannt, z.B. wird `get-nodes` zu `get-abox-nodes`, usw.

Die Regeln benötigen oft den Zugriff auf den jüngsten oder ältesten Knoten, um die Expansionsstrategie eines Tableau-Kalküls korrekt implementieren zu können. Hierzu sind Methoden der Art `get-youngest-node` und `get-oldest-node` sowie Verwandte vorgesehen.

6.2.2.4.2. Verwaltung der Knoten- und Kantenlabels Die Knotenlabel $L_V/told$, $L_V/unexpanded$, $L_V/expanded$ werden durch entspr. Attribute der Klasse ABox-Knoten-Label (Abox-Node-Label) dargestellt. Zur Verwaltung dieser Mengen sind entsprechende Methoden für ABox-Knoten-Label (Abox-Node-Label) $\times$ Konzept

(Concept) definiert: `add-to-expanded`, `add-to-unexpanded`, `delete-from-expanded`, `delete-from-unexpanded`, `move-from-unexpanded-to-expanded`, und `move-from-expanded-to-un-expanded`. Eine `move`-Methode delegiert an die entsprechenden `add`- und `delete`-Methoden zur Erfüllung ihrer Aufgaben.⁶ Diese Methoden sind auch für ABox-Knoten (ABox-Node) definiert (per Delegation).

Damit die Tableau-Regeln ihre Anwendbarkeit prüfen können, sind Prädikate zur Prüfung auf Leerheit der entsprechenden Konzeptmengen des Knotenlabels vorgesehen: `has-unexpanded-and-concepts-p`, `has-unexpanded-or-concepts-p`, etc. Wiederum sind diese Methoden nicht nur für Label, sondern auch für Knoten definiert. Eine Reihe höherwertigerer Prädikate ist vorhanden; so testet `complete-p` z.B., ob überhaupt noch unexpandierte Konzepte vorhanden sind.

Da die Mengendarstellung austauschbar sein soll, werden Iteratoren und mengen- bzw. listenwertige Zugriffsmethoden benötigt und bereitgestellt: `loop-over-node-unexpanded-atomic-concepts`, `loop-over-node-unexpanded-and-concepts`, `get-unexpanded-atomic-concepts`, etc. Wiederum sind diese Iteratoren und Methoden sowohl für Beschreibungsinstanzen (Label) als auch für Knoten definiert.

6.2.2.4.3. Verwaltung von Abhängigkeiten bzw. Verzweigungspunkten Für das dependenzgesteuerte Rücksetzen müssen Abhängigkeiten für die Assertionen verwaltet werden. Wir hatten in Kap. 6.2.2.2, S. 420 erläutert, dass hierfür konzeptionell die Funktion `deps` bzw. die „schwächere Version“ `choice_points` für Assertionen definiert ist. Die Verzweigungspunkt-IDs einer Konzept-Assertion werden knotenlokal im Attribut `choice-points` gespeichert (und analog für die Kanten). Die Funktion `choice_points` wird durch die für Paare vom Typ ABox-Knoten (ABox-Node) $\times$ Konzept (Concept) und ABox-Kante (ABox-Edge) $\times$ Rolle (Role) definierte Methode `get-choice-points` implementiert. Diese liefert bei Aufruf die Liste der Verzweigungspunkt-IDs aller Assertionen, von denen die fragliche Konzept- bzw. Rollen-Assertion abhängt.

Fügt eine Regel nun durch Feuern neue Assertionen zum Tableau hinzu, so wird beim Registrieren einer neuen Assertion mittels `register-as-unexpanded` über das Argument `depends-on` spezifiziert, von welchen anderen Assertionen sie abhängt. Mittels `get-choice-points` werden die Verzweigungspunkt-IDs dieser Assertionen vom Kern angefordert. Diese werden vereinigt und als Menge der Verzweigungspunkt-IDs für die neue Assertion vermerkt. Wurde die Assertion zudem von einer nichtdeterministischen Regel erzeugt, so muss auch ein neuer Verzweigungspunkt angefordert werden (s. optionales Argument `new-choice-points` in `register-as-unexpanded`).

⁶Die Methode `move-from-expanded-to-unexpanded` wird lediglich zum Zurückrollen des Tableaus benötigt.

6.2.2.4.4. Verwaltung der Befehlshistorie Jede das Tableau verändernde Operation wird durch ein Befehlsobjekt protokolliert. Wird z.B. ein Konzept zur Liste der unexpandierten Konzepte eines Knotens durch `register-as-unexpanded` hinzugefügt, so wird vom Kern automatisch ein protokollierendes Befehlsobjekt vom Typ `Registrierte-Unexpandiert-Aktion` (`Put-To-Unexpanded-Action`) angelegt und als letztes Element in die Befehlshistorie eingereiht (aufgrund der `current-action`-Referenz ist dies nicht aufwendig). Ein Befehlsobjekt hat eine eindeutige Identität und verweist in der Befehlshistorie auf das vorherige und nachfolgende Befehlsobjekt (doppelt verkettete Liste). Die Befehlshistorie wächst während der Tableau-Expansion, bis Zurückrollen mittels der Methode `rollback-to` angefordert wird. Die Befehlshistorie ist ABox-spezifisch – jede ABox hat eine Befehlshistorie.

Konzeptuell sind Befehlsobjekte Instanzen (einer Unterklasse) der Klasse `Aktion` (`Action`) (wir sagen „konzeptuell“, da es sich aus Speicherplatzgründen nicht um CLOS-Instanzen, sondern leichtgewichtige Record-Strukturen handelt, s.u.). Wir sehen folgende Typen von Befehlsobjekten bzw. Unterklassen von `Aktion` (`Action`) vor:

- **Constraint-Aktion** (`Constraint-Action`): Instanzen dieser Klasse protokollieren elementare Veränderungen am Tableau:
 - **Verschiebe-Von-Unexpandiert-Nach-Expandiert-Aktion** (`Move-From-Unexpanded-To-Expanded-Action`): Wird zur Protokollierung eines `move-from-unexpanded-to-expanded`-Methodenaufrufs (s.o.) angelegt.
 - **Registrierte-Unexpandiert-Aktion** (`Put-To-Unexpanded-Action`): Wird zur Protokollierung eines `add-to-unexpanded`-Methodenaufrufs (s.o.) angelegt.
 - **Registrierte-Zusätzliche-Dependenzen-Aktion** (`Register-Additional-Dependencies-Action`): Wird zur Protokollierung eines `register-additional-dependencies`-Methodenaufrufs angelegt.
- **Objekt-Aktion** (`Object-Action`): Instanzen dieser Klasse protokollieren Objekterzeugungsaktionen (Erzeugung von Knoten und Kanten) sowie Zustandsänderungen von Objektattributen, die nicht zum Knotenlabel gehören:
 - **Erzeuge-Knoten-Aktion** (`Create-Node-Action`): Wird zur Protokollierung eines `create-node`-Methodenaufrufs angelegt.
 - **Erzeuge-Kanten-Aktion** (`Create-Edge-Action`): Wird zur Protokollierung eines `create-edge`-Methodenaufrufs angelegt.
 - **Verändere-Zustand-Aktion** (`Change-State-Action`): Wird zur Protokollierung einer semantikkrelevanten Zustandsänderung eines Tableau-Objektes angelegt, z.B. wenn ein Knoten als *deaktiviert* oder *gelöscht*

markiert wird. Offensichtlich ist es im Falle eines Zurückrollens erforderlich, dass auch diese Objektattribute wieder ihre alten Werte erhalten, bzw. wieder hergestellt werden.

Jede Klasse stellt eine entspr. undo-Methode bereit. Offensichtlich „weiß“ so jedes Befehlsobjekt, wie es seine Auswirkungen, die es auf das Tableau hatte, wieder rückgängig machen kann. Die Methode *rollback-to* der Klasse ABox hatten wir bereits erläutert.

Befehlsobjekte protokollieren stets das beeinflusste Objekt (Attribut bzw. Assoziation *affected-object*), und vermerken die Vorbedingungs- und Nachbedingungs-Assertionen; hierbei handelt es sich um Listen von Paaren der Form ABox-Knoten (ABox-Node) $\times$ Konzept (Concept) bzw. ABox-Kante (ABox-Edge) $\times$ Rolle (Role).

6.2.2.5. Klassendiagramm des Basis-Entwurfs

In Abb. 6.2 und Abb. 6.3 ist der Entwurf der Basis-Klassen des MIDeLoRA-Systems dargestellt. Für das UML-Diagramm gelten die bereits für Abb. 5.2, S. 247 gemachten Aussagen.

Folgende Punkte sind beachtenswert:

- Dependenzinformation werden nicht in den Labeln, sondern in den Knoten und Kanten selbst gespeichert, sodass wie erläutert prinzipiell *Structure Sharing* von Labeln ermöglicht wird.

Die Dependenzinformation bzw. die diskutierte Funktion *choice_points* wird durch das Attribut *choice-points* bzw. die Methoden *get-choice-points* bereitgestellt.

- Die Klasse ABox enthält Attribute und Methoden, die sich auf die Befehlshistorie beziehen (s. Assoziationen bzw. Attribute *current-action*, *first-action*).

Mittels *first-action* hält die ABox eine Referenz auf das erste Element der Befehlshistorie. Das *aktuelle Befehlsobjekt* wird durch *current-action* referenziert. Zur Nummerierung der Verzweigungspunkte und der Befehlsobjekte werden entspr. Zähler benötigt (*choice-point-counter*, *action-counter*). Das aktuelle Befehlsobjekt ist charakteristisch für den aktuellen Zustand. Um ein (durch das Suchverfahren durch Rücksetzen initiiertes) Wiederherstellen eines früheren Zustands der ABox zu ermöglichen, ist die *rollback-to*-Methode vorgesehen. Der gewünschte wiederherzustellende Zustand wird durch ein für diesen Zustand charakteristisches Aktionsobjekt spezifiziert.

- Ein MIDeLoRA-ABox-Beweiser wird in drei operationale Phasen aufge-

6.2 Entwurf von MIDeLoRA

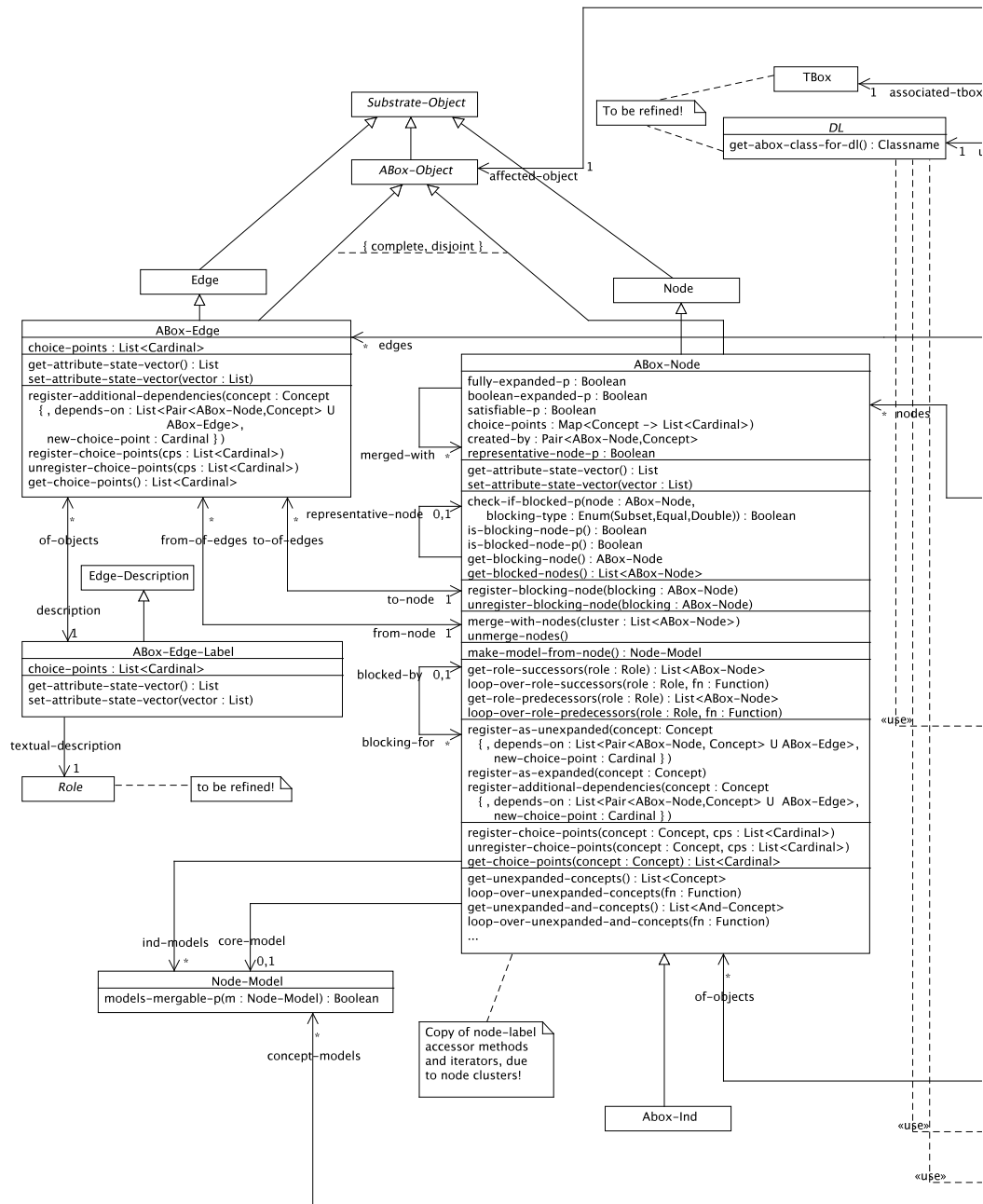


Abbildung 6.2.: Basis-Entwurf von MIDeLoRA (linker Teil).

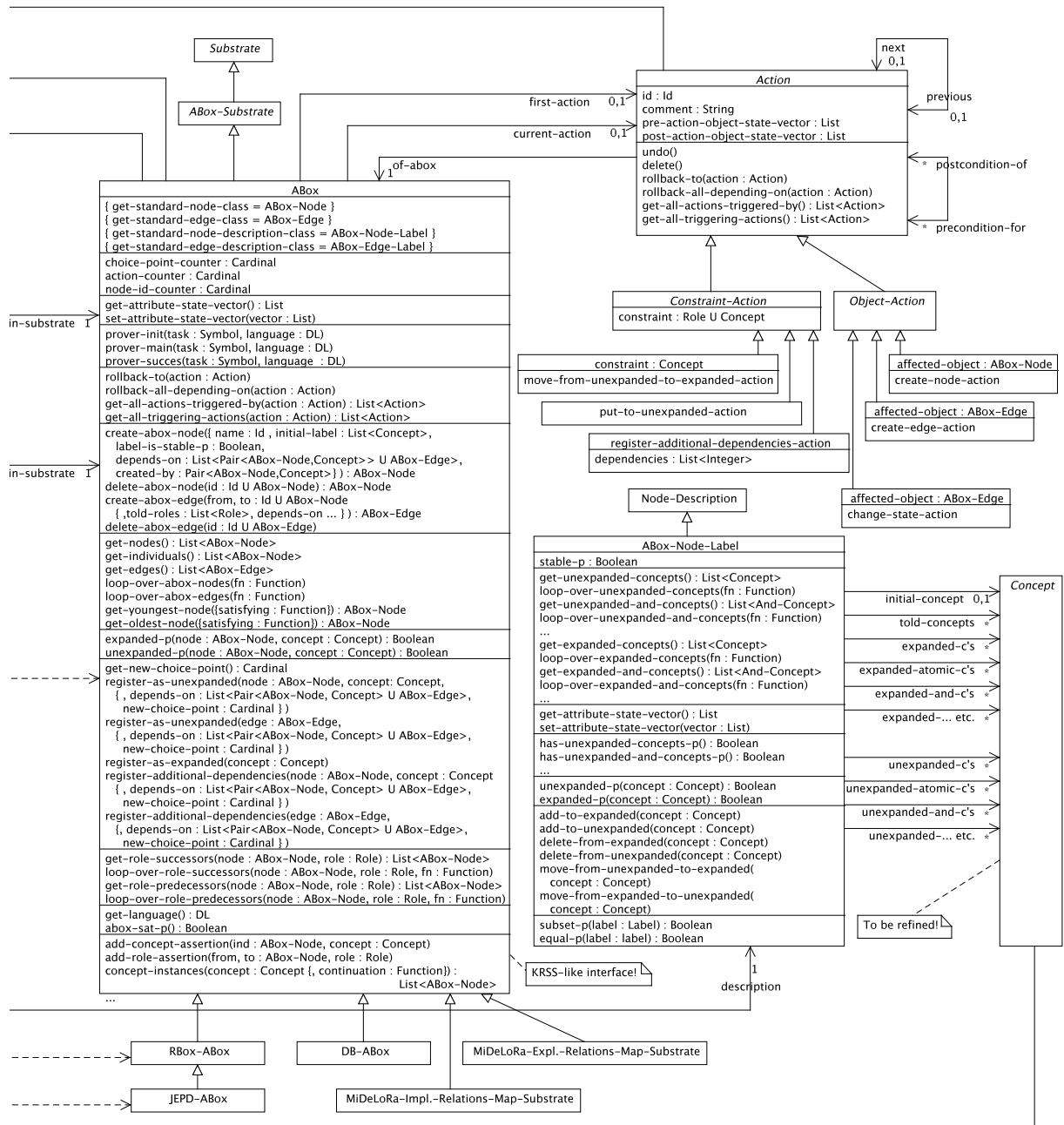


Abbildung 6.3.: Basis-Entwurf von MiDeLoRA (rechter Teil).

teilt, die den Methoden `prover-init`, `prover-main`, `prover-success` der Klasse `ABox` entsprechen. Wir erläutern diese Phasenstruktur weiter unten.

- Die Klasse `ABox` stellt eine Reihe von KRSS Tell-&-Ask-Methoden zur Population bzw. Befragung der `ABox` bereit (s. z.B. `add-concept-assertion`).

6.2.3. Potentiale der MiDeLoRA-Architektur

Wir hatten bereits in der Analyse den Entwurf so vorbereitet, dass er u.a. die leitdomänengetriebenen Anforderungen umsetzt, von Aspekten der physischen Tableau-Repräsentation abstrahiert, usw. Wir wollen die durch die MiDeLoRA-Architektur eröffneten Potentiale im Folgenden noch einmal explizit zusammenstellen und diskutieren. Wir vergleichen die MiDeLoRA-Architektur zudem mit einem *einfachen, rein funktionalen ALC-Beweiser*, der als prototypisch für konventionellere DL-Systeme angesehen wird, um die Unterschiede in der Architektur zu einfacheren Entwürfen deutlich zu machen. Zudem erhöhen wir so die Verständlichkeit der nachfolgenden Darstellung.

6.2.3.1. Reichhaltige ABoxen und virtuelle Assertionen

Um Variabilität bzgl. der `ABox`-Klasse, also bzgl. der $\mathcal{K}$ -Dimension im MiDeLoRA-Raum zu erhalten, sollten *wiederverwendbare* Tableau-Regeln so gestaltet werden, dass sie von der Speicherdarstellung der `ABox` abstrahieren; sie sollten also nur über die Iteratoren bzw. listenwertigen Zugriffsmethoden auf die `ABox` zugreifen. So wird es aus Sicht einer wiederverwendbaren Tableau-Regel irrelevant sein, ob eine `ABox` Instanz der Klasse `MiDeLoRA-Implicite-Relationen-Karten-Substrat` (`MiDeLoRA-Implicit-Relations-Map-Substrate`) oder Instanz der Klasse `DB-ABox` ist.

Wie können nun die von einer reichhaltigen `ABox` gebotenen virtuellen Assertionen (s. Kap. 6.1.6, S. 409) realisiert werden? Für alle virtuellen Assertionen, deren Knoten bzw. Kanten bereits explizit vorhanden sind, ist dies kein Problem. Hier muss lediglich die Label-Instanz entsprechend (berechnungsgesteuert anhand der Reichhaltigkeit) modifiziert bzw. um zusätzliche Konzepte oder Rollen bei Bedarf erweitert werden. Dies betrifft die Fälle **V2** und **E2**.

Sind jedoch die Knoten bzw. Kanten selbst virtuell und daher nicht explizit, so müssen Knoten bzw. Kanten u.U. (inkl. entspr. Label) von den Iteratoren *materialisiert* werden. Letztlich hängt es vom anfordernden Algorithmus bzw. Beweiser ab, ob notwendigerweise eine Instanz zum Weiterrechnen benötigt wird, oder ob nicht bereits ein *Name* für den Knoten bzw. die Kante ausreichend ist. In diesem Fall kann auf eine Materialisierung verzichtet werden.

Die bereitgestellten Standard-Tableau-Regeln (s.u.) werden in jedem Fall materialisierte Knoten benötigen, da andernfalls die benötigten Verwaltungsinformationen zur Steuerung der Expansion fehlen. Das prinzipielle Problem, dass die DB-ABox mehr virtuelle Knoten enthalten kann, als zu einem Zeitpunkt in den Hauptspeicher passen, kann durch *ausschnittsweise Betrachtung bzw. Fokussierung der Iteratoren* gelöst werden; hierzu müssen die materialisierten Knoten (ähnlich den Seiten einer virtuellen Speicherverwaltung) u.U. wieder aus dem Hauptspeicher entfernt bzw. dematerialisiert werden. Dies ist sicherlich möglich, wenn feststeht, dass ein Knoten für den weiteren Beweisprozess nicht mehr benötigt wird. Leider kann diese Entscheidung oftmals nicht automatisch getroffen werden (s. Kap. 6.2.3.2). Muss daher ein zuvor materialisierter, dann dematerialisierter virtueller Knoten nun erneut wieder „eingelagert“ bzw. materialisiert werden, so müssen auch seine Verwaltungsstrukturen wieder hergestellt werden. Diese müssen beim Dematerialisieren daher ebenfalls in die Datenbank gesichert werden. Zwar haben wir dies im Rahmen der Arbeit nicht weiter untersucht, doch bietet MiDeLoRA prinzipiell die technischen Voraussetzungen hierfür.

Analoges gilt für die virtuellen Kanten. Hier gelten prinzipiell die gleichen Aussagen wie für die virtuellen Knoten; da für die Rollen-Assertionen jedoch weniger Verwaltungsinformationen benötigt werden, ist die Wahrscheinlichkeit höher, dass der anfordernde Algorithmus auf das Materialisieren der Kante verzichten kann. Für Algorithmen, die beispielsweise Model Checking realisieren – wie beispielsweise die ABFen der virtuellen RCC-Atome – kann auf ein Materialisieren dann verzichtet werden. Auf ein Materialisieren der Kanten kann auch für die Standard-Tableau-Regeln für Standard-DLs ohne Anzahlrestriktionen verzichtet werden, da diese Regeln lediglich die Rollennachfolger und -vorgänger eines Knotens anfordern bzw. benötigen, nicht jedoch die Kante selbst. Die Kanten-Instanz selbst wird erst aufgrund einer angewendeten Optimierungstechnik für Anzahlrestriktionen erforderlich, da für diese Optimierungstechnik zusätzliche Verwaltungsinformation (in einem Attribut der Kanten-Instanz) gespeichert werden muss.

Zur Illustration des Verhaltens der Iteratoren bzw. zur Verdeutlichung, wie durch sie die physische Darstellungsunabhängigkeit realisiert wird, betrachten wir eine Instanz der Klasse DB-ABox und eine Instanz der Klasse MiDeLoRA-Implicite-Relationen-Karten-Substrat (MiDeLoRA-Implicit-Relations-Map-Substrate):

1. Für die DB-ABox werden die Iteratoren (`loop-over-abox-nodes, ...`) und die Methoden der mengenwertigen Schnittstelle (`get-nodes, ...`) so gestaltet, dass zunächst die expliziten Knoten geliefert werden.

Die Methoden können durch entspr. Argumente so parametrisiert werden, dass sie, sobald die explizit vorhandenen Knoten geliefert wurden, automatisch auch die Datenbank befragen. Die Iteratoren bzw. mengenwertigen

gen Schnittstellenmethoden setzen dann automatisch eine SQL-Anfrage ab. Die so ermittelten Knoten können dann entweder materialisiert oder (der obigen Diskussion entsprechend) lediglich namentlich zurückgegeben werden.

Für die Iteratoren `loop-over-role-successors` bzw. `loop-over-role-predecessors` werden zudem (auf Anforderung) auch Kanten materialisiert (s. obige Diskussion).

2. Betrachten wir nur den Fall virtueller RCC-Assertionen in einem MiDeLoRA-Implicite-Relationen-Karten-Substrat (MiDeLoRA-Implicit-Relations-Map-Substrate) der Art **E3** (die Knoten sind also explizit, lediglich die Kante ist virtuell). Nun werden `get-role-successors`, `get-role-predecessors` bzw. die Iteratoren `loop-over-role-successors` und `loop-over-role-predecessors` die Räumlichkeit der ABox nutzen, um die entspr. Knoten mittels Inspektionsmethoden zu ermitteln. Es werden dann die entspr. Kanten materialisiert. Dies ermöglicht also die in Kap. 6.1.5, S. 405 diskutierte *Anfragefokussierung*, da lediglich die Rollen-Assertionen materialisiert werden, die im Laufe des Beweisverfahrens angefordert bzw. benötigt werden.

In beiden Fällen kann der Beweiser dann mit den vom Iterator gelieferten materialisierten Objekten so rechnen, als ob diese stets im Hauptspeicher gelegen hätten.

Wir haben aus rein technischer Perspektive geschildert, wie virtuelle Assertionen über entspr. Schnittstellen berücksichtigt werden können. Leider ist das Thema „virtuelle Assertionen“ hiermit noch nicht erschöpfend abgehandelt. Insbesondere besteht die Gefahr, dass virtuelle Assertionen im Beweisprozess schlichtweg „übersehen“ oder „vergessen“ werden, wodurch die Verfahren unvollständig würden. Wir illustrieren und erläutern nun diese Problematik.

6.2.3.2. Illustration von Beweisprozessen mit virtuellen Assertionen

Virtuelle Assertionen sollen also während der Tableau-Expansion dynamisch und somit inkrementell angefordert und u.U. materialisiert werden. Die Basis-Idee ist, dass so ausschließlich die für den Beweisprozess relevanten Assertionen angefordert werden, und dass zudem eine Anfragefokussierung realisiert wird. Leider ist die *Bestimmung der für einen Beweisprozess benötigten Assertionen* ein nichttriviales Problem. Zur Illustration betrachten wir zunächst die ABox

$$\mathcal{A} = \mathcal{A}_M \cup \mathcal{A}_V \text{ mit } \mathcal{A}_M = \{j : C\} \text{ und } \mathcal{A}_V = \{(j, k) : R, k : D\}.$$

Hier seien die Assertionen in $\mathcal{A}_M$ materialisiert, und die Assertionen in $\mathcal{A}_V$ virtuell (also z.B. in der Datenbank).

Wir betrachten einen Instanzenretrieval-Beweiser, also eine Implementierung von `concept_instances`. Ein einfacher linearer Beweiser wird die Menge der C -Instanzen durch Reduktion auf eine Reihe von ABox-Unerfüllbarkeitstests berechnen, einen pro ABox-Individuum i :

$$\text{concept\_instances}(C, \mathcal{A}) = \{ i \mid i \in \text{inds}(\mathcal{A}), \text{abox\_consistent?}(\mathcal{A} \cup \{i : \neg C\}, \mathcal{T}_{\mathcal{A}}) = \text{FALSE} \}.$$

Dementsprechend muss jedes (auch virtuelles) Individuum i betrachtet, der Knoten für i materialisiert, die Assertion $i : \neg C$ hinzugefügt, und der ABox-Erfüllbarkeitsbeweiser gestartet werden.

Liegt ein Index in Form einer QBox vor (s. Def. 90, S. 227), so kann die Information aus der QBox verwendet werden, sodass lediglich die Individuen aus

$$\text{cand}(C, \mathcal{A}) =_{\text{def}} \bigcap_{p \in \text{parents}(C, \mathcal{A})} p^{\mathcal{E}} \setminus \bigcup_{c \in \text{children}(C, \mathcal{A})} c^{\mathcal{E}}$$

betrachtet werden müssen; dementsprechend können wir die Anzahl der erforderlichen ABox-Erfüllbarkeitstests u.U. drastisch einschränken (s. [Buchheit u. a. 1994a; 1993b, Haarslev u. Möller 2002]):

$$\text{concept\_instances}(C, \mathcal{A}) = \bigcup_{c \in \text{children}(C, \mathcal{A})} c^{\mathcal{E}} \cup \{ i \mid i \in \text{cand}(C, \mathcal{A}), \text{abox\_consistent?}(\mathcal{A} \cup \{i : \neg C\}, \mathcal{T}_{\mathcal{A}}) = \text{FALSE} \}.$$

Durch Markierung der Individuen in $\text{cand}(C, \mathcal{A})$ oder durch explizite Kandidatenspezifikation mittels eines optionalen Argumentes (wie in RACERPRO) kann eine Fokussierung des `concept_instances`-Beweisers ermöglicht werden.

Welche Knoten bzw. Assertionen müssen nun jedoch *für einen einzelnen anfallenden ABox-Erfüllbarkeitstest* $\mathcal{A} \cup \{i : \neg C\}$ innerhalb von `concept_instances` betrachtet werden? Wir hatten erwähnt, dass sowohl Assertionen in $\mathcal{A}_M$ als auch in $\mathcal{A}_V$ vom Beweiser ignoriert werden können (s. Kap. 6.1.5, S. 405). Die Idee, dass nun lediglich die den Knoten i betreffenden Konzept-Assertionen betrachtet werden müssen, ist sicherlich falsch.

So liefert mit dieser Strategie zwar `concept_instances(C)` korrekterweise $\{j\}$, doch schon die Anfrage `concept_instances( $\exists R.D, \mathcal{A}$ )` liefert statt $\{j\}$ nur $\{\}$, da die Assertionen $\{(j, k) : R, k : D\}$ nicht berücksichtigt wurden, und $\{j : C, j : \forall R. \neg D\}$ erfüllbar ist. Daher wäre `concept_instances` unvollständig.

Die Idee, die während des Beweises *benötigten Rollennachfolger und -vorgänger* dynamisch von der ABox anzufordern, bringt uns einen Schritt weiter. Mit dieser Strategie verläuft der $\mathcal{A} \cup \{j : \forall R. \neg D\}$ -Erfüllbarkeitstest nun wie folgt:

1. $\{j : C, j : \forall R. \neg D\}$ führt dazu, dass die $\forall$ -Regel ihre Anwendbarkeit prüft. Hierzu muss die $\forall$ -Regel alle R -Nachfolger mittels `loop-over-role-successors` betrachten bzw. von der ABox anfordern. Dies führt dann zur Materialisierung von $\{(j, k) : R, k : D\}$.
2. Die $\forall$ -Regel propagiert ihre $\neg D$ -Qualifikation nun über die explizit vorhandene $(j, k) : R$ -Assertion bzw. Kante in das Label von k ; die ABox $\{j : C, j : \forall R. \neg D, k : D, k : \neg D, (j, k) : R\}$ wird sofort als unerfüllbar erkannt. Auf das Materialisieren von $(j, k) : R$ kann zudem verzichtet werden, sofern die $\forall$ -Regel $k : \neg D$ sicherstellt.

Dieses Verfahren realisiert bereits eine einfache Art von dynamischer Anfragefokussierung. Enthielte $\mathcal{A}_V$ nun noch die Assertionen $\{(j, k_1) : S_1, (j, k_2) : S_2, k_2 : E, \dots\}$ so würden diese nicht materialisiert werden und daher den Tableau-Beweiser nicht „belasten“.

Leider ist das Verfahren so noch nicht vollständig. Dies kann anhand der ABox

$$\mathcal{A} = \mathcal{A}_M \cup \mathcal{A}_V \text{ mit } \mathcal{A}_M = \{j : \top\} \text{ und } \mathcal{A}_V = \{(k, j) : R, k : (\forall R.C) \sqcup (\forall R.D)\}$$

und der Anfrage `concept_instances(C  $\sqcup$  D,  $\mathcal{A}$ )` gesehen werden. Es ist leicht zu sehen, dass `concept_instances(C  $\sqcup$  D,  $\mathcal{A}$ ) = {j}` ist, denn die ABox

$$\{j : \neg C \sqcap \neg D, (k, j) : R, k : (\forall R.C) \sqcup (\forall R.D)\}$$

ist unerfüllbar, da in einer Vervollständigung entweder

$$\{j : \neg C, (k, j) : R, k : \forall R.C, j : C, \dots\}$$

oder aber

$$\{j : \neg D, (k, j) : R, k : \forall R.D, j : D, \dots\}$$

gelten muss. Beide Vervollständigungen sind unerfüllbar, und damit die Original-ABox.

Doch das „inkrementelle Anfordern“ virtueller Assertionen funktioniert hier offenbar nicht: Starten wir wie eben mit der ABox

$$\{j : \neg C \sqcap \neg D\},$$

so führt dies lediglich zur Expansion

$$\{j : \neg C \sqcap \neg D, j : \neg C, j : \neg D\}$$

und der Prozess würde terminieren. Da hier keine universellen Rollenrestriktions-Konzept-Assertionen (Assertionen der Art $k : \forall R.C$) vorhanden sind, werden die benötigten virtuellen Assertionen gar nicht angefordert. Dementsprechend wird kein Widerspruch abgeleitet, und wieder wäre `concept_instances` unvollständig.

Ein sicherer Weg zur Selektion von Assertionen wäre, *alle von j aus über Rollen-Assertionen erreichbaren Assertionen* bzw. die entspr. Knoten und Kanten zu markieren und somit anzufordern. Hierbei würden sowohl virtuelle als auch normale Assertionen angefordert, und die durch die Rollen-Assertionen definierten Zusammenhangskomponenten („Inseln“) der ABox würden isoliert auf

Erfüllbarkeit getestet werden.

Für die initiale ABox $\{j : \neg C \sqcap \neg D\}$ würde also zunächst die Zusammenhangskomponente für j ermittelt werden. Aufgrund der $(k, j) : R$ -Assertion ist $k : (\forall R.C) \sqcup (\forall R.D)$ in der Komponente enthalten, und der Widerspruch würde entdeckt. Die offensichtliche Erweiterung der Idee, lediglich die R -Rollen-Assertionen zur Minimierung dieser Zusammenhangskomponenten zu betrachten, funktioniert lediglich für einfache Logiken ohne Rollenhierarchien etc. (z.B. $\mathcal{ALC}$).

Für ABoxen ohne bzw. mit leerer assoziierter TBox können wir aber auch ein rein dynamisches Kriterium finden. Betrachten wir die unterschiedlichen ABox-Konzept-Assertionen, so ist letztlich für diejenigen Assertionen, die in einem Beweis *niemals als expandiert gekennzeichnet werden können* sicherzustellen, dass sie stets in jedem (die entspr. Zusammenhangskomponente betreffenden) ABox-Erfüllbarkeitstest berücksichtigt werden.

Für $\mathcal{ALC}$ sind diese Assertionen ausschließlich durch die universellen Rollenrestriktions-Assertionen gegeben; für ausdrucksstärkere DLs müssen noch weitere Assertionen berücksichtigt werden (z.B. nach oben beschränkende Anzahlrestriktionen, $i : (\leq_n R)$). Diese Assertionen müssen identifiziert und dann zusätzlich zu den Konzept-Assertionen für i quasi als „Keimzellen“ für ein Art „Spreading Activation“-Prozess hinzugefügt werden. Diese Keimzellen veranlassen dann die Anforderung und Materialisierung der erforderlichen Subgraphen.

Für die Beispiel-ABox würde dies bedeuten, dass initial die Assertionen

$$\{j : \neg C \sqcap \neg D, k : (\forall R.C) \sqcup (\forall R.D)\}$$

vorhanden sind. Aufgrund von $k : (\forall R.C) \sqcup (\forall R.D)$ wird dann $(k, j) : R$ angefordert, und der Widerspruch wird abgeleitet. Im Allgemeinen muss für eine solche $\mathcal{ALC}$ -ABox also die virtuelle Assertion $i : C$ genau dann berücksichtigt werden, wenn in $\text{subconcepts}(C)$ eine universelle Rollenrestriktion enthalten ist.

Leider ist dieses rein syntaktische Kriterium lediglich für ABoxen ohne TBox oder mit einfacher (also auffaltbarer) TBox anwendbar, nicht jedoch für allgemeine TBoxen. Eine ABox mit einfacher TBox kann durch Auffalten der in den Konzept-Assertionen verwendeten Konzepte in eine erfüllbarkeitsäquivalente ABox mit leerer assoziierter TBox überführt werden, sodass das Kriterium wieder anwendbar ist (wenn auch die Größe der so expandierten ABox exponentiell zunimmt).

Wird eine allgemeine TBox verwendet, so ist insbesondere durch das *Faule Auffalten* (s.u.) nun nicht mehr offensichtlich, welche ABox-Knoten nun Träger von universellen Rollenrestriktions-Assertionen sind, da diese Information in der TBox „versteckt sein kann“. Offensichtlich gibt es i.d.R. mehr als eine Vervollständigung einer ABox. Unterschiedliche Vervollständigungen re-

präsentieren unterschiedliche Modelle. Im Rahmen eines Erfüllbarkeitstests wird ein Tableau-Beweiser jedoch immer nur *eine* Vervollständigung berechnen. Die Idee, anhand dieser Vervollständigung die *gültigen (also folgerbaren) Assertionen* für diese ABox bzw. WB zu identifizieren, ist prinzipiell richtig. Leider steht diese Information in der Vervollständigung anhand der Abhängigkeitsinformationen der einzelnen Tableau-Assertionen nur *unvollständig* zur Verfügung. Anhand der Abhängigkeitsinformationen können *deterministische* und *nichtdeterministische* Assertionen identifiziert werden. Eine deterministische Assertion ist stets auch *gültig*; doch leider gilt die Umkehrung nicht. Wir werden deterministische Assertionen daher für andere Zwecke nutzen (s.u.), nämlich zur Identifikation *offensichtlicher Konzeptinstanzen*. Wir kommen hierauf in Kap. 6.2.8 zurück. Somit ist es also nicht ausreichend, die deterministischen universellen Rollenrestriktionen (und nach oben beschränkenden Anzahlrestriktionen) in einer einmal erzeugten Vervollständigung zu identifizieren und für *zukünftige* Partitionierungszwecke zu nutzen.

Abschließend ist zu sagen, dass MiDeLoRA zwar die konzeptuellen und technischen Voraussetzungen zur Nutzung virtueller Assertionen schafft, doch das noch nicht alle hiermit verbundenen theoretischen und praktischen Probleme gelöst sind. Hier ist noch Forschungsbedarf zu verzeichnen. Für Spezialfälle kann MiDeLoRA jedoch stets entsprechend konfiguriert werden, was ein gewichtiges Argument ist. Im Allgemeinen werden von MiDeLoRA daher Techniken eingesetzt, die sicherstellen, dass eher zu viel als zu wenig virtuelle Assertionen materialisiert werden, wodurch die Vollständigkeit gewährleistet wird. So eine Technik ist z.B., lediglich eine Partitionierung der ABox auf der Ebene der Zusammenhangskomponenten vorzunehmen (somit werden alle von i aus erreichbaren virtuellen Assertionen berücksichtigt). Wir halten auch fest, dass die diskutierten Probleme für ABoxen, die lediglich Assertionen vom Typ **V1**, **V2**, **E1**, **E2**, **E3** enthalten, nicht auftreten können.

6.2.3.3. Durch die Befehlshistorie eröffnete Potentiale

Die explizite Repräsentation des Beweises bzw. Beweisprozesses in Form der Befehlshistorie birgt zudem einige weitere interessante Potentiale, die nach Meinung des Autors relevant für DL-Systeme der nächsten Generation sind. Die MiDeLoRA-Architektur ermöglicht die prinzipielle Evaluierung dieser Punkte:

1. Die Befehlshistorie kann sowohl für Debugging- als auch für Erklärungsfunktionalität genutzt werden.
2. Durch die explizite Repräsentation der Abhängigkeiten in den Befehlsobjekten kann *dependenzgesteuertes Rücksetzen* (*Dependency Directed Backtracking*, *DDB*) ermöglicht werden. Heutzutage verwenden DL-Systeme

i.d.R. eine schwächere Form des DDBs, das sogenannte *dependenzgesteuerte **Zurückspringen*** (*Dependency Directed Backjumping, DDJ*). Wir erläutern das DDJ später noch ausführlicher. Hier seien die folgenden Erläuterungen zunächst ausreichend.

Durch die Anwendung der Tableau-Regeln im Rahmen der Tableau-Expansion wird ein Suchbaum aufgespannt, dessen Knoten Zustände im Suchraum repräsentieren. Die Knoten in diesem Suchbaum entsprechen Tableau- bzw. ABox-Zuständen. Knoten mit mehr als einem Kind repräsentieren nichtdeterministische Regelanwendungen und werden Verzweigungspunkte oder Oder-Knoten genannt. Verzweigungspunkte werden durch die bereits diskutierten Verzweigungspunkt-IDs eindeutig identifiziert.

Ein solcher Verzweigungspunkt ist z.B. durch die ABox $\{i : C \sqcup D, i : \neg C\}$ gegeben, auf die die $\sqcup$ -Regel angewendet wird. Die ID des Verzweigungspunktes sei 1. Die $\sqcup$ -Regel erzeugt im Suchbaum nun zwei Kind-Knoten bzw. -Bäume: $\{i : C \sqcup D, i : \neg C, i : C\}$ (Alternative 1) und $\{i : C \sqcup D, i : \neg C, i : D\}$ (Alternative 2). Sowohl für die Assertion $i : C$ als auch für $i : D$ wird nun vermerkt, dass sie vom Verzweigungspunkt mit der ID 1 abhängen.

Tritt während der Tableau-Expansion nun ein Widerspruch auf, so wird vom Suchverfahren ein *Zurücksetzen bzw. Zurückspringen* (*Backtracking*) im Suchbaum ausgelöst. Offensichtlich führt die Wahl von Alternative 1 zu einem Widerspruch. Anhand der Abhängenzen bzw. der vermerkten Verzweigungspunkte-IDs der Assertionen $i : \neg C$ und $i : C$ kann nun erkannt werden, dass zum Verzweigungspunkt mit der ID 1 zurückgesetzt und daher mit Alternative 2 fortgesetzt werden sollte.

Beim *chronologischen Rücksetzen* wird hingegen jeweils der *chronologisch älteste Verzweigungspunkt* (mit noch offenen Alternativen) gewählt. Dies ist das klassische Tiefensuch-Verfahren (einfache Backtracking-Suche). Hierfür wird keine Abhängenzinformation benötigt. Nachteilig hieran ist, dass der so identifizierte Knoten evtl. gar nicht am Vorliegen des Widerspruchs mitbeteiligt ist (d.h., durch Wahl seiner nächsten Alternative wird der selbe Widerspruch u.U. erneut auftreten). Durch *Abhängenzanalyse der an einem Widerspruch beteiligten Assertionen* wird daher beim DDJ oder DDB ein Verzweigungspunkt identifiziert, der *am Widerspruch beteiligt* ist, sodass Grund zu der Annahme besteht, dass durch Wahl seiner nächsten Alternative der Widerspruch beseitigt wird (andernfalls wird weiter zurückgesetzt).

Sowohl beim einfachen chronologischen Rücksetzen als auch beim DDJ wird der durch den Verzweigungspunkt repräsentierte Tableau-Zustand wiederhergestellt. Hierzu muss in der MIDELORA-Architektur die *rollback-to*-Methode angewendet werden. Hierbei wird jedoch u.U. auch

„Expansionsarbeit“ rückgängig gemacht, die *nicht* am Vorliegen des Widerspruchs beteiligt ist, und die eventuell bei der Wahl der nächsten Alternative dann erneut geleistet werden muss (es ist bekannt, dass *Caches* beim DDJ eine wesentliche Rolle zur Einsparung ansonsten zu wiederholender Expansionsarbeit leisten).

Es wäre daher vorteilhaft, wenn ein *selektives Zurückrollen des Tableau-Zustands* möglich wäre. Dabei sollen im Rahmen des Zurückrollens nur die Assertionen aus dem Tableau entfernt werden, die direkt oder indirekt von Assertionen abhängen, die somit direkt oder indirekt für den Widerspruch mitverantwortlich sind. In gewisser Weise werden dann selektiv „die schlechten Teile aus dem Tableau herausgeschnitten“. Man spricht dann vom *dependenzgesteuerten Zurücksetzen (Dependency Directed Backtracking, DDB)*. Um diese Assertionen aus dem Tableau zu entfernen, könnte nun prinzipiell *jede* Assertion dediziert daraufhin untersucht werden, ob sie von einem entspr. Verzweigungspunkt abhängt. Dies ist jedoch aufwendig (da die Anzahl der Assertionen i.d.R. groß ist).

Die explizite Befehlshistorie in MIDeLoRA eröffnet jedoch eine direktere Möglichkeit zur Ermittlung der Assertionen, die sicherlich nicht am Widerspruch mitbeteiligt sind. Diese können wie folgt ermittelt werden:

Im Befehlsobjekt kann durch das Attribut bzw. die Assoziation *precondition-for-actions optional* (sofern DDB gewünscht wird) vermerkt werden, welche anderen Befehlsobjekte für die ausgeführte Aktion ursächlich sind. Es existiert somit eine zweite Verkettungsstruktur zwischen Befehlsobjekten, diesmal nicht chronologisch (wie in der Befehlshistorie), sondern kausal bestimmt. Eine inverse Assoziation *postcondition-of-actions* ist ebenfalls vorgesehen, sodass wieder effizientes Navigieren in beide Richtungen ermöglicht wird.

Im Falle eines Widerspruchs werden zunächst alle mitverantwortlichen Verzweigungspunkte identifiziert. Für jeden identifizierten Verzweigungspunkt wird sein Befehlsobjekt lokalisiert, im Folgenden Start-Befehlsobjekt genannt. Durch Traversierung der *precondition-for-actions*-Assoziationen werden nun die „Enden“ der *Befehlswirkungs-Kausalkette* für jedes Start-Befehlsobjekt identifiziert. Alle so erhaltenen Befehlsobjekte werden markiert. In einer zweiten Phase werden in einer Schleife die Befehlsobjekte wieder demarkiert, die einen nichtmarkierten *precondition-for-actions*-Nachfolger haben. Die Schleife endet, wenn keine Befehlsobjekte mehr demarkiert werden können. Nun werden alle markierten Befehlsobjekte in *chronolo-*

gischer Reihenfolge rückgängig gemacht. Die Befehlsobjekte werden ebenfalls aus der Befehlshistorie entfernt (und auch die obigen kausalen Assoziationen werden entspr. geändert).

Hierbei handelt es sich also *nicht* um ein chronologisches Zurückrollen wie im obigen Fall, da bestimmte Teile der Befehlshistorie und somit auch die Auswirkungen der durch sie dokumentierten Tableau-Änderungen erhalten bleiben. Dies sind die Aktionen, die zwar chronologisch nach der für den Widerspruch mitverantwortlichen Start-Aktion ausgeführt wurden, von dieser jedoch nicht kausal abhängen, und die somit nicht am Widerspruch beteiligt sind. Die durch sie herbeigeführten Tableau-Änderungen bleiben daher nach dem Rücksetzen erhalten und werden nicht automatisch „abgeräumt“, was bei Verwendung von rein chronologischem Rücksetzen oder DDJ geschehen würde.

3. Aufgrund der inhärenten Berechnungskomplexität von DLs können Beweise u.U. sehr lange dauern. Im Falle eines Hauptspeicherverlustes (durch Stromausfall o.ä.) wäre das Tableau verloren. Der Beweis müsste nach Neustart erneut beginnen.

Eine Instanz der Klasse DB-ABox kann prinzipiell zu jedem Zeitpunkt persistent gemacht werden; z.B. nach jeder Regelanwendung. Dies erscheint aus Performanzgründen nicht sinnvoll. Intelligenter wäre es, den Zustand des Tableaus nur alle n Operationen persistent zu machen, und dann lediglich die seit dem letzten „Abzug“ erfolgten Operationen, also die Befehlshistorie seit diesem Zustand, persistent zu speichern. Es ist klar, dass die persistente Befehlshistorie das Potential hat, den Zustand des DL-Systems nach Hauptspeicherverlust prinzipiell wieder zu rekonstruieren (ähnlich einem Datenbank-Log). Dieses Vorgehen ist bei hinreichend großem n praktikabel, da Befehlsobjekte leichtgewichtig sind. Relationale Datenbanken verwenden zudem sog. *Logpuffer-Architekturen*, die eine hohe Effizienz gewährleisten, vgl. z.B. [Kemper u. Eickler 2004, Kap. 10].

Da diese Technik auch bzgl. einfacher ABox-Updates (Hinzufügen und Entfernen von Assertionen) anwendbar ist, ermöglicht die MIDELORA-Architektur auch erste Untersuchungen und Experimente bzgl. *transaktionaler DL-Systeme*. Hier sind die ACID-Eigenschaften zu nennen; *Consistency und Isolation (CI)* sind durch grobgranulare Sperrprotokolle auch für DL-Systeme relativ einfach zu realisieren, während *Durability und Atomicity (DA)* durch entspr. Erholungs- und Logpuffer-Architekturen sowie persistente Speicher (Datenbanken) sichergestellt wird.

Betrachtet man den Stand der Kunst dieser Techniken (wie er in relationalen Datenbanksystemen realisiert ist), so wird deutlich, dass DL-Systeme diesbez. noch einen recht langen Weg zu gehen haben. Es ist die Meinung des Autors, dass Eigenschaften wie Transaktionalität für DL-Systeme nur

durch explizite Tableau-Repräsentation und persistente Befehlshistorien (wie oben angedeutet) realisiert werden können. Der *Berechnungszustand* eines DL-Systems muss daher wesentlich ernster genommen werden, als dies in bisherigen DL-Systemen der Fall ist. Wir sehen die MiDeLoRA-Architektur als einen ersten Schritt in diese Richtung an.

4. Die Befehlshistorie könnte prinzipiell sogar dazu verwendet werden, den aktuellen Berechnungszustand eines MiDeLoRA-Beweisers auf einen anderen Rechner zu *migrieren*, wodurch Verfügbarkeit gewährleistet werden könnte: Im Falle eines Rechner-Fehlers kann die Berechnung auf einem anderen Rechner fortgesetzt werden. Andere Anwendungen für migrierende Beweise (z.B. in Multi-Agenten-Systemen) sind denkbar.

Es wird deutlich, dass die untersuchte Architektur sowohl unübersehbare Potentiale, aber auch Probleme birgt. Natürlich sind diese Potentiale nicht alle im Rahmen dieser Arbeit realisiert und untersucht worden, wie bereits hier und da angedeutet. Es wird sich z.B. zeigen müssen, ob die konsequente Verwendung der OOP-Techniken nicht zu Performanzproblemen und Speicherproblemen führt (s. Kap. 7). Wir werden bei der Implementierung jedoch berücksichtigen, dass der durch *dynamisches bzw. spätes Binden* (im Vergleich zu einem normalen Funktionsaufruf) verursachte zusätzliche Laufzeitaufwand möglichst schon zur Übersetzungszeit aufgelöst wird, ähnlich wie wir dies im Falle des Anfrageübersetzers realisiert hatten (s. defquery-code-Beschreibung, Kap. 5.2.2.3.1, S. 296).

6.2.3.4. Vergleich mit einem rein funktionalen $\mathcal{ALC}$ -Tableau-Beweiser

Wir vergleichen nun die für MiDeLoRA getroffenen Entwurfsentscheidungen und die sich hieraus ergebenden Konsequenzen mit einem funktionalen $\mathcal{ALC}$ -Beweiser, der prototypisch für „einfache“ bzw. konventionellere Tableau-Beweiser steht. Die Präsentation und Analyse des einfachen Beweisers wird zudem die Verständlichkeit der nachfolgenden Ausführungen erhöhen bzw. vorbereiten.

Einige Tableau-Beweiser (z.B. die ersten Versionen von RACER) arbeiten rein funktional. Die rein funktionale Programmierung kennt keine Zustände. Eine rein funktionale Implementation wird daher Datenstrukturen nicht destruktiv verändern, und lediglich *Werte* verwenden. Solche Datenstrukturen werden auch „unboxed data structures“ genannt; im Gegensatz hierzu verwendet MiDeLoRA fast ausschließlich „boxed data structures“.

Ein funktionaler Tableau-Beweiser wird zur Repräsentation des Tableaus z.B. eine einfach verkettete Liste verwenden. Die nichtdestruktive Verwendung der Datenstrukturen durch den Beweiser macht insbesondere das Wiederherstel-

len eines älteren Zustands der Datenstrukturen bzw. des Tableaus im Rahmen des Zurücksetzens einfach. Idealerweise muss hier lediglich ein Zeiger auf den Kopf der Datenstruktur (die Liste) umgesetzt werden, was automatisch geschehen kann, wenn die Knoten im Suchbaum durch Funktions-Inkarnationen gegeben sind (dann reicht ein einfaches return aus der Inkarnation).

Die Implementierung eines einfachen funktionalen $\mathcal{ALC}$ -Konzepterfüllbarkeitsbeweisers (Implementierung von `concept_satisfiable?` bzgl. leerer TBox in $\mathcal{ALC}$) sieht in COMMON LISP dann wie folgt aus:

```
(defun alc-sat (concept)
  (labels
    ((alc-sat1 (expanded unexpanded)
      (labels
        ((get-negated-concept (concept) (nnf '(not ,concept)))
         (select-concept (type)
           (find-if
            #'(lambda (concept)
                  (and (consp concept) (eq (first concept) type)))
            unexpanded))
         (select-atom nil
           (find-if
            #'(lambda (concept)
                  (or (symbolp concept)
                      (and (consp concept) (eq (first concept) 'not)
                          (symbolp (second concept))))))
            unexpanded))
         (clash (concept)
           (let ((negated-concept (get-negated-concept concept)))
             (find negated-concept expanded :test #'equal)))
         (register-as-expanded (concept)
           (unless (clash concept)
             (alc-sat1 (cons concept expanded)
                       (remove concept unexpanded :test #'equal))))))
      (let ((atom (select-atom)))
        (if atom
          (register-as-expanded atom)
          (let ((and-concept (select-concept 'and)))
            (if and-concept
              (progn (dolist (conjunct (rest and-concept))
                      (when (clash conjunct) (return-from alc-sat1 nil))
                      (push conjunct unexpanded))
                     (register-as-expanded and-concept))
              (let ((or-concept (select-concept 'or)))
                (if or-concept
                  (let ((unexpanded-old unexpanded))
```

```

      (some #'(lambda (arg)
        (unless (clash arg)
          (setf unexpanded (cons arg unexpanded-old))
          (register-as-expanded or-concept)))
        (rest or-concept)))
    (let ((some-concept (select-concept 'some)))
      (if some-concept
        (let* ((qualification (third some-concept))
              (role (second some-concept))
              (initial-label
               (cons qualification
                 (mapcar #'third
                   (remove-if-not
                     #'(lambda
                       (concept)
                       (and
                        (consp concept)
                        (eq (first concept)
                          'all)
                        (eq (second concept)
                          role)))
                     unexpanded)))))
          (and (alc-sat1 nil initial-label)
               (register-as-expanded some-concept)))
        t)))))))))

(alc-sat1 nil (list (nnf concept))))

```

6.2.3.4.1. Charakterisierung des funktionalen *ALC*-Beweisers Der einfache *ALC*-Beweiser lässt sich wie folgt charakterisieren:

1. Der Beweiser kann ausschließlich Erfüllbarkeit von *ALC*-Konzepten in NNF entscheiden; er ist nicht als ABox-Beweiser nutzbar.
2. Der Beweiser verwendet *einfaches chronologisches Rücksetzen*.
3. Es werden ausschließlich *leichtgewichtige Repräsentationen* verwendet (atomare Konzepte werden als Symbole und komplexe Konzepte als Listen dargestellt; zur Tableau-Repräsentation werden ebenfalls Listen verwendet, s. lokale Variablen *expanded* und *unexpanded*).
4. Der Beweiser verwendet *keine explizite Tableau-Repräsentation*. Weder Tableau-Knoten noch -Kanten sind explizit als Objekte bzw. Datenstrukturen vorhanden. Lediglich die Assertionen des aktuell untersuchten Knotens sind mittels *unexpanded* und *expanded* zugreifbar. Stattdessen ist die Tableau-Struktur zu jedem Zeitpunkt des Beweises *implizit durch die*

Struktur des Aufrufkellers (Stacks) inkl. der jeweiligen Bindungen der lokalen Variablen, u.a. unexpanded und expanded, gegeben. Die Tableau-Repräsentation ist also mit dem Suchbaum und daher mit der Kontrollstruktur gekoppelt. Ein Tableau-Zustand entspricht einem Knoten im Suchbaum und daher auch einem *Bindungsrahmen (Stack Frame)* des Kellers. Die Bindungsrahmen werden durch Funktionsaufrufe angelegt und entsprechen daher Funktions-Inkarnationen. Während Konzept-Assertionen noch in gewisser Weise als explizit repräsentiert angesehen werden können, so ist eine Rollen-Assertion hingegen lediglich implizit durch zwei direkt benachbarte Bindungsrahmen auf dem Keller und die Bindung der *role*-Variable im älteren der beiden Rahmen repräsentiert.

5. Die vom Beweiser *verwendete Regelanwendungsstrategie* expandiert zunächst alle einen Tableau-Knoten betreffenden aussagenlogischen Konzept-Assertionen vollständig, wendet also die $\sqcap$ - und $\sqcup$ -*ALC*-Regeln erschöpfend an. Erst wenn diese nicht mehr anwendbar sind, werden die $\exists$ - und $\forall$ -Regeln angewendet. Für einen neu erzeugten Tableau-Knoten werden dann zunächst wieder die $\sqcap$ - und $\sqcup$ -Regel erschöpfend angewendet, und so weiter, bis alle Assertionen berücksichtigt wurden.

Die Expansionsstrategie ist sehr klar in der Programmstruktur ersichtlich (aufgrund der geschachtelten bedingten Anweisungen). Die Struktur des Beweisers ist sehr klar und konzise und kann beinahe als mathematische Funktionsbeschreibung bzw. ausführbare Spezifikation angesehen werden. Es wäre schön, wenn derartig klare Programmstrukturen auch für hochgradig optimierte Beweiser erreichbar wären (s.u.), denn dann ist eine hohe Verständlichkeit und Wartungsfreundlichkeit gegeben.

6. Der Suchbaum (und somit das implizit durch ihn repräsentierte Tableau) *hat zu jedem Zeitpunkt eine Baumstruktur*. Es ist nicht verwunderlich, dass nur solche Tableaus betrachtet werden müssen, denn *ALC* hat die Baummodell-Eigenschaft (s. Kap. 2.2.2.3, S. 67).
7. Der Beweiser verwendet die sog. *Trace Technique*. Aufgrund der Tiefensuche muss lediglich der aktuelle Zweig des Suchbaums im Speicher gehalten werden. Der Beweiser benötigt daher zu jedem Zeitpunkt höchstens polynomiellen Platz, da die Tiefe des Suchbaums ausschließlich durch die strukturelle Schachtelungstiefe bzw. -Komplexität des Eingabekonzeptes *C* bestimmt ist. Aufgrund der Anwendbarkeit der Trace Technique für *ALC* und weil die Länge der Pfade für *ALC* hier polynomiell beschränkt ist, ist *ALC*-Konzepterfüllbarkeit in PSPACE.⁷
8. Durch die *leichtgewichtige Repräsentation des Tableaus* und die nichtde-

⁷Es ist eben *nicht* notwendig, ein potentiell exponentiell großes Tableau im Speicher zu halten, was zunächst Mitgliedschaft in EXPSPACE bedeuten würde).

strukturellen Modifikationen ist der *Verwaltungsaufwand für das Tableau minimal*.

Das Wiederherstellen einer früheren Version bzw. eines früheren Zustands des Tableaus im Rahmen des Rücksetzens wird unmittelbar durch eine einfache Rückkehr aus einem rekursiven Aufruf bzw. einer Inkarnation von `alc-sat1` durch die Programmiersprache bereitgestellt.

Da in der Inkarnation von `alc-sat1`, aus der zurückgekehrt wird, die durch `unexpanded`, `expanded`, ... referenzierten Listen lediglich nicht-destruktiv verändert wurden (neue Elemente wurden lediglich zum Anfang der Listen hinzugefügt, wofür es ausreicht, die Referenzen auf die Listenköpfe zu verändern), sind nach Rückkehr aus der Inkarnation wieder die alten Referenzen auf die originalen Listenköpfe mittels `unexpanded`, `expanded` verfügbar, da der alte Bindungsrahmen wieder aktiv ist. Der Speicherverwalter (Garbage Collector) gibt den von den nun nicht mehr referenzierten Listenelementen belegten Speicher automatisch wieder frei.

9. *ALC* hat die Eigenschaft, dass die Erfüllbarkeit einer $i : \exists R.C$ -Assertion ausschließlich von den für i geltenden für R anwendbaren universellen Rollenrestriktionskonzepten, also Assertionen der Art $i : \forall R.D_k$, $k \in 1 \dots n$, sowie von C selbst abhängt. Dies wird die *Sub-Tableau-Eigenschaft* genannt [Horrocks 1997a].

Zur Prüfung der Erfüllbarkeit von $i : \exists R.C$ kann daher ein neuer isolierter Knoten j mit $j : C \sqcap D_1 \sqcap \dots \sqcap D_n$ angelegt werden. Dies entspricht einem sog. Sub-Tableau $\{j : C \sqcap D_1 \sqcap \dots \sqcap D_n\}$. Andere Knoten als i können das Sub-Tableau nicht beeinflussen. Das Konzept $C \sqcap D_1 \sqcap \dots \sqcap D_n$ wird auch das *initiale Konzept* oder *initiale Label* von j genannt. Das initiale Label muss zudem *stabil* sein, was bedeutet, dass es sich durch nachfolgende Regelanwendungen in anderen Teilen des Tableaus nicht mehr verändert [Horrocks 1997a, Horrocks u. Patel-Schneider 1999].

Die *Stabilität* des initialen Labels ist Voraussetzung für die Anwendung momentan bekannter *Caching-Techniken* – offensichtlich kann ein einmal errechneter Erfüllbarkeitsstatus für ein initiales Label bei Wiederauftreten dieses Labels zur Einsparung der Sub-Tableau-Expansion genutzt werden. In diesem Falle liefert dann (`alc-sat1 nil initial-label`) einfach den im Cache gespeicherten Erfüllbarkeitsstatus für `initial-label` und spart jede weitere Expansion ein (im einfachen Beweiser ist dies jedoch nicht implementiert).

Die Voraussetzung ist speziell für DLs mit inversen Rollen nicht mehr gegeben; hier ist (zum Zeitpunkt des Schreibens) kein einfaches Kriterium bekannt, ab welchem Expansionszustand des Tableaus das initiale Label eines Knotens stabil ist. Die Forschungen hierzu sind noch nicht abge-

schlossen, vgl. auch [Ding u. Haarslev 2005] für erste Ergebnisse diesbez.

6.2.3.4.2. Defizite des funktionalen *ALC*-Beweisers und durch eine explizite Tableau-Repräsentation eröffnete Potentiale Während die bisher diskutierten Charakteristika des *ALC*-Beweisers fast ausschließlich als positiv zu bewerten sind (mit Ausnahme der Listenrepräsentation), so ist es dennoch die Meinung des Autors, dass für *komplexere* DLs als *ALC* und speziell für ABox-Beweiser, die Nachteile einer solchen impliziten und vermischten Repräsentation von Kontroll- und Tableau-Struktur aus folgenden Gründen überwiegen:

1. Der Keller muss letztlich das gesamte Tableau repräsentieren. Selbst wenn die Trace-Technik anwendbar ist (was nicht für alle DLs der Fall ist), so wird der Kellerspeicher dennoch stark belastet. Die Halde (Heap) bzw. „der Hauptspeicher“ ist zur Tableaurepräsentation wesentlich besser geeignet.
2. Ein ABox-Beweiser benötigt in jedem Falle eine explizite Repräsentation der ABox-Assertionen; damit Indizierung gewährleistet ist, sollte es sich nicht um eine einfache Liste von Assertionen handeln. Eine graphartige Repräsentation (wie im Substrat-Datenmodell) ist daher naheliegend.
3. Für DLs mit transitiven Rollen oder allgemeinen TBoxen müssen Blockierungsbedingungen zur Sicherstellung der Terminierung des Tableau-Beweisers effizient ausgewertet werden können, s. Kap. 2.2.7.3, S. 108. Blockierungsbedingungen verhindern bzw. blockieren die Anwendung *generierender Regeln* (u.a. der $\exists$ -Regel).

Die Bedingung muss hierzu verschiedene Teile des Tableaus betrachten. Dazu müssen die Label der entspr. Knoten jedoch wahlfrei bzw. beliebig zugreifbar sein. Bei einer rein kellerbasierten Repräsentation ist dies nicht direkt möglich, da die „weiter unten liegenden Bindungsrahmen“ nicht wahlfrei zugreifbar sind (ohne die Kontrolle zurückzugeben). Ein weiteres listenwertiges Argument für *alc-sat1* wird daher im Beispiel-*ALC*-Beweiser notwendig, welches die Label *aller Knoten auf dem aktuellen Zweig (Trace)* aufführt. Nun sind bereits drei Argumente notwendig: *expanded*, *unexpanded*, *labels-of-nodes-on-path*. Komplexere Blockierungs-Bedingungen erfordern zudem den Vergleich von *Kanten-labeln*. Diese sind im einfachen *ALC*-Beweiser nun gar nicht explizit verfügbar – also würden weitere zusätzliche Argumente notwendig.

Optimierungstechniken und kompliziertere DLs machen ebenfalls weitere Argumente erforderlich; zudem führt die Partitionierung der Argumente nach *Konzepttyp* (wie für MIDELORA Knoten-Label besprochen, *unexpanded-and-concepts*, *unexpanded-or-concepts*, ...) zur Steigerung der Performanz durch Kleinhalten der Listen bzw. Mengen zu einer

weiteren Multiplikation der von `alc-sat1` benötigten Argumente. Die Bindungsrahmen benötigen so immer mehr Platz. Ein guter COMMON LISP-Übersetzer wird zwar die rekursiven Aufrufe wann immer möglich durch Schleifen ersetzen bzw. Endrekursion identifizieren und somit „Bindungsrahmen einsparen“. Doch missfällt dem Autor die Separierung eigentlich zusammengehöriger Informationen in unterschiedliche Argumente und Datenstrukturen auch aus konzeptioneller Perspektive.

4. Liegt eine Beschreibungslogik mit inversen Rollen vor ($\mathcal{I}$ im Namen), so können die Qualifikationen aus $\forall R^{-1}.C$ -Konzepten auch „rückwärts“ entlang existierender R -Rollen-Assertionen bzw. R -Kanten im Tableau wirken.

Ein Rückpropagieren von Qualifikationen ist aufgrund der Kopplung der Tableau-Repräsentation mit dem Keller nur durch Rückgabe der Kontrolle an den weiter unten liegenden Knoten möglich. Die Qualifikation C würde als Argument nach unten (als Rückgabewert von `alc-sat1`) durchgereicht. Durch das Rücksetzen wird automatisch alle bereits geleistete Expansionsarbeit hinfällig. Die Qualifikation C muss nun zum Knoten hinzugefügt und erneut expandiert werden.

Auch in diesem Fall bietet sich also eine Entkopplung von Tableau- und Kontrollstruktur an; zumal erfordert die Erkennung einer Rückpropagierungssituation ein weiteres Argument, da ein Knoten bzw. eine Inkarnation nun wissen muss, ob sie ein R -Nachfolger ist oder nicht (das Argument `role` muss also in den rekursiven Aufruf von `alc-sat1` mitübergeben werden).

5. Ein explizit repräsentiertes Tableau in Form eines Graphen ist auch unabdingbar für Beweiser für DLs wie $\mathcal{ALCT}_{RCC}$ oder $\mathcal{ALCT}_{\mathcal{RA}^\oplus}$, da hier komplexe Rahmen-Bedingungen (*Frame Conditions*), s. Kap. 2.2.6, S. 97 an die relationale Struktur des Tableaus gestellt werden. Die MIDeLoRA-Architektur ist daher auch gut geeignet zur Implementation von Beweisern für Modallogiken.
6. Komplexe Rahmen-Bedingungen werden auch durch die Horn-Regeln der SEMANTIC WEB-Regelsprache (*SEMANTIC WEB Rule Language*, SWRL) formulierbar [Horrocks u. a. 2004a].

Wir postulieren, dass MIDeLoRA eine geeignete Basis für SWRL darstellt. Evidenz für diese These ist durch die effizienten Substrat-Anfragesprachen sowie die (konzeptionell mit SWRL verwandten) $\mathcal{ALCT}_{RCC}$ -Implementationen in MIDeLoRA gegeben.

7. Anwendungsprobleme, die durch *Modellkonstruktion* formalisiert werden können, sind nicht an der Lösung eines Entscheidungsproblems als Ausgabe interessiert, sondern an einer durch das Verfahren *konstruierten Ver-*

vollständigung.

Beschreibungslogisches Konfigurieren kann als Modellkonstruktionsprozess verstanden werden [Buchheit u. a. 1994b, McGuinness 2003]. Das Herstellen einer validen, der Baubeschreibung entsprechenden Konfiguration kann als Konstruieren eines Modells der Baubeschreibung formalisiert werden. Ein Modell ist hier durch eine Vervollständigung repräsentiert.

Prinzipiell werden von Tableau-Beweisern jedoch nur Vervollständigungen bzw. Modelle einer bestimmten Art betrachtet, z.B. Baum-Modelle. Zudem ist zu beachten, dass eine Vervollständigung i.d.R. nur einen *endlichen Bauplan* für ein evtl. unendliches Modell repräsentiert. Für einfache Logiken gibt es jedoch eine einfache 1-zu-1-Korrespondenz zwischen Vervollständigung und Modell, z.B. für *ALC* (s. auch Bemerkungen in Kap. 2.2.7.2), doch auch für *ALC* werden nicht alle möglichen Modelle betrachtet. Oftmals soll zudem nicht nur *eine* Lösung generiert werden, sondern mehrere; evtl. sollen bestimmte Lösungen sogar präferiert werden. Die MIDELoRA-Architektur ermöglicht aufgrund der expliziten Tableau-Repräsentation die Implementation von anwendungsspezifischen *Modellgeneratoren*. Ähnlich wie in Sprachen wie PROLOG kann nach einer gefundenen Vervollständigung ein erneutes Zurücksetzen (Backtracking) ausgelöst werden, um die Suche nach einer alternativen Vervollständigung anzustoßen.

6.2.4. Benötigte Basis-Funktionalität

Offensichtlich wird für ein DL-System-Baukastensystem umfangreiche weitere Basis-Funktionalität benötigt. Diese Basis-Funktionalität wird in Form von Schablonenmethoden in entspr. Wurzelklassen bereitgestellt.

6.2.4.1. TBox- und ABox-Verwaltungsfunktionalität

MIDELoRA soll mehrere TBoxen und ABoxen verwalten können. Dies impliziert Funktionalität zum Anlegen, Auffinden (anhand des Namens), Löschen, Laden, Speichern und Verändern von ABoxen und TBoxen.

6.2.4.2. Syntaktische Basis-Funktionalität

Ein DL-System benötigt auch Repräsentationen für Konzepte und Rollen. Ein DL-Parser übersetzt Konzepte in KRSS-Syntax in ASGs, die Instanzen entsprechender Klassen sind. Anders als im Falle des einfachen *ALC*-Beweisers wer-

den Konzepten nicht als einfache COMMON LISP-Listen, sondern als Objektgraphen repräsentiert. Diese Konzept-Instanzen bzw. ASGs haben eine Wertsemantik; somit werden zwei syntaktisch gleiche Konzeptterme durch die selbe Konzept-Instanz dargestellt. Die Repräsentation von Konzepten als CLOS-Instanzen ermöglicht wieder die praktische objektlokale und objektindizierte Verwaltung konzeptspezifischer Informationen.

Die Wertsemantik wird beim Parsen bzw. *Enkodieren* der Konzepte sichergestellt. Zwei Konzepte sind genau dann gleich, wenn ihre enkodierten Instanzen identisch sind. Hierzu wird vom Parser ein sog. *Konzeptspeicher* verwendet; dabei handelt es sich um eine Hashtabelle, die (normalisierte) Konzepte in KRSS-Syntax auf ihre enkodierten Konzept-Instanzen abbildet. Jede TBox stellt einen Konzeptspeicher zur Verfügung. Bevor der Parser also für ein Konzept (oder Subkonzept eines komplexeren Konzeptes) eine neue Konzept-Instanz anlegt, wird zunächst im Konzeptspeicher nachgesehen, ob für das entspr. Konzept nicht bereits ein Eintrag bzw. eine Instanz vorhanden ist. Diese wird, falls vorhanden, zurückgegeben; andernfalls erzeugt der Parser eine neue Instanz und registriert sie im Konzeptspeicher.

Um diesen Effekt zu maximieren und um mit möglichst wenigen ASG-Instanzen (durch *Structure Sharing*) auszukommen, werden Konzepte vor dem Eintragen in den Konzeptspeicher zunächst *normalisiert* [Horrocks 2003b; 1997a]. Normalisieren bedeutet hier nicht, dass Konzepte in NNF o.ä. gebracht werden. Stattdessen handelt es sich um eine einfache Form struktureller Inferenz, die dazu dient, die Anzahl der Konzepte im Konzeptspeicher klein zu halten. So werden beim Normalisieren beispielsweise redundante Disjunkte oder Konjunkte entfernt ($C \sqcap C \equiv C$), unnötige Schachtelungen von Disjunktionen und Konjunktionen entfernt ($C \sqcap (D \sqcap E) \equiv C \sqcap D \sqcap E$), etc. So würden also sowohl $C \sqcap (D \sqcap C)$ als auch $D \sqcap C \sqcap D$ durch Normalisierung und Enkodierung auf die selbe Konzept-Instanz $C \sqcap D$ abgebildet werden. Ein weiterer positiver Seiteneffekt ist, dass bereits beim Enkodieren ein Konzept sehr einfach als offensichtlich inkonsistent oder tautologisch erkannt werden kann.

Die ASG-Klassen zur Repräsentation von Konzepten entsprechen den konzeptbildenden Operatoren. Pro Operator sehen wir eine Unterklasse der Klasse *Konzept* (*Concept*) vor (Und-Konzept (*And-Concept*), Oder-Konzept (*Or-Concept*), ...). Wir stellen in MiDeLoRA Konzepte bzw. Operatoren bis zur Sprache $\mathcal{ALCHIfN}_{\mathcal{R}^+}$ bereit.

Die ASG-Klassen zur Repräsentation von Rollen werden so entworfen, dass Rollenhierarchien ($\mathcal{H}$), transitiv abgeschlossene Rollen ($\mathcal{R}^+$), zueinander inverse Rollen ($\mathcal{I}$), sowie Bild- und Urbildrestriktionen deklariert werden können.

Wir unterscheiden zwischen *einfachen Rollen* (Klasse Einfache-Rolle (*Simple-Role*)) und *komplexen Rollen* (Klasse Komplexe-Rolle (*Complex-Role*)). Eine komplexe Rolle ist eine konjunktive oder eine disjunktive Rolle. Disjunktive Rollen wer-

den beispielsweise für ABoxen mit disjunktiven Rollen-Assertionen benötigt. MIDeLoRA unterstützt solche ABoxen, da diese Funktionalität letztlich auch für die $\mathcal{ALCT}_{\mathcal{RCC}}$ -Familie benötigt wird. Zudem können Rollenkompositionsaxiome der Art $S \circ T \sqsubseteq R_1 \sqcup \dots \sqcup R_n$ in einer assoziierten *Relationsbox* (*RBox*), nun: *Rollenbox*, s. Kap. 5.1.2.6.4, S. 257 definiert werden.

6.2.4.2.1. Berechnung der verwendeten DL Die von einem Konzept, einer TBox, einer ABox oder gar einer WB verwendete DL muss zur Selektion eines geeigneten Beweisers im MIDeLoRA-Raum anhand der $\mathcal{DL}$ -Achse identifiziert werden. Hierzu wird *Sprachanalyse-Funktionalität* benötigt. Die Methode `get-language` ist für Konzepte, TBoxen und ABoxen definiert. Die Methode analysiert die im übergebenem Argument verwendeten Operatoren und gibt dann eine sprachbeschreibende Instanz einer entspr. *DL-Sprachklasse* zurück.

Diese Sprachklassen entsprechen Punkten *relevanter* DLs im MIDeLoRA-Raum auf der $\mathcal{DL}$ -Achse. Wir werden unten beschreiben, unter welchen Bedingungen eine DL hier als relevant anzusehen ist und somit Anlass für die Existenz einer Sprachklasse liefert. Für alle durch MIDeLoRA-Beweiser unterstützte DLs werden entsprechende Sprachklassen definiert. Die Klassen können als Singletons angesehen werden. Sprachklassen sind zudem in einer Vererbungshierarchie organisiert, s. Abb. 6.4.⁸

Wie bereits oben bemerkt, belassen wir f im Namen einer DL, die $\mathcal{N}$ bereitstellt, auch wenn das f aus logischer Perspektive redundant ist. Das explizite f soll darauf hinweisen, dass funktionale Rollen effizienter und anders implementiert sind als der $\mathcal{N}$ -Operator (s. Regeln *feature-expansion* für f und *at-least-expansion* für $\mathcal{N}$ in Kap. 6.2.7.1).

Die Sprachklassen werden zum einen verwendet, um Charakteristika der entspr. DLs zu repräsentieren. Diese Charakteristika werden als entspr. „Mixin“-Oberklassen der Sprachklassen angegeben. Beispielsweise ist die Klasse `ALCRPLUS` auch Unterklasse der Klasse `DL-WITH-BLOCKING`, wodurch das Charakteristikum repräsentiert wird, dass ein Blockierungsverfahren für einen Erfüllbarkeitstester für diese DL benötigt wird. Anhand der DL-Spracheninstanz kann ein MIDeLoRA-Beweiser nun leicht erkennen, ob und auch welche Art von Blockierungsverfahren während der Tableau-Expansion benötigt wird.⁹

Zum anderen werden die Vererbungsbeziehungen zwischen den Sprachklassen benötigt, um *die Vererbung von Beweisern entlang der $\mathcal{DL}$ -Achse im MIDeLoRA-Raum zu ermöglichen*. Offensichtlich ist es nicht möglich und auch nicht sinnvoll, für jeden Punkt im MIDeLoRA-Raum einen eigenen Beweiser zu im-

⁸Es handelt sich nicht um ein UML-Diagramm.

⁹Natürlich könnte dieses Wissen auch direkt im Beweiser für $\mathcal{ALC}_{\mathcal{R}+}$ kodiert werden; dieses Schema erlaubt jedoch eine höhere Generizität und daher Wiederverwendbarkeit von Funktionalität.

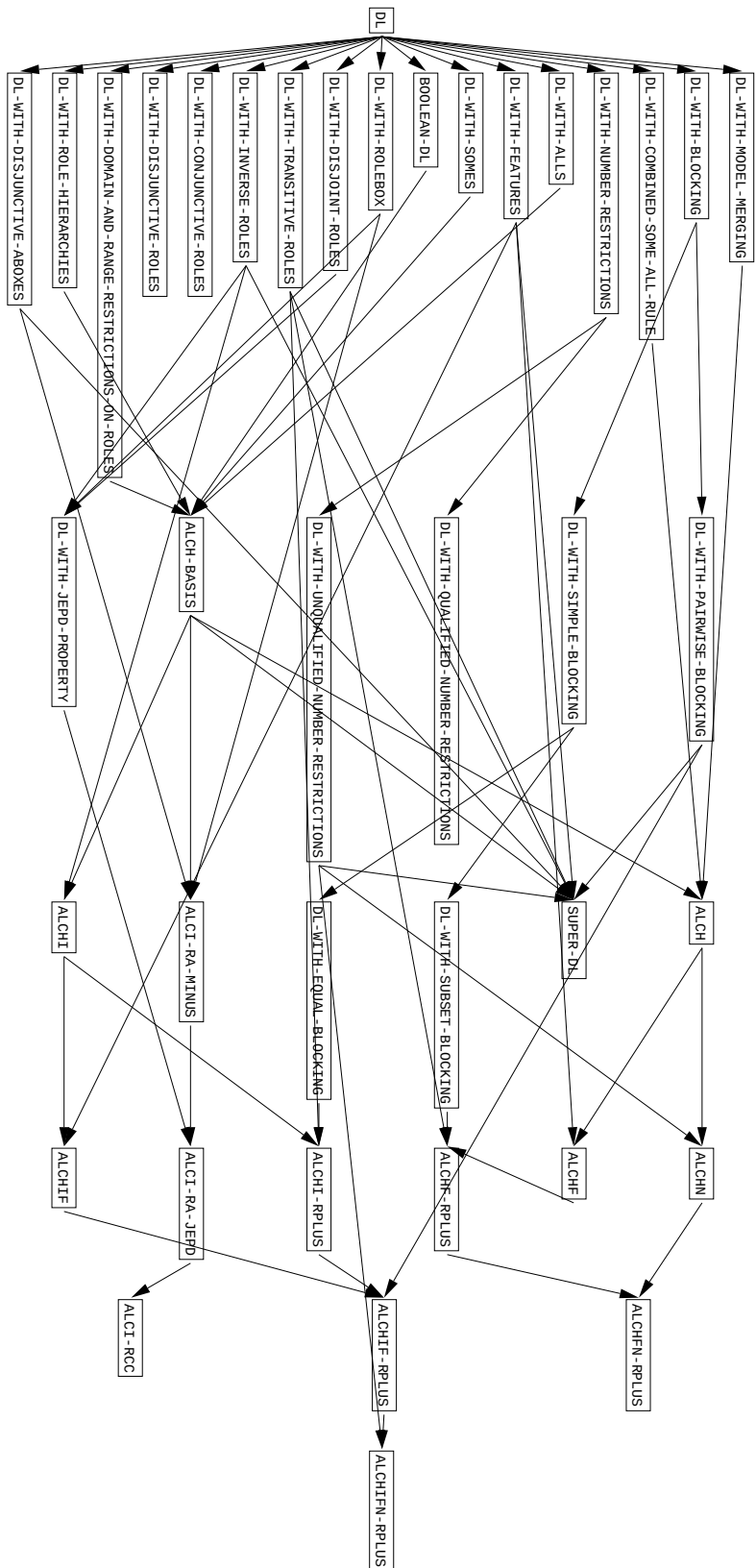


Abbildung 6.4.: Sprachklassen

plementieren; wir müssen mit Beweisern daher Regionen im MiDELORA-Raum abdecken können (s.u.).

Oftmals wird durch eine bestimmte DL, also eine Instanz einer Sprachklasse, auch *eine bestimmte ABox-Klasse gefordert*, also die $\mathcal{K}$ -Koordinate im MiDELORA-Raum bestimmt. Die Standard-ABox-Klasse für eine Sprachklasse kann daher mittels der Klassenmethode `get-abox-type-for-dl` festgelegt werden. So liefert `get-abox-type-for-dl` für die Klasse `ALCIRCC` beispielsweise den Klassennamen `JEPD-ABox` (`JEPD-ABox`), wobei es sich um eine gemeinsame Unterklasse der Klassen `ABox` und `JEPD-Substrat` (`JEPD-Substrate`) handelt.

6.2.4.3. Die TBox

Eine TBox muss zunächst mit einem Namen versehen und als Instanz der Klasse `TBox` angelegt werden. Alsdann kann sie mit Axiomen „gefüllt“ werden. MiDELORA lässt allgemeine TBoxen zu (also GCIs, s. Kap. 2.2.2.4).

Eine bekannte Optimierungstechnik zur Berücksichtigung der durch allgemeine TBoxen gegebenen Axiome im Tableau-Verfahren ist die sog. *GCI-Absorption*, [Horrocks 2003b; 1997a]. Diese wird hier als Schablonenmethode implementiert. Eine GCI-absorbierte, *vorbereitete* TBox ist eine TBox, die äquivalent zur originalen TBox ist, und in der die Menge der TBox-Axiome in eine Menge von *auffaltbaren Axiomen* und eine Menge von *nichtauffaltbaren Axiomen* partitioniert wurde. Letztere Menge gibt Anlass zur Menge der sog. *Meta-Constraints*; für jedes Axiom $C \sqsubseteq D$ in der Menge der nichtauffaltbaren Axiome wird ein Meta-Constraint der Form $\neg C \sqcup D$ erzeugt. Dieses Konzept muss während der Tableau-Expansion zu jedem Knoten-Label hinzugefügt werden.

Die TBox-Axiome sind nach linker und rechter Seite der Implikation indiziert: Methoden wie `get-all-axioms-with-left-side`, `get-all-axioms-with-right-side`, wie sie während der Tableau-Expansion zur Berücksichtigung der TBox z.B. durch das *Faule Auffalten* (s.u.) benötigt werden, können so effizient implementiert werden. Eine TBox kann nach erfolgter Vorbereitung mittels entspr. Methoden (z.B. `unfoldable-p`, `get-meta-constraint`) bzgl. ihrer Eigenschaften befragt werden.

Die Klasse `TBox` hat Attribute zur Referenzierung des verwendeten Konzeptspeichers, der Menge der originalen TBox-Axiome, der Menge der TBox-Axiome nach erfolgter Vorbereitung (Absorption etc.), der Meta-Constraints, etc.

6.2.4.4. Intensionale Standard-Inferenzdienste als Schablonenmethoden

MiDELORA-Beweiser sind stets ABox-Beweiser. Die rein intensionalen (also Konzept- und TBox-spezifischen) Standard-Inferenzdienste werden zunächst

6.2 Entwurf von MIDeLoRA

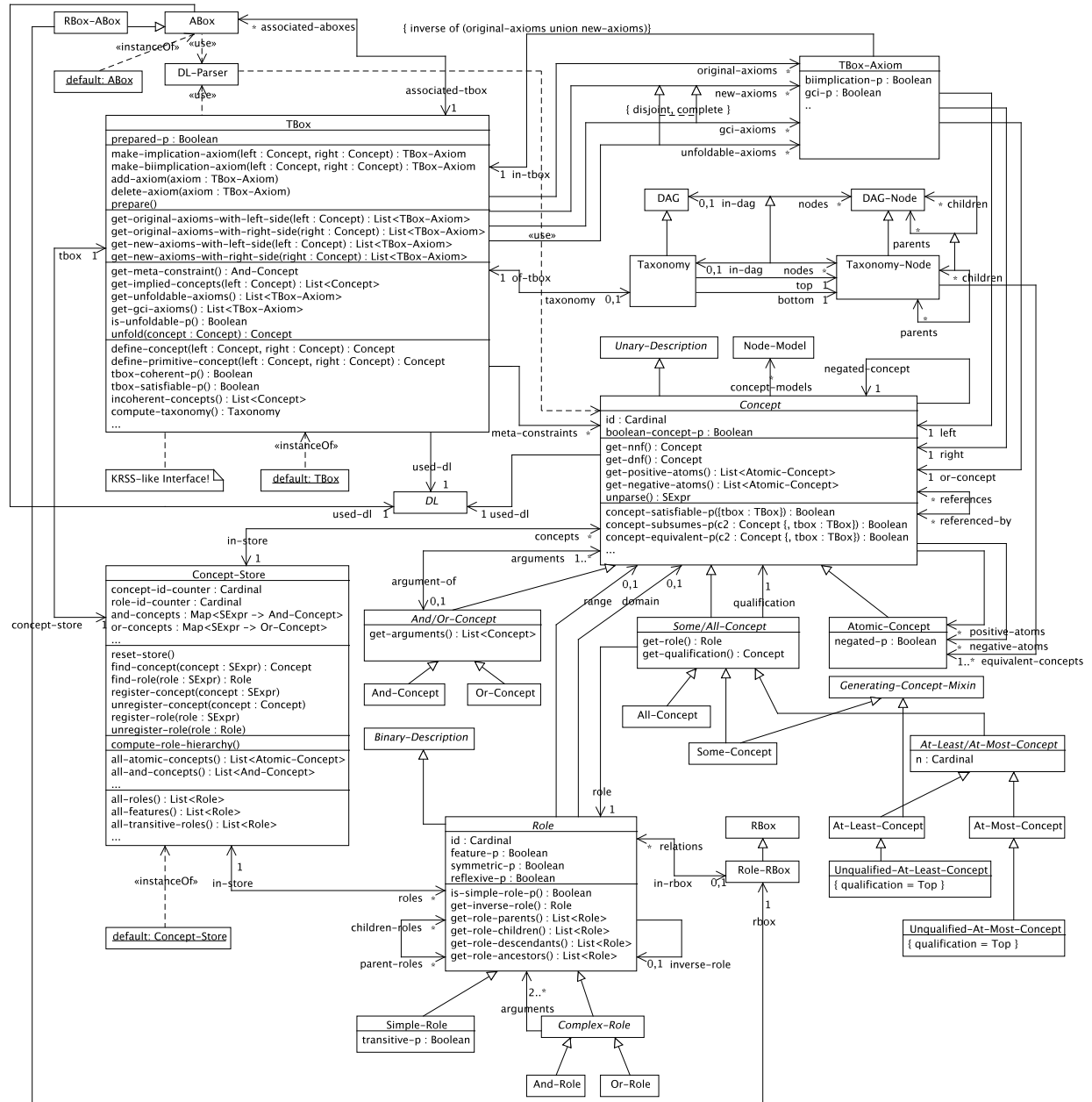


Abbildung 6.5.: Weitere Klassen.

als KRSS-API-Funktionen bereitgestellt (`concept-satisfiable?`, `concept-subsumes?`, ...). Der Aufruf einer dieser API-Funktionen veranlasst zunächst die Lokalisierung oder Erzeugung der Konzept- bzw. TBox-Instanz, da die KRR-Schnittstelle lediglich Namen bzw. Konzepte in KRSS-Syntax akzeptiert (welche ja enkodiert und auf Konzept (Concept)-Instanzen abgebildet werden müssen). Anhand der TBox- oder Konzept-Instanzen wird dann zunächst die verwendete DL bzw. Sprachklasse mittels `get-language` ermittelt. Auf die so gewonnenen Argumente (die nun Instanzen sind) wird nun die der KRSS-API-Funktion entsprechende Schablonenmethode angewendet (`concept-satisfiable-p`, `concept-subsumes-p`, ...). Prinzipiell können diese Methoden natürlich auch überschrieben werden; dies ist jedoch i.d.R. nicht notwendig.

Einige der durch diese Schablonenmethoden implementierten Algorithmen sind recht komplex (z.B. `taxonomy`), andere erfordern lediglich eine Reduktion auf einen oder mehrere Konzepterfüllbarkeitstests (z.B. `concept-subsumes-p`). In den Implementierungen finden jeweils Stand-der-Kunst-Optimierungstechniken aus [Horrocks 2003b] Anwendung (z.B. wird die Anzahl der benötigten Subsumptionstests bei der Taxonomieberechnung mit den Verfahren aus [Baader u. a. 1994] reduziert).

Der Konzepterfüllbarkeitstest ist ebenfalls als Schablonenmethode bereitgestellt. Dieser delegiert seinerseits einfach an einen „passenden“ ABox-Erfüllbarkeitsbeweiser im MiDeLoRA-Raum. Das Konzepterfüllbarkeitsproblem (bzgl. der TBox T) für das Konzept C wird daher auf das ABox- bzw. WB-Erfüllbarkeitsproblem von $(\mathcal{A}, T)$ mit $\mathcal{A} = \{i : C\}$ reduziert. Die Klasse von $\mathcal{A}$ bzw. die $\mathcal{K}$ -Koordinate im MiDeLoRA-Raum wird durch die Methode `get-abox-type-for-dl` bestimmt, nachdem ja bereits die verwendete Sprachklasse DL und somit die $\mathcal{DL}$ -Koordinate im MiDeLoRA-Raum durch die Methode `get-language` zuvor bestimmt wurde. Da nun alle drei Koordinaten feststehen, wird ein Beweiser gerufen, der den Punkt $(\text{abox\_consistent?}, DL, K) \in \mathcal{T} \times \mathcal{DL} \times \mathcal{K}$ im MiDeLoRA-Raum abdeckt. Die ABox $\mathcal{A}$ kann nach erfolgtem Erfüllbarkeitstest gelöscht werden, und die Koordinaten K und DL können durch entspr. optionale Argumente auch manuell bestimmt werden.

Die bereitgestellten CLOS-Schablonenmethoden erlauben insbesondere auch die einfache Integration unvollständiger Verfahren als „Wächter (Guards)“ (s. Kap. 2.2.7.1, S. 102), ohne dass die Implementierungen der primären Schablonenmethoden geändert oder überschrieben werden müssten. CLOS bietet hier sog. *Methodenkombinationen* [Keene 1989]. Beispielsweise kann der vollständige Konzeptsubsumptionstest `concept_subsumes?(C, D)` durch einen unvollständigen sog. *Modellverschmelzungstest* (s.u.) wie folgt bewacht werden: Sind die Konzeptmodelle von C und D verschmelzbar, so impliziert dies bereits, dass die Antwort FALSE ist. In diesem Fall wird die primäre Methode `concept-subsumes-p` nicht gerufen. Diese Bewachung kann in CLOS durch

eine sog. around-Methode für `concept-subsumes-p` realisiert werden. Ist der Modellverschmelzungstest hingegen nicht erfolgreich, so wird die primäre Methode mittels (`call-next-method`) gerufen und die Berechnung fortgesetzt. Ähnliche Techniken sind heute unter dem Schlagwort der *aspektorientierten Programmierung* (*Aspect Oriented Programming, AOP*) u.a. in der Java-Gemeinde bekannt.

6.2.5. Struktur des MIDeLoRA-Raums: Dynamische Auswahl und Wiederverwendung von Beweisern

Der zu verwendende MIDeLoRA-Beweiser wird anhand der Typen (Klassen) dreier Argumente (T, DL, K) durch dynamisches Binden ausgewählt. CLOS bietet Multimethoden, und somit werden die Klassen aller dreier Argumente zur Laufzeitauswahl des Beweisers verwendet.

Bei dem Argument DL handelt es sich um eine Instanz einer Sprachklasse, K ist eine ABox, während T (der Beweiser-Typ) ein einfaches Symbol ist (in CLOS können auch für Symbole Methoden definiert werden). Entlang der Achsen $\mathcal{DL}$ und $\mathcal{K}$ ist Vererbung definiert und nutzbar, nicht jedoch entlang der $\mathcal{T}$ -Achse, denn es ist nicht klar, wie Vererbung zwischen Beweisern, die für verschiedene Inferenzdienste entworfen wurden, definiert und genutzt werden könnte.

Ein MIDeLoRA-Beweiser wird für einen Punkt $(T, DL, K) \in \mathcal{T} \times \mathcal{DL} \times \mathcal{K}$ definiert. Aufgrund der Vererbungsbeziehungen entlang der $\mathcal{DL}$ - und $\mathcal{K}$ -Achse ist dieser Beweiser nun jedoch nicht nur für (T, DL, K) anwendbar, sondern auch für (T, DL', K') , wenn DL' eine Unterklasse von DL , und K' eine Unterklasse von K ist. Zudem ist der Beweiser auch für (T, DL, K') und (T, DL', K) anwendbar. Wir sagen dann, dass der für (T, DL, K) definierte Beweiser an (T, DL', K') (bzw. (T, DL, K') , (T, DL', K)) „vererbt“ wurde. Ein Beweiser definiert dann letztlich eine *Ebene* im MIDeLoRA-Raum, welche die Punkte abdeckt, für die der Beweiser anwendbar ist. Die dynamische Auswahl eines Beweisers für einen Punkt wird nun durch dynamisches Binden von CLOS automatisch unterstützt. Werden bestimmte Punkte durch mehrere Beweiser bzw. „Ebenen“ überdeckt, so wird der *speziellste Beweiser* verwendet. Auch hier ist Spezifität zunächst durch CLOS definiert.

Ein zweiter Blick offenbart jedoch, dass diese Art der Beweiserauswahl durch dynamisches Binden in CLOS nicht in allen Fällen sinnvoll ist:

- Bzgl. der $\mathcal{K}$ -Achse ist die CLOS-Standard-Vererbung gut geeignet, am Beispiel:

Der für $(\text{concept\_instances}, \text{ALC}, \text{ABox})$ definierte Beweiser kann unmittelbar für $(\text{concept\_instances}, \text{ALC}, \text{DB-ABox})$ genutzt werden, da DB-ABox eine Unterklasse von ABox ist.

Die Vererbung entlang der $\mathcal{K}$ -Achse erscheint unkritisch: Eine Instanz einer Unterklasse kann stets für ein Argument einer Oberklasse verwendet werden (Liskov Substitution Principle).

Soll nun `concept_instances` die Spezifika einer ABox-Unterklasse nutzen (z.B. Indexstrukturen), so muss der Beweiser u.U. überschrieben werden. Dies mag auch notwendig werden, um *Vollständigkeit* zu gewährleisten (z.B. bzgl. virtueller Assertionen in einer DB-ABox).¹⁰

- Bzgl. der $\mathcal{DL}$ -Achse ist die Situation nicht so eindeutig. Betrachten wir zur Illustration die DLs $\mathcal{DL}_1$ und $\mathcal{DL}_2$, wobei es sich also um Mengen von Konzepten handelt. Gilt nun $\mathcal{DL}_2 \subseteq \mathcal{DL}_1$, so ist ein $\mathcal{DL}_1$ -Beweiser natürlich auch als $\mathcal{DL}_2$ -Beweiser nutzbar. Ist nun die Sprachklasse $\mathcal{DL}_2$ eine Unterklasse von $\mathcal{DL}_1$, so würden die für $\mathcal{DL}_1$ definierten Beweiser automatisch für $\mathcal{DL}_2$ anwendbar sein.

Existiert nun für $\mathcal{DL}_1$ bereits ein Beweiser, so wird man für $\mathcal{DL}_2$ nur dann einen eigenen Beweiser implementieren (und nicht den für $\mathcal{DL}_1$ definierten Beweiser nutzen bzw. erben), falls der Beweiser für $\mathcal{DL}_2$ Optimierungsverfahren nutzen kann, die für $\mathcal{DL}_1$ nicht anwendbar sind. Natürlich hätte man auch bereits den Beweiser für $\mathcal{DL}_1$ so gestalten können, dass er Eingaben aus $\mathcal{DL}_2$ identifiziert und dann die entspr. Optimierungsverfahren zuschaltet. In MIDELORE entscheiden wir uns bewusst gegen dieses Prinzip, da seine Anwendung aufgrund verschiedenartigster und hochkomplexer Optimierungsverfahren zu einer ausufernden Komplexität des $\mathcal{DL}_1$ -Beweisers führt.

Es fällt auf, dass wir die Vererbungsbeziehungen in der Sprachklassen-Hierarchie in Abb. 6.4 andersrum organisiert haben. Hier wäre $\mathcal{DL}_1$ eine Unterklasse von $\mathcal{DL}_2$ sein, und nicht andersrum. So gilt beispielsweise $\mathcal{ALCH} \subseteq \mathcal{ALCH}_{f_{\mathcal{R}}+}$, doch ist $\mathcal{ALCHFRPLUS}$ in Abb. 6.4 als Unterklasse von $\mathcal{ALCH}$ vorgesehen, und nicht andersrum. Wir verteidigen diese Organisation wie folgt:

- Zum einen erscheint es natürlicher, „speziellere“ DLs (hier $\mathcal{ALCH}_{f_{\mathcal{R}}+}$) als Unterklassen „allgemeinerer“ DLs (hier $\mathcal{ALCH}$) zu definieren, auch wenn hierdurch die Wiederverwendung von Beweisern zunächst unterbunden wird. Wir beschreiben gleich, wie wir dieses Problem umgehen,
- Wäre hingegen $\mathcal{ALCH}$ nun Unterklasse von $\mathcal{ALCHFRPLUS}$, so könnten die Charakteristika der entspr. DL nicht mehr wie hier mittels Mixin-Eigenschaftsklassen (z.B. DL-WITH-BLOCKING) für die Klasse und ihre Unterklassen festgelegt werden. Offensichtlich würde dann DL-WITH-

¹⁰I.d.R. müssen jedoch nur bestimmte Methoden überschrieben, und nicht ganze Beweiser definiert werden.

BLOCKING für ALCHFRPLUS, aber eben auch für ALCH (ohne generelle TBoxen) gelten, was nicht sinnvoll erscheint. Wir müssten dann nichtmonotone Vererbung nutzen, um bestimmte Eigenschaften wie DL-WITH-BLOCKING für spezielle Unterklassen wie ALCH explizit aufzuheben.

Zur Lösung des Wiederverwendungsproblems haben wir drei Möglichkeiten:

1. Wir können auf Wiederverwendung verzichten, müssen dann jedoch sowohl einen $\mathcal{ALCH}$ - als auch einen $\mathcal{ALCH}_{f_{\mathcal{R}^+}}$ -Beweiser implementieren. Die Implementation eines neuen Beweisers ist in MiDeLoRA aufgrund der bereitgestellten Abstraktionen zum Beweiserbau mit relativ wenig Aufwand verbunden (s.u.),
2. Wir können lediglich den spezielleren $\mathcal{ALCH}_{f_{\mathcal{R}^+}}$ -Beweiser implementieren. Der $\mathcal{ALCH}$ -Beweiser wird so implementiert, dass er „manuell“ an den $\mathcal{ALCH}_{f_{\mathcal{R}^+}}$ -Beweiser delegiert, indem z.B. ein „Downcast“ bzw. `change-class` des DL -Argumentes auf ALCHFRPLUS durchgeführt wird.
3. Wir können einen *spezialisierten Auswahlalgorithmus* entlang der DL -Achse implementieren. Dieser Algorithmus interpretiert die Vererbungsbeziehungen der Sprachklassen dann „invers“. Für ein Argument DL wird dann in der Klassenhierarchie solange *verfeinert*, bis eine Sprache DL' gefunden wird, für die ein Beweiser des Typs (T, DL', K) definiert ist. Offensichtlich ist dieser Beweiser auch fähig, das Erfüllbarkeitsproblem der generelleren Sprache zu entscheiden.

In unserem Falle würde also eine Spezialisierung entlang der Sprachklassen-Hierarchie von $DL = \text{ALCH}$ bis zu $DL' = \text{ALCHFRPLUS}$ erfolgen (sofern kein $\mathcal{ALC}$ -Beweiser definiert wurde), und der $\mathcal{ALCH}_{f_{\mathcal{R}^+}}$ -Beweiser würde für $\mathcal{ALC}$ genutzt.

Tatsächlich wäre es in CLOS sogar möglich, dieses Verhalten durch Definition eigener Methodenkombinationen für die BeweiserAuswahl zu realisieren. Wir haben von diesen Möglichkeiten jedoch keinen Gebrauch gemacht.

Manchmal ist es auch sinnvoll, zwei Beweiser (T, DL, K) und (T, DL, K') für verschiedene ABox-Klassen K und K' zu definieren. Dies sei anhand zweier Beweiser $(\text{abox\_consistent?}, \mathcal{ALC}_{\mathcal{R}^+}, *)$, mit $* \in \{K, K'\}$ verdeutlicht:

- Das $\mathcal{ALC}_{\mathcal{R}^+}$ -Standard-Tableau-Verfahren [Sattler 1996b] erfordert zur Behandlung von Rollen-Assertionen für transitive Rollen keinen transitiven Abschluss bzgl. dieser.

Sei R eine transitive Rolle, und gelte $i : \forall R.C$ im Tableau. Offensichtlich muss nun die Qualifikation C nicht nur an direkte R -Nachfolger von i ,

sondern auch an *indirekte* R -Nachfolger propagiert werden.

Betrachten wir die $\forall$ -Regel in $\mathcal{ALC}$, so wird C lediglich an direkte Nachfolger propagiert, da nur explizite Tableau-Rollen-Assertionen betrachtet und somit als R -Nachfolger erkannt werden. Sind nun die Tableau-Rollen-Assertionen $(i, j) : R$ und $(j, k) : R$ gegeben, so ist $(i, k) : R$ eine logische Konsequenz, doch wird diese „Abschluss“-Tableau-Rollen-Assertion vom $\mathcal{ALC}_{\mathcal{R}^+}$ -Kalkül nicht erzeugt. Stattdessen verwendet der $\mathcal{ALC}_{\mathcal{R}^+}$ -Kalkül eine *spezialisierte* $\forall$ -Regel, die sicherstellt, dass bei Vorliegen der Assertionen $(i, j) : R$ und $i : \forall R.C$ nicht nur $j : C$, sondern auch $j : \forall R.C$ zum Tableau hinzugefügt wird.¹¹

Die Expansion von

$$\mathcal{A}_0 = \{i : (\exists R.\exists R.\neg C) \sqcap \forall R.C\}$$

führt dann zunächst zu

$$\mathcal{A}_1 = \mathcal{A}_0 \cup \{(i, j) : R, j : \exists R.\neg C, j : C, j : \forall R.C\}$$

und schließlich zu

$$\text{führt dann } \mathcal{A}_2 = \mathcal{A}_1 \cup \{(j, k) : R, k : C, k : \neg C\},$$

sodass der Widerspruch offensichtlich ist.

- Alternativ könnte eine ABox der Klasse RBox-ABox (RBox-ABox) zur Tableau-Repräsentation verwendet werden, die sowohl eine ABox als auch ein RBox-Substrat (RBox-Substrate) ist, und die somit die Funktionalität zum *Ab-schließen einer relationalen Struktur unter einer Menge von Rollen-Axiomen* ererbt hat. Diese Tableau-Repräsentation kann also implizite R -Kanten bzw. Rollen-Assertionen mittels *complete-substrate* (s. Kap. 5.1.2.6.4, S. 257) explizit machen. In diesem Fall würde die „normale“ $\forall$ -Regel aus $\mathcal{ALC}$ genügen.

Die Expansion von

$$\mathcal{A}_0 = \{i : (\exists R.\exists R.\neg C) \sqcap \forall R.C\}$$

würde zunächst zu

$$\mathcal{A}_1 = \mathcal{A}_0 \cup \{(i, j) : R, j : \exists R.\neg C, j : C\}$$

und dann zu

$$\mathcal{A}_2 = \mathcal{A}_1 \cup \{(i, k) : R, (j, k) : R, k : C, k : \neg C\},$$

führen; wiederum ist der Widerspruch dann offensichtlich. Hierbei wurde $(i, k) : R$ durch *complete-substrate* erzeugt.

Die Berechnung des transitiven Abschlusses mittels *complete-substrate*

¹¹Die zugrundeliegende Idee kommt aus der Modallogik, denn hier gilt für die Modallogik K4 $\Box p \rightarrow \Box \Box p$, wobei p eine beliebige modallogische Formel ist, s. Kap. 2.

kann durch eine spezialisierte $\exists$ -Regel erfolgen. Alternativ können die Abschluss-Rollen-Assertionen als *virtuelle Assertionen* betrachtet werden, deren Erzeugung und Materialisierung wie beschrieben durch die Iteratoren veranlasst wird.

MIDeLoRA ermöglicht also vergleichende Untersuchungen von Beweisern, die verschiedene Tableau-Repräsentationen verwenden. Offensichtlich unterscheiden sich die beiden $\mathcal{ALC}_{\mathcal{R}^+}$ -Beweiser nur in wenigen Details voneinander. Die Regeln sind daher so zu gestalten, dass sie durch Parametrierung bzw. Konfigurierung möglichst für beide Beweiser einsetzbar und wiederverwendbar werden. Beispielsweise können beide Beweiser die selbe $\forall$ -Regel-Implementierung nutzen, wenn diese so parametriert werden kann, dass sie auch transitive $\forall R.C$ -Konzepte wie besprochen weiter propagieren kann.

6.2.6. Softwaretechnische Abstraktionen für MIDeLoRA-Beweiser

MIDeLoRA-Erfüllbarkeitsbeweiser sind Tableau-Beweiser. Zur Definition von MIDeLoRA-Tableau-Beweisern werden daher softwaretechnische Abstraktionen identifiziert, die möglichst direkt den mathematischen Konstrukten bzw. „Bestandteilen“ dieser Inferenzverfahren entsprechen sollen. Die mathematischen Konstrukte eines Tableau-Verfahrens sind letztlich durch

1. das Tableau,
2. die Tableau-Regeln, und
3. die Tableau-Regelanwendungsstrategie gegeben,

s. Kap. 2.2.7.2, S. 102. Wir hatten zudem in Kap. 6.2.3.4, S. 440 eine einfache Implementation dieses Verfahrens diskutiert (welches sich im Detail jedoch von dem in Kap. 2.2.7.2 vorgestelltem Verfahren unterscheidet, insbesondere bzgl. der Tableau-Repräsentation). Wir haben daher sowohl die mathematische als auch die softwaretechnische Perspektive eingenommen. Da Beweiser im MIDeLoRA-Raum als Softwareabstraktionen vorgesehen sind, nehmen wir als viertes Element in obiger Liste noch die Abstraktion des Beweisers auf. Die vier primären in MIDeLoRA bereitgestellten Abstraktionen zur Konstruktion von Beweisern können wir wie folgt beschreiben:

Das Tableau: Ein Tableau ist eine Instanz der Klasse ABox. Dies bedarf (aufgrund der vorangegangenen Diskussionen) keiner weiteren Erläuterungen.

Die Regel: Eine Regel hat einen Namen und wird ebenfalls für eine bestimmte Aufgabe, ABox- und DL-Sprachklasse definiert. Regeln können dabei sowohl logische als auch nichtlogische Aufgaben wahrnehmen (eine Tableau-Regel ist eine logische, und eine Speicherverwaltungs-Regel eine nichtlogische Regel).

Die Regeln sollten möglichst wiederverwendbar sein, um so dem generischen Charakter der Tableau-Verfahren Rechnung zu tragen: So sollte z.B. die $\Box$ -Regel in Beweiser für beliebige DLs (die den $\Box$ -Operator bieten) ohne Änderung übernommen werden können. Eine entspr. Regel wird sich in jedem Tableau-Verfahren für solche DLs finden. In Kap. 6.2.5 hatten wir anhand zweier verschiedener $\mathcal{ALC}_{\mathcal{R}^+}$ -Tableau-Verfahren mit unterschiedlichem Verhalten der $\forall$ -Regel erläutert, dass prinzipiell eine Implementierung des Beweisers ausreichen würde, wenn das Verhalten der $\forall$ -Regel durch entspr. Anpassung parametrisiert werden kann (s. auch Kap. 6.1.3). Wird eine entspr. Verhaltensvariabilität und Parametrierbarkeit bereits beim Entwurf der Regeln berücksichtigt, so erhöht dies das Wiederverwendungspotential. Dies ist wünschenswert, da die Implementation optimierter Tableau-Regeln sehr komplex ist. Daher wird ein umfangreiches Arsenal an wiederverwendbaren Standard-Regeln bereitgestellt, die zudem entspr. parametrisiert werden können.

Wir beschreiben im Folgenden die speziell für Tableau-Regeln bereitgestellten Abstraktionen.

Eine Regel wird zunächst ihre Anwendbarkeit prüfen, indem sie den zur Entscheidung ihrer Anwendbarkeit relevanten Teil des Tableaus „betrachtet“. Oftmals kann eine Regel an mehreren Stellen des Tableaus angewendet werden (z.B. die $\Box$ -Regel, wenn unterschiedliche Knoten i, j mit nichtexpandierten Konjunktionen existieren). In diesem Fall wird die *lokale Strategie der Regel* eine Konfliktresolution vornehmen. Die lokale Strategie ist direkt in der Regel kodiert.

Einige Regeln sind zudem *nichtdeterministisch*, d.h., sie können mehr als eine *Nachfolgekongfiguration bzw. Nachfolge-ABox* herstellen, beispielsweise die $\sqcup$ -Regel. Eine Nachfolgekongfiguration ist per Definitionem immer widerspruchsfrei bzw. konsistent. Wie beschrieben, muss Nichtdeterminismus durch ein Suchverfahren aufgelöst werden. Das Suchverfahren resultiert zur Laufzeit in einem Suchbaum. Zur Repräsentation des Suchbaums können in MIDELORE entweder der COMMON LISP-Keller oder entspr. Attribute der Befehlsobjekte genutzt werden.

Prinzipiell ist jede Regel selbst für die *Enumerierung der Nachfolgekongfigurationen* zuständig. Die Reihenfolge, in der die Regel die Nachfolgekongfigurationen enumeriert, bestimmt letztlich die Suchstrategie (ob z.B. Tiefen- oder Breitensuche vorgenommen wird). Es ist wünschenswert, die Regeln diesbez. flexibel zu halten (durch Parametrierung).

Für die Regeln verwenden wir ein durch Prolog inspiriertes *5-Port-Modell*, vgl. z.B. [Cordes u. a. 1992, Kap. 7.2]. Eine Regel hat 2 Eingänge und 3 Ausgänge, gemeinsam „Ports“ genannt, s. Abb. 6.6. Eine globale Regelanwendungsstrategie kann dann die einzelnen Regeln durch Verbindung ih-

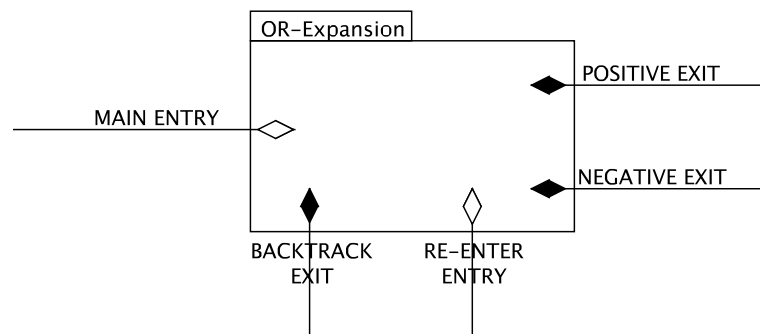


Abbildung 6.6.: Eine Regel nach dem 5-Port-Modell.

rer Ports zusammenschalten, wobei eine Daten- und Kontrollflussmetapher verwendet wird:

- Eine Regel hat 2 Eingänge:

Der Haupteingang (Main Entry): Über den Haupteingang bekommt die Regel die ABox und andere benötigte Argumente sowie (optionale) Argumente zur Parametrierung als Eingabe.

Durch Betreten des Haupteinganges wird *konzeptuell* (s.u. für *Ausnahmen*) eine frische Inkarnation der Regel erzeugt, die sog. *Regel-Inkarnation*.¹² Die Inkarnation entspricht einem *Knoten im Suchbaum*.

Bei Verwendung der Keller-Repräsentation des Suchbaums entspricht ein solcher Knoten einem Bindungsrahmen. Bindungsrahmen werden durch Funktions- bzw. Methoden-Aufrufe erzeugt; daher entspricht dem Betreten des Haupteinganges dann ein „Funktionsaufruf der Regel“. Tatsächlich müssen jedoch nur für nichtdeterministische Regeln neue Knoten im Suchbaum und somit Bindungsrahmen erzeugt werden. Wie erwähnt, kann der Suchbaum auch durch die Befehlshistorie repräsentiert werden. Hier wird dann der Zustand der sonst im Bindungsrahmen vermerkten lokalen Variablen in entspr. Attributen eines protokollierenden Befehlsobjektes gespeichert. Wir verwenden im Folgenden primär die Kellerrepräsentation des Suchbaums, weisen aber darauf hin, dass insbesondere die MIDELoRA-Beweiser (s.u.) bei einem Wechsel der Suchbaumrepräsentation nicht verändert werden müssen (lediglich die Regeln müssen entspr. angepasst werden, was wiederum durch einfach Parametrierung geschehen kann).

¹²Im Folgenden unterscheiden wir nicht mehr streng zwischen Regel und Regel-Inkarnation; der Kontext wird jeweils deutlich machen, was gemeint ist.

Die Regel wird als erste Aktion nach Betreten des Haupteinganges nun ihre Anwendbarkeit prüfen. Fällt der Test positiv aus, so wird sie eine konsistente bzw. widerspruchsfreie Nachfolgekonfiguration herstellen. Eine *nichtdeterministische* Regel kann auf Anforderung u.U. mehr als eine Nachfolgekonfiguration herstellen. Die Regel gibt dann die Kontrolle durch Verlassen über einen entsprechenden *Ausgang* (s.u.) weiter.

Eine Regel hat einen lokalen Kontext. Der Kontext enthält u.a. eine Referenz auf das *aktuelle Befehlsobjekt* und weitere lokale Variablen der Regel, z.B. bzgl. der bereits erzeugten Nachfolgekonfigurationen im Falle einer nichtdeterministischen Regel. Für nichtdeterministische Regeln muss der lokale Kontext der Regel beim Betreten des Haupteinganges durch einen Bindungsrahmen auf dem Keller (und somit durch einen Funktionsaufruf) oder in entspr. Attributen des Befehlsobjektes bewahrt werden, damit eine Wiederherstellung des Kontextes nach einem Zurücksetzen zu diesem Knoten im Suchbaum im Rahmen des Suchverfahrens ermöglicht wird.

Der Rückkehr-Eingang (Re-Enter Entry): Dieser Eingang wird verwendet, wenn die Kontrolle an die Regel *zurückgegeben wird*. Die Kontrolle wird stets von einer Kind-Regel-Inkarnation (mittels *return*) bzw. von einem Kind-Knoten im Suchbaum zurückgegeben. Die Kontrolle wird immer dann zurückgegeben, wenn

1. bei der Expansion ein Widerspruch aufgetreten ist, oder aber
2. bereits eine Vervollständigung gefunden wurde und der ABox-Erfüllbarkeitstest somit bereits positiv ausgefallen ist, aber noch eine *alternative Vervollständigung* angefordert wurde. Wir hatten erwähnt, dass MiDELORA prinzipiell mehr als eine Vervollständigung einer ABox konstruieren können soll.¹³

Im ersten Fall zeigt der mittels *return* an die Eltern-Inkarnation übergebene Rückgabewert an, welche Regel-Inkarnation bzw. welche Knoten im Suchbaum *für die Beseitigung des Widerspruchs (mit)verantwortlich ist*. Wir beschreiben später, wie die Regel-Inkarnation feststellen kann, ob sie (mit)verantwortlich ist oder nicht.

Je nachdem, ob die Regel, die den Rückgabewert und die Kontrolle zurückerhalten hat, nun (mit)verantwortlich für die Beseitigung des Widerspruchs ist oder nicht, wird wie folgt verfahren:

1. Ist die Regel selbst für den Widerspruch (mit)verantwortlich,

¹³Dies entspricht der Eingabe des „;“ in Prolog.

so muss sie versuchen, den Widerspruch zu beseitigen.

Sie verwendet dazu ihren nun wiederhergestellten lokalen Kontext, um den Zustand der ABox wiederherzustellen, der beim Betreten des Haupteinganges der Regel vorhanden war. Dieser Zustand wird durch ein *Zurückrollen der Befehlshistorie* (s. Methode `rollback-to`) bis zu dem im lokalen Kontext der Regel vermerkten Befehlsobjekt (Referenz auf `current-action`) wiederhergestellt. Der wiederhergestellte Zustand ist per Definitionem konsistent (enthält keinen Widerspruch). Die Regel ist daher selbst für das Wiederherstellen des ABox-Zustands im Rahmen des Zurücksetzens verantwortlich; anders als im diskutierten funktionalen *ALC*-Beweis (s. Kap. 6.2.3.4) geschieht dies nicht „automatisch“.

Die Regel-Inkarnation wird nun die Berechnung wie folgt fortsetzen:

- Kann die Regel nun noch weitere alternative Nachfolgekonfigurationen erzeugen (handelt es sich also um eine nichtdeterministische Regel, die noch nicht alle Möglichkeiten enumeriert hat), so wird die nächste Nachfolgekonfiguration erzeugt, und der *positive Ausgang* wird genommen, um die Berechnung fortzusetzen.
 - Kann die Regel keine alternative Nachfolgekonfigurationen erzeugen, so wird sie die Kontrolle (mittels `return`) an die Eltern-Inkarnation zurückgeben; die selbst erhaltenen Rückgabewerte werden ihrerseits zurückgegeben. So findet Rücksetzen in weiter oben liegende Teile des Suchbaums statt.
2. Ist die Regel nicht für den Widerspruch (mit)verantwortlich, so wird sie die Kontrolle (mittels `return`) an die Eltern-Inkarnation zurückgeben; die selbst erhaltenen Rückgabewerte werden ihrerseits zurückgegeben. So findet Rücksetzen in weiter oben liegende Teile des Suchbaums statt.
- Eine Regel hat 3 Ausgänge:

Der positive Ausgang (Positive Exit) wird verwendet, wenn die Regel anwendbar war. Sie hat dementsprechend eine widerspruchsfreie Nachfolgekonfiguration produziert, die ABox also modifiziert.

Für den positiven Ausgang wird durch die globale Regelanwendungsstrategie definiert, wie die Berechnung mit dieser Nachfolgekonfiguration fortzusetzen ist. Hier wird normalerweise eine Regel zur Weiterverarbeitung gerufen.

Der negative Ausgang (Negative Exit) wird verwendet, wenn die Regel nicht anwendbar war.

Wird der negative Ausgang genommen, so wird entweder anhand der globalen Regelanwendungsstrategie die nächste Regel auf Anwendbarkeit überprüft, oder aber eine Vervollständigung wurde konstruiert.

Der Zurücksetz-Ausgang (Backtrack Exit) wird verwendet, wenn die Regel die Kontrolle mittels `return` zurückgibt. Die Kontrolle wird dann an die Eltern-Regel-Inkarnation zurückgegeben, wozu ihr *Rückkehr-Eingang* verwendet wird. Über den Rückkehr-Eingang werden Rückgabewerte übergeben. Wie beschrieben, wird dieser Ausgang verwendet, wenn

1. bei der Expansion ein Widerspruch aufgetreten ist, oder aber
2. bereits eine Vervollständigung gefunden wurde und der ABox-Erfüllbarkeitstest somit bereits positiv ausgefallen ist, aber noch eine alternative Vervollständigung angefordert wurde.

Obwohl die meisten Regeln mit diesen Ports auskommen werden, sehen wir dennoch eine Möglichkeit vor, weitere Ports definieren zu können. Die Liste der oben angegebenen Ports ist also nicht fix.

Einige Regeln werden zudem keine getrennten positiven und negativen Ausgänge spezifizieren. Ist es für die nachfolgenden Regelanwendungen unerheblich, ob die Regel angewendet wurde oder nicht, so kann lediglich ein Ausgang definiert werden.

Die globale Regelanwendungsstrategie (auch: Regelanwendungskette): Hier werden die einzelnen Regeln in einer Art *Regelanwendungskette* „zusammengeschaltet“, wodurch die vom Tableau-Beweiser verwendete globale Regelanwendungsstrategie bestimmt wird. Die Ports der Regeln werden durch diese Regelanwendungskette fest miteinander verdrahtet. Dies sorgt dafür, dass eine globale Regelanwendungsstrategie bei der Tableau-Expansion eingehalten wird. Zusätzlich verwendet jede Regel eine (enodierte) lokale Strategie.

Die Strategie ist fix, da i.d.R. nur ihre Einhaltung die Qualitätskriterien des Tableau-Kalküls (Korrekt, Vollständigkeit, Terminierung) sicherstellt. Aus diesem Grund verwenden wir beispielsweise nicht das *Strategie-Entwurfsmuster* aus [Gamma u. a. 1996], welches das Kapseln eines Strategie-Algorithmus' durch ein Strategie-Objekt und somit wesentlich mehr „Dynamik“ ermöglichte (z.B. könnte die Strategie während des Beweises gewechselt werden).

Im einfachen funktionalen *ALC*-Beweiser (s. Kap. 6.2.3.4) wird die Strate-

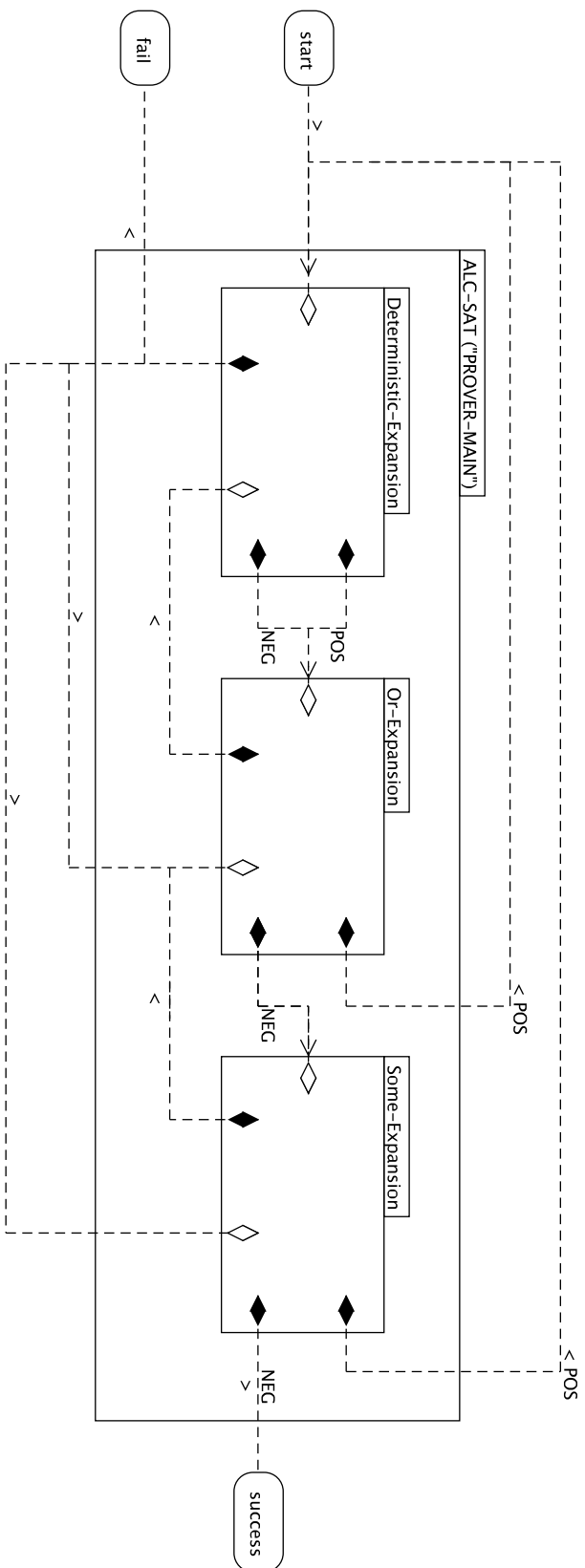


Abbildung 6.7.: Die Regelanwendungskette des einfachen *ALC*-Beweisers.

gie verwendet, dass zunächst *aussagenlogische Expansionen* erschöpfend vorgenommen werden, bevor die Regeln zur Befriedigung von $\exists R.C$ - und $\forall R.C$ -Assertionen angewendet werden. Wir können diesen Beweiser in drei „Regel-Abschnitte“ zerlegen: Eine Regel zur $\sqcap$ -Expansion, eine zur $\sqcup$ -Expansion, und eine kombinierte $\exists\forall$ -Regel (hier sind die $\exists$ - und $\forall$ -Regel wie in der Implementation aus Kap. 6.2.3.4 verschmolzen; aufgrund der Sub-Tableau-Eigenschaft ist dies für *ALC* möglich). Die „Verschaltung“ dieser Regeln ist durch die geschachtelten bedingten (if-) Anweisungen gegeben. Wir können diese Regeln und deren Verschaltung im 5-Port-Modell deuten. Die *alc-sat1*-Implementation aus Kap. 6.2.3.4 entspricht dann in der Grobstruktur der in Abb. 6.7 dargestellten Regelanwendungskette im MIDELOA-Modell.

Die äußere, mit ALC-SAT (PROVER-MAIN) beschriftete Box entspricht der *alc-sat1*-Funktion. Die innerhalb dieser Box miteinander verketteten Regeln visualisieren die Regel-Verkettungen. Ein Funktionsaufruf der ALC-SAT (PROVER-MAIN)-Funktion wird durch eine gerichtete Verbindung angezeigt, die von außerhalb nach innerhalb der Box verläuft. Verläuft die Verbindung innerhalb der Box, so findet keine Funktionsinkarnation statt.

War beispielsweise die OR-EXPANSION-Regel anwendbar, so wird der positive Ausgang genommen. Da die Verbindung vom positiven Ausgang der OR-EXPANSION-Regel zum Haupteingang der DETERMINISTIC-EXPANSION-Regel außerhalb der PROVER-MAIN-Box verläuft, wird also eine neue Funktionsinkarnation von PROVER-MAIN gestartet. Analog werden *returns* visualisiert; die Kontrolle wird dabei i.d.R. von einem Zurücksetz-Ausgang an einen Rückkehr-Eingang übergeben. Verlaufen diese innerhalb von ALC-SAT (PROVER-MAIN), so handelt es sich um keine echte Rückkehr aus einer Funktionsinkarnation (bzw. der Keller wird nicht abgebaut).

Der Beweiser: Ein MIDELOA-Beweiser wird für eine bestimmte Aufgabe, eine DL, sowie eine ABox-Klasse definiert, wie mehrfach diskutiert.

Ein Beweiser durchläuft im MIDELOA-Modell drei voneinander zu trennende *operationale Phasen*:

Die Initialisierungsphase wird verwendet, um die ABox für die entsprechende Aufgabe vorzubereiten. Diese Phase wird durch die Methode *prover-init* initiiert. Je nach Aufgabe des Beweisers sind hier verschiedene Aufgaben denkbar.

Für einen ABox-Erfüllbarkeitsbeweiser muss beispielsweise für jedes ABox-Individuum die Zuweisung *unexpanded-concepts := told-concepts* durchgeführt werden, *Meta-Constraints* müssen hinzugefügt werden, und Knoten aktiviert werden. Zudem muss u.U. ein *Prä-Tab-*

leau hergestellt werden (s.u.).

Nach erfolgter Vorbereitung wird die Hauptphase des Beweisers aufgerufen.

Die Hauptphase führt den eigentlichen Beweis durch; sie wird durch die Methode `prover-main` implementiert. Handelt es sich um einen Tableau-Beweiser, so kann die Hauptphase durch die globale Regelanwendungsstrategie wie soeben beschrieben definiert werden.

Die Endphase wird aus der Hauptphase heraus initiiert; sie wird durch die Methode `prover-success` implementiert. Handelt es sich um einen Tableau-Beweiser, so wird diese Endphase initiiert, sobald eine Vervollständigung gefunden wurde.

In der Endphase können z.B. Operationen wie das Speichern einer gefunden Vervollständigung (durch Kopieren), das Berechnen und Abspeichern von *Individuumsmodellen* (s.u.) anhand der Vervollständigung etc. stattfinden. Prinzipiell kann auch eine alternative Vervollständigung angefordert und die Kontrolle somit an die Hauptphase zurückgegeben werden.

Ein MIDeLoRA-Beweiser ist konzeptionell also als *dreiphasiges und dreistufiges Kompositum mit hierarchischer Struktur „Beweiser-Strategie-Regel“* angelegt. Diese identifizierten softwaretechnischen Abstraktionen werden in MIDeLoRA durch COMMON LISP-Makrosprachen bereitgestellt, wodurch sie zu programmiersprachlichen Abstraktionen werden. Wir behaupten, dass die identifizierten Abstraktionen den Entwurf und die Implementierung von Tableau-Beweisern wesentlich erleichtern.

Die vorgenommene Zerlegung in einzelne, kombinierbare objektorientierte Basis-Bausteine resultiert in einer (noch zu demonstrierenden) hohen Flexibilität und Kombinierbarkeit, die (nach Meinung des Autors) Resultat einer sauberen Trennung der unterschiedlichen Aspekte eines Beweisers ist.

6.2.7. Ein Arsenal von Tableau-Regeln

MIDeLoRA bietet sowohl Standard- als auch Spezial-Regeln. Die Regeln ermöglichen die Bottom-Up-Konstruktion von Beweisern; wir verwenden hier *Bottom-Up-Konstruktion bzw. Bottom-Up-Entwurf* im Sinne von Paul Graham [Graham 1994, Kap. 1.2].

Wir stellen zunächst das Arsenal an Standard-Regeln vor. Dann stellen wir die Spezial-Regeln vor, die für Nicht-Standard-DLs erforderlich werden. Die Tableau-Regeln werden auf einer konzeptionellen Ebene erläutert. Berücksichtigte Optimierungstechniken werden hingegen im nächsten Unterkapitel dargestellt.

Im Idealfall ist das bereitgestellte Regel-Arsenal zur Implementierung eines neuen MIDeLoRA-Beweisers ausreichend; diese müssen dann lediglich in einer entsprechenden Regelanwendungskette zusammengeschaltet und entspr. parametrisiert bzw. konfiguriert werden.

6.2.7.1. Standard-Tableau-Regeln

MIDeLoRA bietet die folgenden Standard-Tableau-Regeln:

Die Regel deterministic-expansion: Die Regel ist deterministisch (erzeugt also nur eine Nachfolgekonfiguration); es handelt sich hierbei um eine „Makro-Regel“, die mehrere Einzel-Regeln zusammenfasst:

- Die $\sqcap$ -Regel, s. Abb. 2.5.
- Die $\forall$ -Regel, s. Abb. 2.5, wenn gewünscht. Diese in der Regel deterministic-expansion *eingebettete* $\forall$ -Regel ist durch Parametrierung deaktivierbar, da für DLs wie $\mathcal{ALC}$ eine *kombinierte* $\exists\forall R$ -Regel angegeben werden kann (s. Regel some-expansion), wie auch im einfachen $\mathcal{ALC}$ -Beweiser aus Kap. 6.2.3.4. In diesem Falle soll innerhalb der Regel deterministic-expansion nicht erneut nach anwendbaren Rollenqualifikationen gesucht werden.

Die hier eingebettete $\forall$ -Regel ist jedoch genereller als die entspr. Regel im $\mathcal{ALC}$ -Beweiser aus Kap. 6.2.3.4, da hier nun auch die Effekte von Rollenhierarchien, inversen und transitiv abgeschlossenen Rollen berücksichtigt werden, wie es zur Unterstützung ausdrucksmächtigerer DLs erforderlich wird. Gilt nun $i : \forall R.C \in \mathcal{A}$, so fügt die Regel $j : C$ zu allen $j \in V$ (bzw. zur Liste der unexpandierten Konzepte dieser) hinzu, für die $\mathcal{A} \rightsquigarrow (i, j) : R$ gilt (s. Lemma 5, S. 323).

Die $\rightsquigarrow$ -Relation kann nun so parametrisiert werden, dass sie die Effekte durch Rollenhierarchien, inverse und/oder transitive Rollen beachtet oder eben nicht beachtet (dies entspricht einem „Ausschalten“ der entspr. Fälle in Lemma 5).

Die eingebettete $\forall$ -Regel sieht dann wie folgt aus:

$\forall$ -Regel:

wenn

1. $x : \forall R.C \in \mathcal{A}$
2. $\mathcal{A} \rightsquigarrow (x, y) : R$
3. $y : C \notin \mathcal{A}$

dann $\mathcal{A}' := \mathcal{A} \cup \{y : C\}$

Gilt nun $\mathcal{A} \rightsquigarrow (i, j) : R$, so wollen wir j im Folgenden einen R -

Nachfolger von i nennen, womit wir Def. 40, S. 74 verfeinern. Die R -Nachfolger werden von der $\forall$ -Regel durch Verwendung des Iterators *loop-over-role-successors* bestimmt. Durch entspr. Argumente kann vom Iterator die obige Parametrierung der $\leadsto$ -Relation veranlasst werden, sodass lediglich bestimmte Arten von R -Nachfolgern betrachtet werden. Aus der Perspektive der $\forall$ -Regel kann somit die Komplexität der R -Nachfolger-Berechnung einfach in den entspr. Iterator verlagert werden.

- Unexpandierte Konzepte, die die Existenz von Nachfolgern fordern ($\exists R.C, \geq_n R$), werden von der Regel als expandiert gekennzeichnet, wenn die entsprechenden R -Nachfolger bereits existieren.

Schließlich integriert die Regel noch eine Reihe von Techniken, die semantisch nicht erforderlich sind, sondern Optimierungsmethoden darstellen. Dementsprechend werden diese erst im nächsten Abschnitt erläutert:

- Die Regel zur *Berücksichtigung der TBox-Axiome durch faules Auffalten*.
- Die Regel zur Durchführung der *aussagenlogischen Constraintpropagation (BCP)*.
- Die Regel zur *Expansion deterministischer $\sqcup$ -Konzepte*.

Die Regel or-expansion: Diese Regel ist für das Expandieren von $\sqcup$ -Konzepten verantwortlich und entspricht der $\sqcup$ -Regel, s. Abb. 2.5.

Die Regel kann so *konfiguriert bzw. parametriert* werden, dass sie *syntaktisches oder semantisches Verzweigen* verwendet (s. auch Kap. 6.1.3). Semantisches Verzweigen ist als Optimierungsmethode zu werten und wird im nächsten Abschnitt dargestellt; der funktionale $\mathcal{ALC}$ -Beweiser aus Kap. 6.2.3.4 verwendet syntaktisches Verzweigen.

Wird semantisches Verzweigen (s.u.) verwendet, so wird die Regel genau 2 Nachfolgekonfigurationen erzeugen ($a, \neg a$), andernfalls jedoch n Nachfolgekonfigurationen (wobei n die Anzahl der Disjunkte des zu expandierenden $\sqcup$ -Konzeptes ist). Die Regel muss eine neue Inkarnation zur Bewahrung des lokalen Kontextes starten, da sie nichtdeterministisch ist.

Die Regel some-expansion: Diese Regel ist für das Expandieren von $\exists R.C$ -Konzepten für nichtfunktionale Rollen $R \notin \mathcal{N}_{\mathcal{F}}$ verantwortlich und entspricht der $\exists$ -Regel, s. Abb. 2.5.

Die Expansion von $\exists R.C \in L_V(i)$ führt i.d.R. zur Erzeugung eines R -Nachfolgers j , $(i, j) \in E$ mit $R \in L_E((i, j))$ und $C \in L_V(j)$, was den Assertionen $\{j : C, (i, j) : R\}$ entspricht. Dementsprechend handelt es sich um eine *generierende Regel*, und die Assertion $i : \exists R.C$ wird im Folgenden *generierende Assertion*, und $\exists R.C$ *generierendes Konzept* genannt.

Das Verhalten der Regel kann wie folgt durch Parametrierung beeinflusst werden:

- Die Regel kann instruiert werden, einen *bereits existierenden* R -Nachfolger j zur Erfüllung des $\exists R.C$ -Konzeptes zu verwenden (für diesen gilt notwendigerweise $C \notin L_V(j)$, da $i : \exists R.C$ andererseits bereits durch die Regel deterministic-expansion als expandiert gekennzeichnet worden wäre). Da mehrere R -Nachfolger existieren können, wird die Regel dann nichtdeterministisch. Es sei erneut betont, dass j nun nicht nur aufgrund von $(i, j) : R \in \mathcal{A}$ ein R -Nachfolger sein kann, sondern dass lediglich $\mathcal{A} \rightsquigarrow (i, j) : R$ gelten muss, wobei die $\rightsquigarrow$ -Relation je nach Parametrierung verschieden generell ist.

Das „Wiederverwenden“ bereits existierender Nachfolger ist zum einen für Anwendungen relevant, die Modell- bzw. Vervollständigungs-Enumeration benötigen, da durch diese Regel eine größere Anzahl von Modellen bzw. Vervollständigungen generiert werden kann (es werden dann nicht nur Baummodelle betrachtet), z.B. für das Konfigurieren. Zum anderen wird dieses Verhalten für den $\mathcal{ALCI}_{RCC}$ -Beweiser benötigt (s.u.). In Kap. 3.1.5, S. 158 und in den MIDELORE-Anforderungen in Kap. 6.1.3, S. 401 hatten wir zudem argumentiert, dass eine solche Regel für ontologiebasierte Anfragebeantwortung in der DISK-Domäne auf abgeschlossenen RCC-ABoxen vorteilhaft sein kann. Die Anforderung wird somit umgesetzt.

- Vor Anwendung der Regel kann eine Blockierungsbedingung `blocked?` ausgewertet werden. Diese wird als Methode `blocked-p` bereitgestellt. Fällt der Test positiv aus, wird kein R -Nachfolger erzeugt. Der Knoten wird dann als blockiert markiert und deaktiviert, wodurch wiederholtes Auswerten der Blockierungsbedingung vermieden wird.

Die $\exists$ -Regel sieht dann wie folgt aus:

$\exists$ -Regel:

wenn

1. $x : \exists R.C \in \mathcal{A}$
2. es gibt kein y , sodass $y : C \in \mathcal{A}$, $\mathcal{A} \rightsquigarrow (x, y) : R$
3. $\neg \text{blocked?}(x, \mathcal{A}, \text{get\_language}(\mathcal{A}))$

dann $\mathcal{A}' := \mathcal{A} \cup \{y : C, (x, y) : R\}$

Ein *blockierter* Knoten wird daher zunächst als erfüllbar angenommen (da seine weitere Expansion zunächst eingespart wird). Sollte sich der *blockierende* Knoten nun als unerfüllbar herausstellen, so ist jedoch auch der blockierte Knoten unerfüllbar. Für DLs *ohne* inverse Rollen wird ein einmal blockierter Knoten immer blockiert bleiben;

für DLs mit inversen Rollen kann eine einmal erfolgte Blockierung hingegen wieder aufgehoben werden (sog. *dynamische Blockierung*).

Knoten können prinzipiell auch durch eine dedizierte *Blockierungsregel* betrachtet und als blockiert markiert werden. Doch ist das Auswerten der Blockierungsbedingung für einen speziellen Knoten vor Anwendung der generierenden Regeln i.d.R. effizienter.

- Das Label $L_V(j)$ des (eventuell nicht neuen) R -Nachfolgers j wird je nach DL und ABox-Typ unterschiedlich ausfallen. Natürlich wird nach Regelanwendung stets $C \in L_V(j)$ gelten.

Für bestimmte DLs kann eine kombinierte $\exists\forall$ -Regel verwendet werden, wie bereits im einfachen $\mathcal{ALC}$ -Beweiser (s. Kap. 6.2.3.4) demonstriert. Die Regel kann daher so parametrisiert bzw. konfiguriert werden, dass zu $L_V(j)$ auch die Menge der anwendbaren Qualifikationen $\{ D \mid \forall S.D \in L_V(i), (\mathcal{A} \cup \{(i,j) : R\}) \rightsquigarrow (i,j) : S \}$ hinzugefügt wird. Hier wird die $\rightsquigarrow$ -Relation i.d.R. so parametrisiert, dass zwar die Effekte von Rollenhierarchien und inversen Rollen berücksichtigt werden, nicht jedoch die Effekte transitiv abgeschlossener Rollen. Stattdessen wird für transitive Rollen $R \in \mathcal{N}_{\mathcal{R}^+}$ auch $\forall R.C \in L_V(j)$, also die in Kap. 6.2.5 besprochene *Weiterpropagierung* sichergestellt.

Zusätzlich können *Bild-Restriktionen* von Rollen berücksichtigt werden: Gilt beispielsweise $\text{range}(R) \dot{\sqsubseteq} E$, so wird $E \in L_V(j)$ sichergestellt.¹⁴

Weitere Konzepte für $L_V(j)$ entstehen durch *Meta-Constraints* in der assoziierten TBox der ABox. Diese müssen für jeden neu erzeugten Knoten hinzugefügt werden.

Da sich das Label $L_V(j)$ des neu erzeugten Knotens j nun je nach Koordinaten auf der $\mathcal{DL}$ - und $\mathcal{K}$ -Achse oder durch entspr. Parametrierung der Regel unterscheidet, liegt es nahe, die Berechnung des Labels des neuen Nachfolgers in eine Methode `compute-some-successor-label` aus der Regel heraus zu verlagern. Zur Anpassung des Verhaltens der Regel ist dann lediglich ein Überschreiben oder Parametrieren dieser Methode erforderlich.

Die Regel feature-expansion: Diese generierende Regel ist für das Expandieren von $\exists F.C$ -Konzepten für funktionale Rollen F ($F \in \mathcal{N}_{\mathcal{F}}$) zuständig. Die Regel ist eng verwandt mit der *some-expansion*-Regel. Alle für diese gemachten Aussagen gelten auch für die *feature-expansion*-Regel.

Im Unterschied zur *some-expansion*-Regel wird hier jedoch nicht ein einzelnes $\exists F.C$ -Konzept expandiert, sondern gleich eine ganze Menge von

¹⁴Ähnlich werden auch Urbild-Restriktionen behandelt. Gilt $\text{domain}(R) \dot{\sqsubseteq} F$, so wird $F \in L_V(i)$ sichergestellt, sobald $\exists R.C$ zu $L_V(i)$ hinzugefügt wird.

$\exists F_i.C_i$ -Konzepten. Dies sei am Beispiel $\{\exists F_1.C, \exists F_2.D\} \subseteq L_V(i)$ mit der Rollenhierarchie $F_1 \sqsubseteq G$, $F_2 \sqsubseteq G$ verdeutlicht, wobei G funktional ist (also $G \in \mathcal{N}_F$). Da G funktional ist, müssen auch die Unterrollen F_1, F_2 funktional sein. Würden die Konzepte $\exists F_1.C, \exists F_2.D$ nun isoliert betrachtet, so würden zwei Nachfolger erzeugt werden: Ein F_1 -Nachfolger j sowie ein F_2 -Nachfolger k . Beide sind jedoch auch G -Nachfolger – die Funktionalität von G wäre also verletzt. Die beiden Nachfolger müssten also verschmolzen werden (dies könnte prinzipiell durch eine at-most-merging-Regel erfolgen). Wir vermeiden dies, indem die sonst ohnehin in einem weiteren Schritt zu verschmelzenden $\exists F_i.C_i$ -Konzepte identifiziert und gleichzeitig expandiert werden.

Die feature-expansion-Regel wird daher zur Expansion von $i : \exists F_1.C$, $i : \exists F_2.D$ lediglich einen $F_1 \sqcap F_2$ -Nachfolger erzeugen, also $(i, j) : F_1 \sqcap F_2$. MIDELORA bietet komplexe Rollen und kann die $(i, j) : F_1 \sqcap F_2$ -Assertion daher darstellen. Die $\leadsto$ -Relation berücksichtigt daher auch konjunktive Rollen-Assertionen.

Analog zur some-expansion-Regel wird auch hier eine Methode `compute-feature-successor-label` zur Berechnung des Knoten-Labels des Nachfolgers aufgerufen.

Die Regel at-least-expansion: Diese deterministische Regel ist für die Expansion von $\geq_n R$ -Konzepten (also den $\mathcal{N}$ -Konzeptkonstruktor) verantwortlich:

$\geq$ -Regel:

wenn

1. $x : \geq_n R \in \mathcal{A}$
2. $|\{y \mid \mathcal{A} \leadsto (x, y) : R, y : \boxed{x : \geq_n R}\} \in \mathcal{A}| < n$
3. $\neg \text{blocked?}(x, \mathcal{A}, \text{get\_language}(\mathcal{A}))$

dann $\mathcal{A}' := \mathcal{A} \cup \{(x, y_i) : R, y_i : \boxed{x : \geq_n R} \mid i \in 1 \dots n\}$

Ebenso wie für die beiden zuvor diskutierten Regeln müssen hier neue R -Nachfolger erzeugt werden, nun jedoch n Stück. Damit ein potentiell nachfolgendes Verschmelzen (durch die at-most-merging-Regel, s.u.) der zur Befriedigung ein und derselben $x : \geq_n R$ -Assertion erzeugten Nachfolger vermieden wird, werden die erzeugten Nachfolger mittels des frischen atomaren Konzeptes $\boxed{x : \geq_n R}$ markiert, sodass erkennbar ist, welche Assertion für die Existenz des Nachfolgers verantwortlich ist (in diesem Falle die Assertion $x : \geq_n R$). Diese speziellen Markierungskonzepte werden nur zur mathematischen Darstellung der Regel verwendet; intern werden zur Verschmelzung vorgesehene Knoten stattdessen anhand der vermerkten Abhängigkeiten daraufhin überprüft, ob sie durch die selbe generierende

Assertion erzeugt wurden (in diesem Falle wird das Verschmelzen nicht zugelassen).

Auch für diese generierende Regel gilt prinzipiell, dass ihre Anwendung blockiert werden kann, und dass sie so parametriert werden kann, dass sie versucht, existierende Nachfolger zur Befriedigung des $\geq_n$ R -Konzeptes zu nutzen.

Zu Berechnung des Knoten-Labels der neuen Nachfolger wird wieder die Methode `compute-some-successor-label` verwendet. Obwohl ein $\geq_n$ R -Konzept keine Qualifikation hat, müssen dennoch Rollen-Bildrestriktionen und Meta-Constraints zum Label des neuen Nachfolgers hinzugefügt, und die transitiven universellen Rollenrestriktionskonzepte weiterpropagiert werden, etc.

Die Regel ist zudem so parametrierbar, dass sie eine Optimierung verwendet, die auf [Haarslev u. Möller 2001b] zurückgeht: Statt n einzelne R -Nachfolger zu erzeugen, wird nur *ein* R -Nachfolger erzeugt, und für die R -Kante bzw. die $(x, y) : R$ -Assertion wird vermerkt, dass sie in Wirklichkeit n äquivalente R -Nachfolger repräsentiert – die Zahl n wird im Folgenden die *Multiplizität der Kante* genannt. Diese Optimierung ist valide, denn würden n verschiedene R -Nachfolger von dieser Regel erzeugt, so hätten sie alle das gleiche Knoten-Label. Eventuell müssen diese „Mehrfach-Nachfolger“ jedoch später aufgrund von $\leq_n$ R -Anzahlrestriktionen wieder aufgesplittet werden (s. nächste Regel).

Die korrekte Umsetzung dieser Idee ist jedoch insbesondere bei inversen Rollen technisch anspruchsvoll, da Kanten nun in zwei Richtungen gelesen werden können. Es muss daher auch eine *inverse Multiplizität* von Tableau-Kanten verwaltet werden.

Die Regel at-most-merging: Diese nichtdeterministische Regel ist für das Auflösen von Konflikten zuständig, die dadurch entstehen, dass ein Knoten i bereits mehr R -Nachfolger hat, als durch $\leq_m R \in L_V(i)$ erlaubt sind:

$\leq$ -Regel:

wenn

1. $x : \leq_m R \in \mathcal{A}$
2. $\mathcal{S} := \{ y \mid \mathcal{A} \rightsquigarrow (x, y) : R \}, \quad |\mathcal{S}| > m$
3. $y_i, y_j \in \mathcal{S}, y_i \neq y_j$
4. $\text{mergable?}(\mathcal{A}, y_i, y_j) = \text{TRUE}$

dann $\mathcal{A}_{y_i, y_j} := \mathcal{A}_{y_i \leftarrow y_j}$

Bei der Berechnung der Anzahl n der R -Nachfolger muss die *Multiplizität* (s.o.) der R -Kanten natürlich berücksichtigt werden.

Die Regel identifiziert also zwei miteinander verschmelzbare Individuen

y_i und y_j (wofür der Test $\text{mergable?}(\mathcal{A}, y_i, y_j) = \text{TRUE}$ verwendet wird), und verschmilzt diese dann miteinander.

Die Funktion bzw. Schablonenmethode mergable? ist standardmäßig wie folgt definiert:

$$\text{mergable?}(\mathcal{A}, y_i, y_j) =_{\text{def}} (\neg \exists (z : \geq_n S) \in \mathcal{A}. \{y_i : \boxed{z : \geq_n S}, y_j : \boxed{z : \geq_n S}\} \subseteq \mathcal{A}) \wedge (\text{UNA?}(\mathcal{A}) \rightarrow \neg(\text{old?}(s_i) \wedge \text{old?}(s_j)))$$

Somit sind y_i und y_j nur dann miteinander verschmelzbar, wenn sie nicht zur Befriedigung derselben $z : \geq_n S$ -Assertion erzeugt wurden. Wird für die ABox die UNA verwendet, so sind „alte“ Knoten ebenfalls nicht miteinander verschmelzbar.

Das Verschmelzen zweier Knoten führt dazu, dass die Knoten-Label vereinigt werden müssen. Um ein destruktives Modifizieren der zu verschmelzenden Knoten-Label zu verhindern, wird ein neuer R -Nachfolger mit dem Vereinigungslabel der verschmolzenen Knoten erzeugt. Die Multiplizitäten der R -Kanten, die zu den verschmolzenen Nachfolgern führen, werden jeweils um Eins reduziert. Ein verschmolzener Nachfolger kann gelöscht oder als gelöscht markiert werden, wenn die Multiplizität seiner R -Kante 0 erreicht hat [Haarslev u. Möller 2001b].

Tatsächlich ist die at-most-merging-Regel also eine generierende Regel in diesem Entwurf. Da die Knoten-Label der verschmolzenen Knoten nicht modifiziert werden, wird hierdurch insbesondere das Wiederherstellen beim Rücksetzen erleichtert.

Beim Verschmelzen von Knoten in DLs mit inversen Rollen können *Zyklen* im Tableau auftreten. Die Blockierungsbedingungen sind so auszuwerten bzw. zu implementieren, dass sie nicht in Endlosschleifen geraten. Diese Problematik besteht für *alte* Individuen bzw. ABox-Individuen im Gegensatz zu Tableau-Individuen nicht, da diese niemals als blockierende Knoten verwendet werden dürfen.

Die Regel initial-abox-saturation: Diese Regel wird zur Implementation von ABox-Erfüllbarkeitsbeweiser benötigt, die eine kombinierte $\exists\forall$ -Regel, bzw. die some-expansion-Regel entsprechend parametrisiert verwenden (s.o.). Diese Regel wird ausschließlich in der *Initialisierungsphase* (prover-init) des Beweisers zur „Vorbereitung“ der ABox verwendet. Die Regel ist nicht anwendbar für DLs mit inversen Rollen.

Die Regel wird durch folgende Beobachtung motiviert: Die Propagierung von C -Qualifikationen aus $\forall R.C$ -Konzepten ist für DLs mit kombinierter $\exists\forall$ -Regel an die Existenz generierender Assertionen gekoppelt. Die anwendbaren C -Qualifikationen werden in der $\exists$ - und der $\geq$ -Regel bereits bei der Berechnung des Labels des neuen Nachfolgers (mittels

compute-some-successor-label) berücksichtigt. Offensichtlich ist dieses Vorgehen nur für *neue Knoten* möglich. Betrachten wir die ABox $\{i : \forall R.C, (i, j) : R, j : \neg C\}$. Diese würde nicht als inkonsistent erkannt werden, da sie keine generierenden Assertionen enthält.

In der Initialisierungsphase des ABox-Erfüllbarkeitsbeweisers wird daher zunächst ein sog. *Prä-Tableau* hergestellt, was die Aufgabe dieser „Makro-Regel“ ist. In gewisser Weise handelt es sich bereits um einen kleinen Beweiser mit der Aufgabe, ein Prä-Tableau zu konstruieren. Die Makro-Regel expandiert die ABox erschöpfend, bis alle verbleibenden unexpandierten Assertionen entweder generierende Assertionen sind, oder aber niemals als expandiert gekennzeichnet werden können (also $i : \forall R.C$ -, $i : \leq_n R$ - und $i : \leq_n R.C$ -Assertion).

Bei der Makro-Regel handelt es sich um eine Regelanwendungskette: Hier werden die Regeln deterministic-expansion und or-expansion verkettet. Die Kette wird in einer Schleife bzw. Rekursion solange durchlaufen, bis keine unexpandierten $\sqcup$ -Konzepte mehr vorhanden sind, und alle Qualifikationen aus $\forall R.C$ -Konzepten propagiert wurden. Wie oben beschrieben, kann die Regel deterministic-expansion so parametrisiert werden, dass sie auch die Propagierung aus $\forall R.C$ -Konzepten heraus durchführt. Da $\forall R.C$ -Konzepte jedoch auch in Disjunktionen „versteckt“ sein können, müssen diese Disjunktionen mittels der or-expansion-Regel aufgelöst werden, denn andererseits würde z.B. die ABox $\{i : (\forall R.C) \sqcup X, i : \neg X, (i, j) : R, j : \neg C\}$ nicht als inkonsistent erkannt.

In der Regelanwendungskette der Makro-Regel startet der negative Ausgang der or-expansion-Regel die Hauptphase des Beweisers (und ruft also prover-main); der Beweis wird dann mit dem konstruierten Prä-Tableau fortgesetzt. Der positive Ausgang startet hingegen eine neue Inkarnation der Regelanwendungskette, also der deterministic-expansion-Regel, wodurch die Schleife implementiert wird. Diese wird erst verlassen, wenn ein Prä-Tableau konstruiert wurde, oder kein konsistentes Prä-Tableau erzeugt werden konnte.

6.2.7.2. Tableau-Regeln für Nicht-Standard-DLs

Während die bisher vorgestellten Regeln als (wenn auch weitgehend parametrierbare) „Standard-Regeln“ bezeichnet werden können, so sind die folgenden Tableau-Regeln als „Spezial-Regeln“ zur Implementierung von Beweisern für Nicht-Standard-DLs (speziell der $\mathcal{ALCI}_{\mathcal{RCC}}$ -Familie) zu bezeichnen:

Die Regel abox-enumeration: Diese nichtdeterministische Regel wird für ABoxen mit disjunktiven Rollen-Assertionen benötigt, z.B. $\{(i, j) : R \sqcup S\}$.

Die Regel enumeriert die atomaren Konfigurationen der ABox, hier also $\{(i, j) : R\}$ und $\{(i, j) : S\}$. Die Regel ist mit der $\sqcup$ -Regel vergleichbar:

$R \sqcup S$ -Regel:

wenn 1. $(x, y) : R_1 \sqcup \dots \sqcup R_n \in \mathcal{A}$
 2. $\mathcal{A} \not\vdash (x, y) : R_i$ für alle $i \in 1 \dots n$
dann $\mathcal{A}_1 := \mathcal{A} \cup \{(x, y) : R_1\}$
 $\vdots$
 $\mathcal{A}_n := \mathcal{A} \cup \{(x, y) : R_n\}$

Tatsächlich kann die Regel *alle* vorhandenen disjunktiven Rollen-Assertionen durch eine einzige Regelanwendung auflösen; mathematisch ist dieser Unterschied irrelevant. Die benötigte Basis-Funktionalität wird von der Klasse Disjunktives-Kanten-Substrat (Disjunctive-Edges-Substrate) in Form der Methoden `get-next-configuration` und `get-all-configurations` ererbt, s. Kap. 5.1.2.6.3, S. 255. Die Regel ist daher nur auf eine ABox anwendbar, die auch Instanz der Klasse Disjunktives-Kanten-Substrat (Disjunctive-Edges-Substrate) ist.

Eine generierte Nachfolgekonfiguration kann durch Auswertung einer überschreibbaren Methode `role-consistent-p` daraufhin überprüft werden, ob sie einer (frei definierbaren) *relationalen Konsistenzbedingung* genügt. Die Regel unterstützt daher ein „Generate and Test“-Vorgehen. Wir erläutern weiter unten, wie und wofür diese Funktionalität genutzt werden kann. Natürlich kann die Regel für spezielle Klassen von ABoxen auch so überschrieben werden, dass bereits beim Generieren lediglich „konsistente“ Nachfolgekonfigurationen erzeugt werden (wodurch der Test dann eingespart werden kann).

Die Regel wird zur Implementation von Beweisern für DLs mit Rollenboxen ($\mathcal{ALCI}_{\mathcal{RA}^\oplus}$, $\mathcal{ALC}_{\mathcal{RA}}$, $\mathcal{ALCI}_{\mathcal{RCC}}$) benötigt. Unabhängig hiervon sind disjunktive Rollen-Assertionen für ABoxen eine sinnvolle Erweiterung.

Wir hatten oben erläutert, dass für Rollen-Assertionen i.d.R. nicht zwischen expandierten und unexpandierten Assertionen unterschieden wird. Wir sehen nun, dass eine solche Unterscheidung zur Entscheidung der Anwendbarkeit der `abox-enumeration`-Regel dennoch nützlich wäre, da nach einmal erfolgter Anwendung und somit Kennzeichnung von $(x, y) : R_1 \sqcup \dots \sqcup R_n \in \mathcal{A}$ als expandiert die nachfolgenden erneuten $\{(x, y) : R_1, \dots, (x, y) : R_n\} \cap \mathcal{A} = \emptyset$ -Tests eingespart werden könnten. Prinzipiell ist es einfach, die Klasse ABox-Kanten-Label (ABox-Edge-Label) um entspr. Attribute (ähnlich wie ABox-Knoten-Label (ABox-Node-Label)) zu erweitern, doch haben wir im Rahmen dieser Arbeit hierauf verzichtet.

Die Regel rolebox-application: Diese deterministische Regel berücksichtigt die in einer RBox definierten Rollen-Axiome. Die drei unterschiedlichen Arten von Axiomen in der RBox führen zu drei Regeln, die von der rolebox-application-Regel erschöpfend angewendet werden:

$S \circ T \sqsubseteq R_1 \sqcup \dots \sqcup R_n$ -Regel:

wenn

1. $S \circ T \sqsubseteq R_1 \sqcup \dots \sqcup R_n \in \text{RBox}(\mathcal{A})$
2. $\mathcal{A} \rightsquigarrow (x, y) : S, \mathcal{A} \rightsquigarrow (y, z) : T$
3. $(x, z) : R_1 \sqcup \dots \sqcup R_n \notin \mathcal{A}$

dann $\mathcal{A}' := \mathcal{A} \cup \{(x, z) : R_1 \sqcup \dots \sqcup R_n\}$

reflexive-Regel:

wenn

1. $\text{reflexive}(R) \in \text{RBox}(\mathcal{A})$
2. $x \in \text{inds}(\mathcal{A})$
3. $\mathcal{A} \not\rightsquigarrow (x, x) : R$

dann $\mathcal{A}' := \mathcal{A} \cup \{(x, x) : R\}$

$^{-1}$ -Regel:

wenn

1. $S = R^{-1} \in \text{RBox}(\mathcal{A})$
2. $\mathcal{A} \rightsquigarrow (x, y) : R$
3. $\mathcal{A} \not\rightsquigarrow (y, x) : S$

dann $\mathcal{A}' := \mathcal{A} \cup \{(y, x) : S\}$

Dabei sei angenommen, dass aus $S = R^{-1} \in \text{RBox}(\mathcal{A})$ auch stets $R = S^{-1} \in \text{RBox}(\mathcal{A})$ folgt. Diese Regeln werden letztlich bereits durch die von der Klasse RBox-Substrat (RBox-Substrate) bereitgestellte beschriebene Funktionalität zur Substrat-Vervollständigung mittels der Methode `complete-substrate` implementiert, s. Kap. 5.1.2.6.4, S. 257; wir geben diese drei Regeln daher nur zur Dokumentation an. Die Regeln sind nur auf ABox-Instanzen einer gemeinsamen Unterklasse der Klassen ABox und RBox-Substrat (RBox-Substrate) (s. Kap. 5.1.2.6.4, S. 257) anwendbar. Wir hatten diese gemeinsame Unterklasse bereits oben als RBox-ABox (RBox-ABox) bezeichnet.

So führt die Anwendung der Regeln auf $\{(i, j) : R, (j, k) : S\}$ mit der RBox $\{\text{reflexive}(U), R \circ S \sqsubseteq T \sqcup V\}$ zu $\{(i, j) : R, (j, k) : S, (i, i) : U, (j, j) : U, (k, k) : U, (i, k) : T \sqcup V\}$. Eine nachfolgende Anwendung von `abox-enumeration` in der Regelanwendungskette kann dann verwendet werden, um die Disjunktionen aufzulösen. Prinzipiell können Disjunktionen sowohl *vor* als auch *nach* Anwendung von `rolebox-application` durch Anwendung von `abox-enumeration` aufgelöst werden.

Die Regel abox-role-completion: Diese deterministische Regel wird spezi-

ell für ABoxen mit JEPD-Eigenschaft vorgesehen und wird somit u.a. für $\mathcal{ALCT}_{RCC}$ -Beweiser benötigt. Die benötigte ABox ist Instanz der Klasse JEPD-ABox (JEPD-ABox), wobei es sich um eine gemeinsame Unterklasse der Klassen RBox-ABox (RBox-ABox) und JEPD-Substrat (JEPD-Substrate) handelt, s. Kap. 5.1.2.6.5, S. 260. Die Regel entspricht der dort beschriebenen complete-substrate-Vervollständigungsmethode:

$\sqcup_R$ -Regel:

wenn 1. $\exists x, y \in V. \forall R \in \text{roles}(\text{RBox}(\mathcal{A})). \mathcal{A} \not\vdash (x, y) : R$
 2. $((x, y) : \sqcup_{R \in \text{roles}(\text{RBox}(\mathcal{A}))} R) \notin \mathcal{A}$
dann $\mathcal{A}' := \mathcal{A} \cup \{(x, y) : \sqcup_{R \in \text{roles}(\text{RBox}(\mathcal{A}))} R\}$

Die möglichen atomaren Konfigurationen der so vervollständigten ABox können dann mittels abox-enumeration enumeriert, und jede Konfiguration kann mittels role-consistent-p auf RCC-Konsistenz im Rahmen des durch die abox-enumeration-Regel unterstützten „Generate & Test“-Vorgehens überprüft werden. So kann die Einhaltung der entspr. RCC-Rollenkompositionsassiome sichergestellt werden. Eine so erzeugte Konfiguration erfüllt also die RBox-Axiome. Dieses Vorgehen ist aufgrund der JEPD-Eigenschaft möglich.

Andererseits können die durch die RBox-Axiome implizierten Rollen-Assertionen auch direkt durch Anwendung der rolebox-application-Regel hinzugefügt werden.

Für einen $\mathcal{ALCT}_{RCC}$ -Beweiser haben wir somit zwei prinzipiell verschiedene Möglichkeiten, um die relationale RCC-Struktur in der ABox zu konstruieren.

6.2.7.3. Blockierungsbedingungen

MIDELORA stellt die folgenden aus der Literatur [Horrocks 2003b] bekannten Blockierungsbedingungen bereit:

Teilmengen-Blockieren (Subset Blocking): Diese Blockierungsbedingung wird für alle DLs mit transitiven Rollen, ohne inverse Rollen, und ohne funktionale Rollen (und somit auch ohne Anzahlrestriktionen) benötigt, also u.a. für $\mathcal{ALC}_{R+}$, $\mathcal{ALCH}_{R+}$, etc.

Hier ist ein Knoten j durch einen (nicht notwendigerweise direkten) Vorgänger-Knoten i blockiert, falls $L_V(j) \subseteq L_V(i)$ gilt. Wir hatten bereits entspr. Beispiele für $\mathcal{ALC}_{R+}$ diskutiert, s. Kap. 2.2.7.3, S. 108.

Es sei angemerkt, dass Teilmengen-Blockieren auch bereits für $\mathcal{ALC}$ mit

allgemeinen TBoxen benötigt wird (sofern GCIs in der TBox vorkommen). Wie beschrieben, kann die TBox durch ein $\mathcal{ALCH}_{\mathcal{R}^+}$ -Konzept internalisiert werden, s. Kap. 2.2.2.4. Damit ist klar, das Teilmengen-Blockieren für $\mathcal{ALC}$ mit allgemeinen TBoxen ausreichend ist. Sobald also eine allgemeine TBox verwendet wird, enthält die von get-language (s. Kap. 6.2.4.2.1, S. 449) zurückgegebene DL mindestens $\mathcal{H}$ und $\mathcal{R}^+$. Es sei angemerkt, dass u.U. auch Bild- und Urbild-Restriktionen für Rollen zu entspr. GCIs führen, und somit ebenfalls ein Blockieren erforderlich machen, selbst wenn die WB keine transitiven Rollen enthält.

Gleichheits-Blockieren (Equal Blocking): Diese Blockierungsbedingung wird für alle DLs mit transitiven Rollen und inversen Rollen, aber ohne funktionale Rollen (und ohne Anzahlrestriktionen) benötigt, also u.a. für $\mathcal{ALCHI}_{\mathcal{R}^+}$.

Hier ist ein Knoten j durch einen (nicht notwendigerweise direkten) Vorgänger-Knoten i blockiert, falls $L_V(j) = L_V(i)$ gilt.

Aufgrund der inversen Rollen kann eine Blockierung eines Knotens j später wieder aufgehoben werden, falls sich (durch weitere Tableau-Expansionen) $L_V(i)$ oder $L_V(j)$ ändern sollten. Man spricht daher vom *dynamischen Blockieren* [Baader u. Sattler 2001]. Wir hatten schon erwähnt, dass für DLs mit inversen Rollen kein Kriterium für die *Stabilität des Labels* bekannt ist.

Paarweises Blockieren (Pairwise Blocking): Diese Blockierungsbedingung wird für alle DLs mit transitiven, inversen, sowie funktionalen Rollen (oder Anzahlrestriktionen) benötigt, also für alle DLs, die $\mathcal{ALCI}_{\mathcal{F}_{\mathcal{R}^+}}$ erweitern. Diese Blockierungsbedingung ist auch noch für $\mathcal{ALCHI}_{\mathcal{F}_{\mathcal{N}_{\mathcal{R}^+}}}$ bzw. $\mathcal{SHIN}$ ausreichend, und somit für alle in dieser Arbeit betrachteten DLs.

Hier ist ein Knoten j durch einen (nicht notwendigerweise direkten) Vorgänger-Knoten i blockiert, falls

1. $L_V(i) = L_V(j)$ und $L_V(i') = L_V(j')$, sowie
2. $L_E((i', i)) = L_E((j', j))$

gilt, wobei i' direkter Vorgänger von i , und j' direkter Vorgänger von j ist.

ABox-Individuen (also „alte“ Individuen) werden niemals blockiert, und können auch nicht als *blockierende Knoten* verwendet werden [Buchheit u. a. 1993a].

Interessanterweise werden für $\mathcal{ALCI}_{\mathcal{RCC1}}$, $\mathcal{ALCI}_{\mathcal{RCC2}}$ und $\mathcal{ALCI}_{\mathcal{RCC3}}$ *keine* Blockierungsbedingungen benötigt [Wessel 2003a]. Dies liegt letztlich daran, dass diese DLs sich auf $\mathcal{FO}_2$ abbilden lassen und somit die Endliche-Modell-Eigenschaft haben. Die Blockierungsbedingung wird aufgrund der stets symmetrischen Rollen-Assertionen nicht benötigt, s. [Wessel 2003a] für Details.

6.2.8. Berücksichtigung von Stand-der-Kunst-Optimierungen

Für MIDeLoRA sollen *Stand-der-Kunst-Optimierungstechniken* [Horrocks 2003b] Anwendung finden, da diese Techniken essentiell zur Erzielung einer akzeptablen Performanz eines DL-Beweisers sind. In den MIDeLoRA-Regeln berücksichtigen wir folgende Optimierungstechniken [Horrocks 2003b]:

Faules Auffalten (Lazy Unfolding) der TBox: Die Axiome allgemeiner, vorbereiteter TBoxen (s. Kap. 6.2.4.3) werden durch sog. *faules* Auffalten während der Tableau-Expansion und somit *erst bei Bedarf* berücksichtigt.

Gilt für einen Knoten i und ein positives atomares Konzept C nun $C \in L_V(i)$, und ist $C \sqsubseteq D$ in der Menge der *auffaltbaren Axiome* der assoziierten TBox enthalten, so wird durch faules Auffalten einfach $L_V(i) := L_V(i) \cup \{D\}$ gesetzt.

Gilt hingegen $\neg C \in L_V(i)$, so wird nur dann $L_V(i) := L_V(i) \cup \{\neg D\}$ gesetzt, wenn $C \sqsubseteq D$ in der Menge der *auffaltbaren Axiome* ist, und daher $\neg C \sqsubseteq \neg D$ gilt.

Das faule Auffalten wird von der Regel *deterministic-expansion* durchgeführt.

Die Menge der *nichtauffaltbaren Axiome* der TBox führt wie in Kap. 6.2.4.3 beschrieben zur Menge der *Meta-Constraints*, die zu jedem Knoten im Tableau hinzugefügt werden müssen. Dies wird von den generierenden Regeln bzw. von einer entspr. Automatik stets sichergestellt.

Aussagenlogische Constraintpropagierung (Boolean Constraint Prop., BCP): Das Hinzufügen von Konzepten zum Label $L_V(i)$ kann als „Seiteneffekt“ andere Konzepte in $L_V(i)$ wahr oder falsch machen. Wird beispielsweise C zu $L_V(i)$ hinzugefügt, so kann eine noch unexpandierte Disjunktion $C \sqcup D \in L_V/unexpanded(i)$ automatisch als *wahr* und damit als *expandiert* gekennzeichnet werden. In der Folge können dann auch Konzepte wie $C \sqcap (C \sqcup D)$ als wahr, und Konzepte wie $\neg(C \sqcup E)$ als falsch erkannt werden und somit einen Widerspruch auslösen. Diese Inferenzen werden durch *aussagenlogische Constraintpropagierung* gezogen. Durch das Hinzufügen eines Konzeptes zum Label eines Knotens wird also i.d.R. eine ganze Kaskade ausgelöst.

Durch BCP werden auch die bereits oben erläuterten *deterministischen Disjunktionen* bzw. deterministischen Disjunkte identifiziert: Wird $\neg C$ zu $L_V(i)$ hinzugefügt, so wird $C \sqcup D \in L_V(i)$ zu einer deterministischen Disjunktion, und D zu einem deterministischen Disjunkt. Dies entspricht folgender Tableau-Regel:

DET $\sqcup$ -Regel:

- wenn**
1. $x : C_1 \sqcup \dots \sqcup C_n \in \mathcal{A}$
 2. $\{x : \neg C_2, \dots, x : \neg C_n\} \subseteq \mathcal{A}$ (o.B.d.A.)
 3. $\{x : C_1, x : \neg C_1\} \cap \mathcal{A} = \emptyset$
- dann** $\mathcal{A}' := \mathcal{A} \cup \{x : C_1\}$

Die $DET \sqcup$ -Regel ist in der Makro-Regel deterministic-expansion integriert.

Semantisches Verzweigen (Semantic Branching): Diese Optimierung betrifft die Behandlung des Nichtdeterminismus' in der $\sqcup$ -Regel.

Gilt $C_1 \sqcup \dots \sqcup C_n \in L_V / \text{unexpanded}(i)$, so wird *syntaktisches Verzweigen* n Nachfolgekonfigurationen mit $L_V / \text{unexpanded}(i) := \{C_j\} \cup L_V / \text{unexpanded}(i)$ für $j \in 1 \dots n$ erzeugen, eine für jedes Disjunkt. In allen Konfigurationen gilt zudem $C_1 \sqcup \dots \sqcup C_n \in L_V / \text{expanded}(i)$. Diese Konfigurationen können in einer *durch eine Heuristik definierten Reihenfolge* untersucht werden.

Semantisches Verzweigen wählt im Unterschied hierzu nicht eine *Disjunktion* und dann ein Disjunkt, sondern eine *boolesche Wertzuweisung* für ein *noch offenes Disjunkt*. Da für eine boolesche Wertzuweisung nur TRUE oder FALSE möglich ist, können hier also nur zwei Nachfolgekonfigurationen erzeugt werden.

Ein Disjunkt C_j ist *offen*, wenn es noch keinen Wahrheitswert zugewiesen bekommen hat. Dies ist der Fall, wenn weder $C_j \in L_V(i)$ noch $\neg C_j \in L_V(i)$ gilt. Semantisches Verzweigen wählt also eine Wertzuweisung für ein offenes Disjunkt. Ist C_j offen in $C_1 \sqcup \dots \sqcup C_n \in L_V / \text{unexpanded}(i)$, so werden die beiden Nachfolgekonfigurationen

1. $L_V / \text{unexpanded}(i) := \{C_j\} \cup L_V / \text{unexpanded}(i)$ und
2. $L_V / \text{unexpanded}(i) := \{\neg C_j\} \cup L_V / \text{unexpanded}(i)$

in einer durch eine Heuristik definierten Reihenfolge untersucht. Ist mehr als ein Disjunkt offen, so können zudem Selektions-Heuristiken verwendet werden. In der Literatur wurden eine Reihe von Heuristiken untersucht [Horrocks 2003b].

Wird 1. gewählt, so kann zudem $L_V / \text{expanded}(i) := \{C_1 \sqcup \dots \sqcup C_n\} \cup L_V / \text{expanded}(i)$ gesetzt werden. Wurde 2. gewählt, und ist nur noch der boolesche Wahrheitswert eines einzelnen Disjunktes C_k in i offen (woraus auch folgt, dass alle nichtoffenen Disjunkte bereits auf FALSE gesetzt wurden, da andererseits die Disjunktion bereits als expandiert gekennzeichnet worden wäre), so muss für i notwendigerweise C_k auf TRUE bzw. $L_V / \text{unexpanded}(i) := \{C_k\} \cup L_V / \text{unexpanded}(i)$ gesetzt werden, damit die Disjunktion erfüllt ist (es muss mindestens ein wahres Disjunkt vorhanden sein). Durch BCP (s.o.) wird dann C_k als *deterministisches Disjunkt* identifiziert, und $C_1 \sqcup \dots \sqcup C_n$ als *deterministische Disjunktion*. Diese ist dann ebenfalls expandiert. Seman-

tisches Verzweigen benötigt daher BCP, um maximal wirken zu können. BCP kann natürlich auch im Falle syntaktischen Verzweigens angewendet werden (z.B. für Disjunktionen mit gemeinsamen Disjunkten).

Die Regel *or-expansion* beherrscht sowohl syntaktisches als auch semantisches Verzweigen.

Dependenzgesteuertes Zurückspringen (Dep. Dir. Backjumping, DDJ): Hierbei handelt es sich um ein optimiertes Suchverfahren, dass die mit rein chronologischem Zurückspringen (bzw. klassischer Tiefensuche) verbundenen Probleme vermeidet (s.u.). Wie bereits in Kap. 6.2.6, S. 459 sowie anhand des einfachen *ALC*-Beweisers in Kap. 6.2.3.4 erläutert, wird durch die Anwendung der Tableau-Regeln im Rahmen der globalen Regelanwendungsstrategie ein Suchbaum aufgespannt, dessen Knoten Such-, Tableau- bzw. ABox-Zustände repräsentieren.

Die Anwendung einer Tableau-Regel auf einen Knoten in diesem Baum erzeugt ein oder mehrere Kinder-Knoten, je nachdem, ob die Regel deterministisch oder nichtdeterministisch ist. Kinder-Knoten mit mehr als einem Nachfolger werden Oder-Knoten oder Verzweigungspunkte genannt. Für Oder-Knoten müssen in der *MIDELORA*-Architektur, wie auf S. 459 erläutert, entweder Funktions-Inkarnationen und somit Bindungsrahmen auf dem Keller zur Bewahrung des lokalen Kontextes (der Bindungen der lokalen Variablen zum Regel-Inkarnationszeitpunkt), oder aber entspr. Attribute in den Befehlsobjekten zur Bewahrung des Kontextes vorgesehen werden, damit der Kontext für ein Wiederaufsetzen gesichert wird und somit nach einem Rücksetzen der Suche in diesen Knoten wieder restauriert werden kann. Diese Verzweigungspunkte bzw. Oder-Knoten *werden durch Verzweigungspunkt-IDs numerisch identifiziert*.

Einfache Suchverfahren arbeiten *rein chronologisch* und leiden am sog. *Thrashing*: Wird während der Tableau-Expansion ein Rücksetzen aufgrund eines erzeugten Widerspruchs notwendig, so wird der (chronologisch) älteste Verzweigungspunkt (der Verzweigungspunkt mit höchster numerischer ID) im Baum aufgesucht, der noch nichtbesuchte Kinder-Knoten hat, und die Suche wird von einem seiner noch nicht besuchten Kind-Knoten aus fort- bzw. wiederaufgesetzt. Diese Wahl stellt jedoch nicht sicher, dass dieser Widerspruch dann nicht unmittelbar wieder erneut auftritt, da u.U. ein jüngerer Knoten im Suchbaum für diesen Widerspruch verantwortlich ist. Dessen Entscheidung wurde jedoch nicht revidiert. So werden u.U. Alternativen im Suchraum exploriert, die nicht dazu beitragen, den Widerspruch zu beseitigen.

Um nun die für den Widerspruch (mit)verantwortlichen Verzweigungspunkte identifizieren zu können, müssen die Dependenzinformationen (kurz: *Dependenzen*, S. 420) der am Widerspruch beteiligten Assertio-

nen ausgewertet werden. Die Abhängigkeiten $\text{deps}(A)$ einer Assertion A geben Auskunft darüber, welche anderen Assertionen für die Assertion ursächlich sind, und in welchem Verzweigungspunkt sie zum Tableau hinzugefügt wurden. Im Falle eines Widerspruchs können so alle (direkt oder indirekt) am Widerspruch beteiligten Assertionen und somit die verantwortlichen Verzweigungspunkte identifiziert werden.

Betrachten wir das widersprüchliche Tableau

$$\begin{aligned} & \{ i : \exists R.((C \sqcup D) \sqcap \exists R.E), i : (\forall R.\forall R.\neg E), \\ & (i, j) : R, j : (C \sqcup D) \sqcap \exists R.E, j : (C \sqcup D), j : D, j : \exists R.E, j : \forall R.\neg E, \\ & (j, k) : R, k : E, k : \neg E \}. \end{aligned}$$

Hier ist lediglich i alt; j und k sind durch Expansion der i -Assertionen entstanden. Es gelten daher die Abhängigkeiten:

- $\text{deps}(i : \exists R.((C \sqcup D) \sqcap \exists R.E)) = \emptyset$
- $\text{deps}(i : (\forall R.\forall R.\neg E)) = \emptyset$
- $\text{deps}((i, j) : R) = \{i : \exists R.((C \sqcup D) \sqcap \exists R.E)\}$
- $\text{deps}(j : (C \sqcup D) \sqcap \exists R.E) = \{i : \exists R.((C \sqcup D) \sqcap \exists R.E)\}$
- $\text{deps}(j : (C \sqcup D)) = \{j : (C \sqcup D) \sqcap \exists R.E\}$
- $\text{deps}(j : D) = \{j : (C \sqcup D)\}$
- $\text{deps}(j : \exists R.E) = \{j : (C \sqcup D) \sqcap \exists R.E\}$
- $\text{deps}(j : \forall R.\neg E) = \{i : (\forall R.\forall R.\neg E), (i, j) : R\}$
- $\text{deps}((j, k) : R) = \{j : \exists R.E\}$
- $\text{deps}(k : E) = \{j : \exists R.E\}$
- $\text{deps}(k : \neg E) = \{j : \forall R.\neg E, (j, k) : R\}$

Zur Auflösung des Widerspruchs $\{k : E, k : \neg E\}$ wird die transitive Hülle der Abhängigkeiten der am Widerspruch beteiligten Assertionen berechnet. Wir erhalten daher $\text{deps}(\{k : E, k : \neg E\}) = \{j : \exists R.E\} \cup \{j : \forall R.\neg E, (j, k) : R\}$, und schließlich $\{j : \exists R.E, j : \forall R.\neg E, (j, k) : R, j : (C \sqcup D) \sqcap \exists R.E, i : \exists R.((C \sqcup D) \sqcap \exists R.E), i : (\forall R.\forall R.\neg E), (i, j) : R\}$. Insbesondere ist $j : (C \sqcup D)$ nicht in dieser Liste. Alle Assertionen sind also deterministisch, m.a.W., es ist keine Möglichkeit vorhanden, den Widerspruch zu beseitigen, da keine Wahlmöglichkeiten in Form von Oder-Knoten im Tableau existieren. Insbesondere wird nun *nicht* ein fruchtloses Zurückspringen zum Verzweigungspunkt für $j : (C \sqcup D)$ und Herstellen von $j : C$ (statt $j : D$) probiert, was bei rein chronologischem Rücksetzen der Fall wäre, da nach der glo-

balen Strategie vor Erzeugung von k die Disjunktion $j : (C \sqcup D)$ expandiert wurde.

Tatsächlich ist es ausreichend, nicht die beteiligten Assertionen mittels $\text{deps}(A)$ zu identifizieren, sondern lediglich die IDs der Verzweigungspunkte, in denen die Assertionen zur ABox hinzugefügt wurden. Die Verzweigungspunkt-IDs einer Assertion A können mittels $\text{choice\_points}(A)$ angefordert werden. Wird eine Assertion A durch den Oder-Knoten mit ID n zum Tableau hinzugefügt, so gilt $n \in \text{choice\_points}(A)$. Zusätzlich enthält $\text{choice\_points}(A)$ aber noch all die IDs der Assertionen, von denen A abhängt (so muss die transitive Hülle bei Auftauchen eines Widerspruchs nicht erst berechnet werden, sondern steht schon zur Verfügung).

Im Falle eines Widerspruchs werden nun die Verzweigungspunkt-IDs der am Widerspruch beteiligten Assertionen vereinigt und absteigend sortiert. Die so erhaltene Liste $CTDS$ (Clash IDs) identifiziert nun alle potentiell am Widerspruch beteiligten Oder-Knoten. Es wird nun zum chronologisch ältesten Oder-Knoten (= Knoten mit größter ID) in dieser Liste zurückgesetzt, der noch nicht explorierte Nachfolgekonfigurationen bzw. Kinder-Knoten hat. Im obigen Beispiel wäre $CTDS = \{0\}$ wodurch sofort gesehen werden kann, dass keine weiteren Alternativen exploriert werden müssen, da für Assertionen A mit $\text{deps}(A) = \emptyset$ auch $\text{choice\_points}(A) = \{0\}$ gilt. Hingegen gilt $\text{choice\_points}(j : D) = \{1\}$, da der Oder-Knoten für $j : C \sqcup D$ die ID 1 hat.

In Kap. 6.2.6, S. 461 waren wir die Erklärung schuldig geblieben, wie eine MIDELORA-Regelinkarnation beim Betreten des *Rückkehr-Einganges* feststellen kann, ob sie für den Widerspruch (mit)verantwortlich ist und somit eine nächste Konfiguration herstellen muss oder nicht. Dies können wir jetzt nachholen.

Wurde, wie soeben beschrieben, die Liste der verantwortlichen Knoten-IDs identifiziert, so wird die Kontrolle durch die Regel-Inkarnationen solange mittels $(\text{return } CTDS)$ an weiter oben im Suchbaum liegende Regel-Inkarnationen durchgereicht, bis eine Regel-Inkarnation ihren Verzweigungspunkt-ID in $CTDS$ findet (damit dieser Test schnell geht, wurde die Liste vorher sortiert). Eine Regel-Inkarnation mit ID m wird die zurückerhaltene Kontrolle mittels $(\text{return } CTDS)$ also an ihre Eltern-Inkarnation und somit an ihren Eltern-Oder-Knoten $m - 1$ zurückgeben, wenn $m \notin CTDS$ gilt. Jede nichtdeterministische Regel-Inkarnation hat ihren lokalen Kontext ja gesichert, der somit auch ihre Verzweigungspunkt-ID enthält. Gilt hingegen $m \in CTDS$, so ist die Inkarnation mitverantwortlich, und sie wird somit aufgefordert, ihre nächste noch nicht generierte Nachfolgekonfiguration herzustellen. Kann die Regel hingegen keine weitere Nachfolgekonfiguration herstellen, so wird sie die Kontrolle

ihrerseits mittels (return *CIDS*) an ihre Eltern-Inkarnation durchreichen. Offensichtlich wird so der chronologisch älteste (mitverantwortliche) Oder-Knoten mit noch nicht explorierten Kinder-Knoten identifiziert.

Konzeptmodelle, Modellverschmelzungstest und Sub-Tableau-Caching: Der Modellverschmelzungstest ist ein unvollständiger Erfüllbarkeitstest für Konjunktionen. Er wird daher als Wächter (s. Kap. 2.2.7.1, S. 102) des Konzepterfüllbarkeitsbeweisers bzw. von *concept-satisfiable-p* für Konjunktionen der Art $C_1 \sqcap \dots \sqcap C_n$ genutzt. Er beschleunigt insbesondere die Taxonomieberechnung, da hier viele Subsumptionstests der Art $C \sqsubseteq D$ auftreten, und $C \not\sqsubseteq D$ gilt gdw. $C \sqcap \neg D$ erfüllbar ist. Die Erfüllbarkeit von $C \sqcap \neg D$ kann evtl. bereits positiv durch einen Modellverschmelzungstest festgestellt werden. Sind nun die *Konzeptmodelle* von C und $\neg D$ verschmelzbar, so ist das Konzept $C \sqcap \neg D$ erfüllbar, und ein teurer Tableau-Unerfüllbarkeitstest kann eingespart werden. Hierfür werden die Konzeptmodelle von C und $\neg D$ benötigt, s. Attribut *concept-models* in Klasse *Konzept* (*Concept*). Ein Konzept kann auch mehrere Konzeptmodelle haben.

Konzeptmodelle werden anhand einer Vervollständigung erstellt; tatsächlich handelt es sich nur um „Ausschnitte“ der Vervollständigung (wir beschreiben Konzeptmodelle später genauer, hier geht es um die Darstellung der prinzipiellen Idee). Wurden noch keine Konzeptmodelle für C und $\neg D$ berechnet, so werden diese durch einen Erfüllbarkeitstest erzeugt. Oftmals sind Konzeptmodelle jedoch bereits vorhanden.

Da der Test für beliebige Konjunktionen anwendbar ist, ist er nicht nur als Wächter von *concept-satisfiable-p* verwendbar, sondern auch dynamisch während der Tableau-Expansion. Wird ein neuer Tableau-Knoten mit $L_{V/unexpanded}(i) = \{C_1 \sqcap \dots \sqcap C_n\}$ bzw. $L_{V/unexpanded}(i) = \{C_1, \dots, C_n\}$ angelegt, und ist dieses initiale Label stabil, so wird ein initiales Konzept $C_1 \sqcap \dots \sqcap C_n$ erzeugt (das Konzept soll nur erzeugt werden, wenn das Label stabil ist). Nun kann der Sub-Tableau-Erfüllbarkeitstest für i eingespart werden, wenn entweder $C_1 \sqcap \dots \sqcap C_n$ bereits als erfüllbar bekannt ist (hierfür ist nicht notwendigerweise ein Konzeptmodell notwendig), oder aber für alle $1 \leq i < j \leq n$ die Konzepte C_i und C_j paarweise verschmelzbar sind. Diese Aufgabe kann von einer (hinter den generierenden Regeln eingeschobenen) *model-merging*-Regel wahrgenommen werden, die somit neu erzeugte Knoten mit stabilem initialen Label u.U. *deaktiviert* und als erfüllbar markiert, sodass die weitere Expansion des neu erzeugten Knotens eingespart wird.

Dual hierzu kann während der Tableau-Expansion nach einem positiv verlaufenem Sub-Tableau-Erfüllbarkeitstest für i ein Konzeptmodell für das initiale Konzept von i erzeugt und abgespeichert werden. Das Mo-

dell darf natürlich erst angelegt werden, wenn die Expansion bzgl. i „beendet“ ist und die Erfüllbarkeit von i sicher feststeht. Prinzipiell ist es dann möglich, den Knoten aus dem Tableau zu löschen. Vor dem Löschen kann dann ein Konzeptmodell erzeugt werden. Diese Aufgabe kann von einer spezialisierten Regel *cache-and-delete* wahrgenommen werden. Diese Regel nimmt also auch die Aufgaben einer (*informierten*) *Speicherverwaltungs-Regel* wahr.

Die Technik des Erzeugens, Abspeicherns und Ausnutzens der Konzeptmodelle initialer $\sqcap$ -Konzepte von Knoten wird *Sub-Tableau-Caching* genannt. In seiner einfachsten Form wird ausschließlich der Erfüllbarkeitsstatus (TRUE oder FALSE) des initialen Konzeptes gespeichert. Offensichtlich gilt:

- Ist bereits $C_1 \sqcap \dots \sqcap C_n$ als unerfüllbar bekannt, so ist auch jede Konjunktion $\sqcap_{i \in X, \{1 \dots n\} \subseteq X} C_i$ unerfüllbar.
- Ist bereits $C_1 \sqcap \dots \sqcap C_n$ als erfüllbar bekannt, so ist auch jede Konjunktion $\sqcap_{i \in X, X \subseteq \{1 \dots n\}} C_i$ erfüllbar.

Die vorgeschlagene Architektur ermöglicht die einfache Integration konzeptmodellgenerierender und konzeptmodellnutzender Regeln in das Beweisverfahren. Hierzu müssen die anderen Regeln nicht verändert werden. Wir meinen, dass diese „Modularität“ für die Architektur spricht.

Prinzipiell ist Sub-Tableau-Caching bisher nur für DLs ohne inverse Rollen verstanden. Das Problem bei DLs mit inversen Rollen ist, dass ein Knotenlabel nicht ohne weiteres als *stabil* erkannt werden kann.

Verwendung von Unerfüllbarkeitscaches: Ein im ASG bzw. der Konzept (Concept)-Instanz für C vermerktes Konzeptmodell bezeugt bereits die *Erfüllbarkeit* dieses Konzeptes. Analog können Konzepte einfach als *unerfüllbar* markiert werden (s. Attribut *satisfiable-p* in der Klasse Konzept (Concept)). Erscheint ein unerfüllbares Konzept auf dem Label eines Knotens, so kann sofort ein Widerspruch und damit Zurücksetzen ausgelöst werden.

Verwendung von Individuumsmodellen: Die soeben diskutierten Konzeptmodelle werden *Konzepten* (Instanzen der Klasse Konzept (Concept)) zugeordnet. Speziell zur Beschleunigung des Instanzentests *individual_instance?*, welcher ja auch für das *Instanzenretrieval* *concept_instances* verwendet wird, können *Modelle für ABox-Individuen* durch unvollständige Tests genutzt werden [Haarslev u. a. 2001a, Haarslev u. Möller 2002]. Wie bekannt, gilt *individual_instance?(i, C, A) = TRUE* gdw. *abox_consistent?(A ∪ {i : ¬C}) = FALSE*.

Ist nun ein sog. *Individuumsmodell* für das Individuum i bekannt, notiert als *ind_models(i)* (s. Attribut *ind_models*, Klasse ABox-Knoten (ABox-Node)), so können wir ebenfalls den Modellverschmelzungstest verwenden.

den, um einen teuren ABox-Erfüllbarkeitstest einzusparen. Hierzu wird das Konzeptmodell von $\neg C$ mit einem beliebigen (oder allen) Individuumsmodell(en) aus $\text{ind\_models}(i)$ auf Verschmelzbarkeit überprüft. Fällt der bzw. mindestens ein Modellverschmelzungstest positiv aus, so ist $\mathcal{A} \cup \{i : \neg C\}$ offensichtlich erfüllbar, und i ist daher eine *offensichtliche Nicht-Instanz* von C (man beachte, dass dies nicht bedeutet, dass i Instanz von $\neg C$ ist). Dieser Test wird durch die Funktion/Methode `obvious_non_instance` durchgeführt. Der Test wird in Kap. 6.3.5 konkretisiert.

Tatsächlich ist das Konzeptmodell von C ein Individuumsmodell von $i : C$. Es wird also die gleiche Datenstruktur zur Modellrepräsentation verwendet.

Um nun hingegen festzustellen, dass i eine *offensichtliche Instanz* von C ist, sind andere Techniken erforderlich, da mit dem Verschmelzungstest auf den bisher diskutierten Modellen lediglich ein unvollständiges Kriterium für *Erfüllbarkeit (einer Konjunktion)* gegeben ist. Nun benötigen wir jedoch ein unvollständiges Kriterium für die *Unerfüllbarkeit (einer Konjunktion)*. Es ist daher nicht mehr ausreichend, lediglich ein aus einer Vervollständigung extrahiertes Konzept- bzw. Individuumsmodell zu betrachten. Stattdessen müssen wir „Modelle“ betrachten, die in *allen* Vervollständigungen gelten, und die wir *Kern-Modelle* nennen [Möller u. a. 2006a;b].

Das Kern-Modell eines Individuums/Knotens i (s. Attribut `core-model`, Klasse ABox-Knoten (ABox-Node)) enthält ausschließlich Modellinformationen, die *in allen Modellen der ABox bzgl. i gelten*. Analog sei das Kern-Modell des Konzeptes C als Kern-Modell des Individuums i der ABox $\{i : C\}$ definiert.

Wir berechnen das Kern-Modell von i anhand *einer* Vervollständigung der ABox, wobei nun jedoch nur die Tableau-Assertionen berücksichtigt werden, für die ersichtlich ist, dass sie in *allen* Vervollständigungen der ABox enthalten sein müssen. Solche Assertionen hatten wir bereits zuvor *deterministisch* genannt. Durch Dependenz-Analyse kann festgestellt werden, ob eine Assertion deterministisch ist oder nicht. Eine deterministische Assertion A hat keine Verzweigungspunkt-IDs, sodass $\text{choice\_points}(A) = \emptyset$ gilt. Dementsprechend ist keine nichtdeterministische Regel (direkt oder indirekt) für ihr Vorhandensein (mit)verantwortlich. Im obigen Beispiel zum DDJ (S. 482) waren alle Assertionen mit Ausnahme von $j : D$ deterministisch.

Das Kernmodell von i wird nun anhand der vorhandenen deterministischen Assertionen, die sich auf i beziehen, gebildet. Bei den deterministischen Assertionen einer ABox handelt es sich daher um eine Teilmenge der *logisch folgerbaren Assertionen der ABox* (bzw. der Theorie der ABox).

Eine *Maximierung der deterministischen Assertionen* könnte nur durch das Berechnen und miteinander Verschneiden mehrerer Vervollständigungen geschehen.¹⁵ Für das Kern-Modell werden wir wieder die gleiche Datenstruktur wie für die anderen Modelle verwenden.

Soll nun geprüft werden, ob i eine Instanz von C ist, so wird das Kern-Modell von i mit dem Kern-Modell von $\neg C$ „vereinigt“. Tritt hierbei ein elementarer Widerspruch auf, so ist i eine offensichtliche Instanz von C . Dieser Test wird durch die Funktion/Methode `obvious_instance` durchgeführt.

Beibehaltung deterministischer Assertionen: Die deterministischen Assertionen eröffnen auch die Möglichkeit, wiederholte und gleichartige ABox- bzw. Tableau-Expansionen zu vermeiden bzw. zu minimieren. Hierbei wird nach einem ABox-Erfüllbarkeitstest die ABox nicht wieder in den originalen Zustand gebracht. Stattdessen werden die deterministischen Assertionen in der Vervollständigung identifiziert und bleiben erhalten. MIDELoRA wird daher kein automatisches Rollback mittels `rollback-to` zum ersten Befehlsobjekt der Befehlshistorie dieser Tableau-Expansion durchführen. Stattdessen werden selektiv die Assertionen A aus der Vervollständigung entfernt, die nichtdeterministisch sind, und für die somit $\text{choice\_points}(A) \neq \emptyset$ gilt. Wird nun ein weiterer ABox-Erfüllbarkeitstest auf dieser ABox durchgeführt, so kann Expansionsarbeit eingespart werden, da viele Assertionen nahezu sofort als expandiert gekennzeichnet werden können. Betrachten wir beispielsweise die ABox $\mathcal{A} = \{i : \exists R.C, j : A \sqcup B\}$, so ist eine Vervollständigung durch $\mathcal{A}' = \mathcal{A} \cup \{(i, k) : R, k : C, j : A\}$ gegeben. Offensichtlich sind alle Assertionen bis auf $j : A$ deterministisch. Diese Assertion wird entfernt, und die ABox bleibt dann in einem Zustand, der $\mathcal{A}'' = \mathcal{A} \cup \{(i, k) : R, k : C\}$ entspricht. Ein erneuter ABox-Erfüllbarkeitstest kann dann die Expansion von $i : \exists R.C$ einsparen, da von der Regel *deterministic-expansion* die Assertion $i : \exists R.C$ sofort als expandiert gekennzeichnet wird. Lediglich $j : A \sqcup B$ wird vor einer erneuten Expansion als unexpandiert gekennzeichnet.

Spezialverfahren wie das *Signaturverfahren* (Haarslev u. Möller 2001b) können in den Regelimplementationen wie beschrieben berücksichtigt werden (s. Regeln *at-least-expansion* und *at-most-merging*, S. 471).

¹⁵Im Gegensatz zu vielen anderen DL-System erlaubt MIDELoRA prinzipiell die Berechnung mehrerer Vervollständigungen.

6.3. Implementation von MiDeLoRA

Der Basis-Entwurf des MiDeLoRA-Baukastens (s. Abb. 6.2, Abb. 6.3) wurde nun sorgfältig motiviert und ausführlich beschrieben, sodass wir uns in diesem Abschnitt der Darstellung der Implementation widmen. Wir beschreiben nun insbesondere die COMMON LISP-Makrosprachen bzw. domänenspezifischen Sprachen zur Definition von Beweisern, Regelanwendungsketten und Tableau-Regeln und beschreiben, wie der MiDeLoRA-Raum implementiert wurde. Die Modellverschmelzungsverfahren sind insbesondere für das Instanzenretrieval essentiell zur Erzielung einer ansprechenden Performanz. Aufgrund der Wichtigkeit dieser Optimierungstechnik stellen wir die Datenstrukturen zur Repräsentation von Konzept- und Individuums-Modellen inkl. der entspr. Implementationen der Modellverschmelzungsalgorithmen vor.

6.3.1. Repräsentation von Mengen

In MiDeLoRA können zur Mengenrepräsentation sowohl Listen als auch AVL-Bäume (mit den sich hieraus ergebenden Vor- und Nachteilen) genutzt werden. Speziell für die Knoten-Label, die u.U. sehr viele Einträge enthalten können, ist es wesentlich, dass der Elementtest performant durchgeführt werden kann. Zur Beschleunigung des Tests können optional zwei ABox-globale Hashtabellen `unexpanded-assertions` und `expanded-assertions` verwendet werden.

6.3.2. Implementation der Befehlshistorie und Befehlsobjekte

Aus Effizienzgründen sind Befehlsobjekte keine CLOS-Instanzen. Stattdessen werden Records verwendet (in COMMON LISP „Structures“ genannt), die eine leichtgewichtige und speichersparende Listenrepräsentation erlauben. Die Listenrepräsentation ist hier auch aus Performanzgründen kein Nachteil, da die Befehlsobjekte nur wenige Attribute haben. COMMON LISP-Structures erlauben auch einfache Vererbung (keine Mehrfach-Vererbung), sodass der Entwurf aus Abb. 6.2 und Abb. 6.3 dennoch direkt umgesetzt werden konnte. Die befehlsobjektspezifischen Methoden, z.B. `undo`, sind als Makros implementiert, sodass hier keine Laufzeiteinbußen durch dynamisches Binden auftreten.

6.3.3. Definition von Tableau-Regeln

Das 5-Port-Modell für Regeln wurde durch Bereitstellung einer COMMON LISP-Makrosprache zur Definition von Regeln implementiert. Natürlich kann Programmierung von Tableau-Regeln hierdurch nicht vermieden werden. Die Ma-

makrosprache vereinfacht das Erstellen von Regeln jedoch wesentlich, da sie problemadäquate Abstraktionen wie das 5-Port-Modell realisiert.

Eine Regel hat einen Namen und ist ein Punkt auf der $\mathcal{DL} \times \mathcal{K}$ -Ebene im MiDELoRA-Raum. Regeln sind konzeptionell CLOS-Multimethoden, hier für die Sprachklasse `dl` (s. Abb. 6.4) und die ABox-Klasse `abox` (entspricht ABox):

```
(defrule or-expansion (dl abox) ...)
```

Um den durch spätes Binden verursachten Laufzeitaufwand einzusparen, wird die Regelimplementation jedoch nicht zur Laufzeit des Beweisers durch spätes Binden, sondern statisch zur *Kompositions- bzw. Übersetzungszeit der Regelanwendungskette* durch spätes Binden ausgewählt. Wir verwenden hier die gleiche Technik wie in Kap. 5.2.2.3.2, S. 297 zur Realisierung des codeschablonenbasierten Anfragenübersetzers.

Die obige Regel definiert letztlich eine Methode `get-code-for-or-expansion ((dl dl) (abox abox) ...)`, welche die Codeschablone für die Regel zurückgibt. Die Codeschablonen werden anhand der spezifizierten Regelanwendungskette (s.u.) anhand von $\mathcal{DL} \times \mathcal{K}$ ausgewählt.

Wir erreichen somit automatisch Vererbung von Regeln entlang der Dimensionen $\mathcal{DL}$ und $\mathcal{K}$ im MiDELoRA-Raum mit der CLOS-Vererbungssemantik. Prinzipiell kann eine Regel auch einfach die Implementation einer beliebigen anderen Regel „anfordern“, sollte die Vererbung anhand der Raumstruktur nicht direkt möglich oder sinnvoll sein:

```
(defrule or-expansion (alc abox) :inherit-from (alchi abox)).
```

Die Makrosprache unterstützt die explizite Definition des negativen und des positiven Ausganges. Hierzu werden die speziellen Markierungssymbole `+insert-negative-code+` und `+insert-positive-code+` in den Regeldefinitionen verwendet. In der Regelanwendungskette (s.u.) wird letztlich spezifiziert, wie die referenzierten Regeln anhand ihrer Ports zusammengeschaltet werden. Da Regeln Codeschablonen sind, hat der Compiler bei der Übersetzung der durch die Regelanwendungskette spezifizierten Regelkomposition letztlich die Freiheit, entweder einen Funktions- bzw. Methodenaufruf zur Fortsetzung der Berechnung durch die nächste Regel zu generieren, oder aber den Code der nachfolgenden Regel textuell einzusetzen („Code Unfolding“). Sollte die Kellerrepräsentation des Suchbaums verwendet werden, so ist dies jedoch nur dann möglich, wenn die nachfolgende Regel keine frische Inkarnation auf dem Keller zur Bewahrung des Kontextes benötigt; die nachfolgende Regel muss daher deterministisch sein. Wir hatten erläutert, dass lediglich die *nichtdeterministischen Regeln* eine neue Inkarnation auf dem Keller benötigen. Alternativ

6.3 Implementation von MIDeLoRA

könnte der lokale Kontext in einem Befehlsobjekt zur späteren Wiederherstellung gesichert werden.

Der Wiedereintrittsausgang einer Regel ist nicht explizit gekennzeichnet, sondern letztlich implizit durch die Rückgabe der Kontrolle aus bzw. nach `+insert-positive-code+` gegeben. Der *Zurücksetz-Ausgang* wird stets durch `return-from prover` ausgelöst (somit ist auch dieser nicht „explizit“ gekennzeichnet).¹⁶ Der von `return-from prover` zurückgelieferte Rückgabewert wird an die Eltern-Inkarnation bzw. den Eltern-Knoten im Suchbaum durchgereicht. Insbesondere müssen hier Verzweigungspunkte-IDs im Falle eines Widerspruchs zurückgegeben werden (s. *CTDS*, S. 483).

Wir geben zur Illustration die Implementation der *or-expansion*-Regel an, die für Instanzen beliebiger ABox- und Sprachklassen anwendbar ist:

```
(defrule or-expansion (dl abox)
  (multiple-value-bind (disjuncts or-concept node)
    (cond ((not *semantic-branching-p*)
      (multiple-value-bind (or-concept node)
        (select-or-concept abox language)
        (when node
          (values
            (remove-if #'(lambda (x)
                          (or (on-tableau-p node x)
                              (on-tableau-p node
                                (get-negated-concept x))))
              (arguments or-concept))
            or-concept
            node))))))
    (t (multiple-value-bind (disjunct or-concept node)
      (select-open-disjunct abox language)
      (when node
        (values
          (list disjunct (get-negated-concept disjunct))
          or-concept
          node))))))

  (if (not (and node disjuncts))
    +insert-negative-code+
    (let ((sat nil)
          (new-choice-point
            (get-new-choice-point))
          (collected-dependencies nil)
```

¹⁶Der lexikalische Block namens `prover` wird automatisch durch `defrule` aufgesetzt. Er erstreckt sich über die gesamte Regel-Implementation.

```

(memorized-action (get-current-action))
(disjuncts (copy-list disjuncts)))
(loop while disjuncts do
  (let* ((disjunct (first disjuncts))
        (last-p (not (cdr disjuncts))))
    (setf disjuncts (cdr disjuncts))
    (let ((added
          (register-as-unexpanded disjunct
            :node node
            :new-choice-point new-choice-point
            :depends-on (list (list node or-concept)))))
      (check-for-clash node added))
    (multiple-value-bind (sub-sat deps)
      +insert-positive-code+
      (when *compute-all-completions-p*
        (rollback-to abox memorized-action))
      (setf sat (or sat sub-sat))
      (when (and (not sat)
                  (not *compute-all-completions-p*))
        (dolist (dep deps)
          (push dep collected-dependencies)))
      (when last-p
        (return-from prover
          (if sat
            t
            (if *compute-all-completions-p*
              nil
              (values
                nil
                (remove new-choice-point
                  collected-dependencies))))))
      (when (not last-p)
        (when (and sat (not *compute-all-completions-p*))
          (return-from prover t))
        (when (not sat)
          (when (not (member new-choice-point deps))
            (return-from prover
              (values
                nil
                (remove new-choice-point
                  collected-dependencies))))
          (when (member new-choice-point deps)
            (rollback-to abox memorized-action))))))))))

```

Die Regel identifiziert zunächst einen Knoten und ein $\sqcup$ -Konzept mit offenen

Disjunkten (Variablen `node`, `or-concept`, `disjuncts`). Diese Selektion wird in Abhängigkeit vom verwendeten Suchverfahren (semantisches oder syntaktisches Verzweigen) durchgeführt; hier wird also die *lokale Strategie* durch die Regel implementiert.

Wird kein entsprechender Knoten in der ABox gefunden, so ist die Regel nicht anwendbar, und der negative Ausgang `+insert-negative-code+` wird genommen; dort wird die Berechnung wie in der Regelanwendungskette beschrieben (s.u.) fortgesetzt.

Andernfalls fordert die Regel eine neue ID für den Verzweigungspunkt mittels `get-new-choice-point` vom MIDeLoRA-Kern an (es handelt sich ja um eine nichtdeterministische Regel), und eine Schleife zur alternativen Wahl jedes möglichen Disjunktes wird gestartet. Das aktuell gewählte Disjunkt wird mittels (`register-as-unexpanded disjunct ...`) zum Label des gewählten Knoten `node` hinzugefügt. Dabei wird mittels `check-for-clash` geprüft, ob nun durch das Hinzufügen ein elementarer Widerspruch erzeugt wurde. Ein solcher Widerspruch wird dann registriert.

Unabhängig davon, ob nun ein Widerspruch hergestellt wurde oder nicht, wird die Berechnung wie durch `+insert-positive-code+` spezifiziert fortgesetzt. Dieser Ausgang wird also immer dann genommen, wenn die Regel *anwendbar* war. Dies deckt sich nicht ganz mit der Beschreibung des *konzeptionellen* positiven Ausganges in Kap. 6.2.6, da der Rücksetz-Ausgang und der positive Ausgang hier „vereint“ sind. Der durch `+insert-positive-code+` spezifizierte Code muss daher zunächst prüfen, ob ein Widerspruch erzeugt wurde oder nicht, und die Ausgänge wieder trennen, wie in Kap. 6.2.6 beschrieben. Wir beschreiben diese Trennung bei der Darstellung der globalen Regelanwendungsstrategie im nächsten Abschnitt. Die Kopplung der beiden Ausgänge ist nicht zwingend.

Kehrt die Kontrolle schließlich im Rahmen des Zurücksetzens aus bzw. nach `+insert-positive-code+` wieder zurück, so werden die aus der Eltern-Inkarnation übermittelten Rückgabewerte mittels (`multiple-value-bind (sub-sat deps) ...`) gebunden. Wurde ein Widerspruch erzeugt, so ist `sub-sat = NIL (FALSE)`, und `deps` enthält dann die Liste der Verzweigungspunkt-IDs der am Widerspruch (mit)beteiligten Assertionen, s. *CIDS* auf S. 483.

Findet die Regel nun in der Liste `deps` ihren eigenen `new-choice-point`, so wird sie damit aufgefordert, das Tableau in den Zustand zurückzurollen, der dem Befehlsobjekt `memorized-action` entspricht, wozu (`rollback-to memorized-action`) gerufen wird. Nach Zurückrollen wird dann das nächste Disjunkt gewählt. In der Implementation wird zudem illustriert, wie die Berechnung *mehrere Vervollständigungen*, also die besprochene *Modell- bzw. Vervollständigings-Enumerationsfunktionalität* realisiert wird (s. `Variable *compute-all-completions*`).

Es sei nun noch illustriert, wie die Definition von „Makro-Regeln“ durch `defrule` unterstützt wird. Eine Makro-Regel nimmt konzeptionell mehrere Aufgaben wahr und nutzt die Dienste von „Komponentenregeln“ zur Lösung von Teilaufgaben. Die Regel `deterministic-expansion` hat, wie in Kap. 6.2.7.1, S. 467 erläutert, mehrere Aufgaben: Sie berücksichtigt die TBox-Axiome durch faules bzw. verzögertes Auffalten, sie expandiert $\sqcap$ -Konzepte, sie propagiert Qualifikationen aus $\forall R.C$ -Konzepten, etc. Eine monolithische Implementierung dieser Regel wäre sehr komplex, schwer verständlich, und daher fehlerträchtig und schwer wartbar. Die Komplexität kann durch Verwendung von Komponentenregeln reduziert werden, indem pro Teilaufgabe eine Komponentenregel definiert wird.

So führt z.B. die Komponentenregel `look-for-ands` die knotenlokale $\sqcap$ -Expansion durch. Die Regel `look-for-ands` wird von der Regel `deterministic-expansion` per `(perform! (look-for-ands :node node :node-changed-p node-changed-p))` gerufen und damit aufgefordert, für den als Argument übergebenen Knoten `node` die $\sqcap$ -Expansion durchzuführen; analog werden die anderen benötigten Komponentenregeln gerufen. Die Parameter sind *bidirektional*. Mittels `node-changed-p` kann die aufrufende Regel daher feststellen, ob ein $\sqcap$ -Konzept von `look-for-ands` expandiert wurde. Komponentenregeln sind keine „echten“ Regeln, da sie lediglich literal als Codeschablonen (Makros) in den Code der aufrufenden Regel eingeschachtelt werden. Dementsprechend starten sie weder eigene Inkarnationen, noch spezifizieren sie entsprechende Ports. Die `look-for-ands`-Regel ist hier für den Punkt `dl × abox` definiert:

```
(defrule look-for-ands (dl abox :args (node node-changed-p))
  (loop-over-node-unexpanded-and-concepts (and-concept node)
    (unless clashes
      (unless (expanded-p node and-concept)
        (announce "Found UNEXPANDED AND ~A : ~A" node and-concept)
        (register-as-expanded and-concept
                              :node node)
        (dolist (arg (arguments and-concept))
          (unless clashes
            (unless (on-tableau-p node arg)
              (setf node-changed-p t)
              (let ((added
                    (register-as-unexpanded
                     arg
                     :node node
                     :depends-on
                     (list (list node and-concept)))))
                (check-for-clash node added))))))))))
```

6.3.4. Beweiser und Regelanwendungsketten

Das MiDeLoRA-Architekturmodell sieht Regelanwendungsketten (globale Regelanwendungsstrategien) und Beweiser als Strukturelemente vor. Wie erwähnt, ist ein MiDeLoRA-Beweiser ein dreiphasiges dreistufiges Kompositum „Beweiser-Regelanwendungskette-Regel“ im MiDeLoRA-Raum $\mathcal{T} \times \mathcal{DL} \times \mathcal{K}$. Zur Definition von MiDeLoRA-Beweisern dient das `define-prover`-Makro.

Ein Beweiser durchläuft drei operationale Phasen: *Initialisierung (Vorbereitung)*, *Hauptphase*, *Endphase (im Falle einer gefundenen Vervollständigung)*. Jede Phase kann eine Regelanwendungskette spezifizieren. Die Syntax von `define-prover` ist daher durch

```
(define-prover ((<Aufgaben-Typ> <Sprach-Klasse> <ABox-Klasse>))
  (:init ...)
  (:main ...)
  (:success ...))
)
```

gegeben, womit ein Beweiser

$(\langle \text{Aufgaben-Typ} \rangle, \langle \text{Sprach-Klasse} \rangle, \langle \text{ABox-Klasse} \rangle) \in \mathcal{T} \times \mathcal{DL} \times \mathcal{K}$

definiert wird.

Die entsprechenden Abschnitte bzw. Phasen führen zur Definition der phasen-spezifischen Methoden `prover-init`, `prover-main`, `prover-success`, die wir bereits in Kap. 6.2.6, S. 465 diskutiert hatten (s. auch entspr. Methoden der Klasse `ABox` in Abb. 6.3). Wird einer der Abschnitte nicht definiert, so sind für die entspr. Phase automatisch die für diese Phase ererbten Methoden anhand der Standard-CLOS-Vererbungssemantik anwendbar. Wie auch für die Regeln, so kann von dieser abgewichen werden, indem beliebige Implementationen bereits existierender anderer Beweiser mittels `:inherit-from` „angefordert“ werden. Zur Illustration geben wir einen *ALCH*-ABox-Erfüllbarkeitsbeweiser für Instanzen der Klasse `abox` an:

```
(define-prover ((abox-sat alch abox))
  (:init
    (perform (initial-abox-saturation)
      (:body
        (start-main))))
  (:main
    (perform (deterministic-expansion)
      (:body
        (if clashes
          (handle-clashes))
        ))))
```

```

(perform (or-expansion)
  (:positive
    (if clashes
      (handle-clashes)
      (restart-main)))
  (:negative
    (perform (some-expansion)
      (:positive
        (if clashes
          (handle-clashes)
          (restart-main)))
      (:negative
        (success))))))
(:success
  (completion-found))

```

Regelanwendungsketten sind keine Strukturelemente erster Ordnung. Sie können lediglich in Beweiserphasen oder Regeldefinitionen verwendet werden. In diesem Beweiser sind zwei Regelanwendungsketten zu sehen: In der `:init`-Phase wird mittels `(perform (initial-abox-saturation) (:body ...))` zunächst eine Anwendung der Regel `initial-abox-saturation` ausgelöst. Wie in Kap. 6.2.7.1, S. 473 erläutert, stellt diese ein Prä-Tableau der ABox her. Dies ist erforderlich, da vom Beweiser eine kombinierte $\exists\forall$ -Regel verwendet wird. Wir hatten erläutert, dass einige Regeln nur einen Ausgang definieren. Dies ist für die Regel `initial-abox-saturation` der Fall, denn unabhängig davon, ob die Regel nun angewendet wurde oder nicht, wird die Berechnung wie unter `:body` spezifiziert mit dem konstruierten Prä-Tableau fortgesetzt. Durch `(start-main)` wird die Hauptphase des Beweisers initiiert.

In der Haupt- bzw. `:main`-Phase des Beweisers ist eine weitere Regelanwendungskette zusammengeschaltet. Die bereits bekannte Strategie (s. funktionaler *ALC*-Beweiser, Kap. 6.2.3.4, S. 440) ist deutlich zu erkennen. Wie erläutert, müssen einige Regeln in dem Falle, dass sie anwendbar waren (wenn also der positive Ausgang genommen wird) eine neue Inkarnation herstellen. Dies ist im Falle der `or-expansion`-Anwendung gut zu sehen: War die Regel anwendbar, so wird mittels `(restart-main)` eine neue Inkarnation der Hauptphase des Beweisers gestartet. Erst wenn alle deterministischen und alle disjunktiven Konzepte expandiert worden sind, wird `(perform (some-expansion) ...)` angewendet. Wurden keine unexpandierten $\exists R.C$ -Konzepte gefunden, so ist die ABox vollständig expandiert, und die Endphase wird mittels `(success)` initiiert.

Die Methode `handle-clashes` löst dependenzgesteuertes Rückspringen (DDJ) aus. Ein Widerspruch wird bereits von der (mit)verantwortlichen Regel in der geteilten Variablen `clashes` registriert. Wie in Kap. 6.3.3 beschrieben, wird die

in `defrule` mittels `+insert-positive-code+` spezifizierte Berechnung immer dann verwendet, wenn die Regel angewendet wurde, unabhängig davon, ob die Nachfolgekongfiguration nun einen Widerspruch enthält oder nicht. Tatsächlich sind daher Rücksetzausgang und *konzeptioneller positiver Ausgang* (s. Kap. 6.2.6) erst durch das Idiom `(if clashes (handle-clashes) <P>)` gegeben. Durch `(handle-clashes)` wird der Rücksetzausgang implementiert, und der positive Ausgang durch `<P>`.

Die End- bzw. `:success`-Phase ruft lediglich `(completion-found)` auf, wobei es sich um eine einfache Benachrichtigungsmethode handelt, die beliebig überschrieben werden kann (beispielsweise kann hier die Vervollständigung kopiert und die Kontrolle dann erneut zurückgegeben werden).

Der Autor vertritt die These, dass die Verständlichkeit der so erzeugten Beweiser hoch ist, und dass diese Beweiser als *ausführbare Spezifikationen* angesehen werden können.

6.3.5. Repräsentation von Modellen für den Modellverschmelzungstest

Wir hatten drei Arten von Modellen diskutiert (s. Kap. 6.2.8):

1. Konzeptmodelle,
2. Individuumsmodelle, und
3. Kernmodelle.

MIDeLoRA verwendet für all diese Modelle die gleiche Datenstruktur, die sog. *Knotenmodelle*.

Ein *Konzeptmodell* wird als Seiteneffekt eines Konzepterfüllbarkeitstests für C angelegt. Dazu wird die anonyme ABox $\{i : C\}$ auf Erfüllbarkeit getestet. Ist das Konzept erfüllbar, so kann anhand der Vervollständigung ein Konzeptmodell für C im Attribut `concept-models` (s. gleichnamige Assoziation in der Klasse `Konzept` (`Concept`)) abgespeichert werden. Dieses Modell ist ein sog. *Knotenmodell*, welches für i anhand der Vervollständigung angelegt wird. Die Methode `make-model-from-node` erzeugt solch ein Knotenmodell.

Auch *Individuumsmodelle* sind letztlich einfache Knotenmodelle. *Kernmodelle* sind ebenfalls Knotenmodelle, doch werden hier nur solche Assertionen aus der Vervollständigung in das Knotenmodell übernommen, die in *allen* Vervollständigungen gelten und somit *logisch folgerbar* sind.

Knotenmodelle sind leichtgewichtige Datenstrukturen (Listen). Das Knotenmodell ist im Wesentlichen eine Kopie der *expandierten und unexpandierten Konzepte* auf dem Label eines als vollständig expandiert gekennzeichneten Kno-

tens.

Ein $\mathcal{ALCH}$ -Knotenmodell ist ein Tupel (bzw. eine Liste) von Mengen (bzw. Listen) der Art

$$(expanded\_concepts, all\_concepts, role\_profiles).$$

Für DLs mit Anzahlrestriktionen ($\geq \mathcal{ALCN}$) wird hingegen

$$(expanded\_concepts, all\_concepts, at\_most\_concepts, role\_profiles)$$

verwendet. Bei den sog. *Rollenprofilen* in $role\_profiles$ handelt es sich um Paare (Listen) der Art (R, n) , $R \in \mathcal{N}_{\mathcal{R}}$, $n \in \mathbb{N}$, wobei $(R, n) \in role\_profiles$ bedeutet, dass der entspr. Knoten n verschiedene R -Nachfolger in der Vervollständigung hatte. MIDELoRA verwendet momentan also lediglich *flache* Modelle, keine *tiefen* Modelle [Haarslev u. Möller 2001b].

Der Test, ob zwei Modelle verschmelzbar sind, kann DL-spezifisch überschrieben werden. Die Methode `models-mergeable-p` mit der Signatur `((language dl) a b)` überprüft die hinreichenden Bedingungen, ob zwei Modelle a und b *offensichtlich nicht miteinander interagieren*.

Zwei $\mathcal{ALC}$ -Knotenmodelle

$$(expanded\_concepts, all\_concepts, role\_profiles)$$

und

$$(expanded\_concepts', all\_concepts', role\_profiles'),$$

sind miteinander verschmelzbar, wenn

1. $expanded\_concepts \cup expanded\_concepts' \cup all\_concepts \cup all\_concepts'$ keinen (syntaktisch offensichtlichen) Widerspruch enthält, und
2. es kein $\forall R.C \in all\_concepts$ gibt, sodass $(R, n) \in role\_profiles'$ (für beliebiges n), und andersrum.

Die entspr. Methode sieht wie folgt aus:

```
(defmethod models-mergeable-p ((language alch) a b)
  (and (not (some
    #'(lambda (concept)
      (let ((neg-concept (negated-concept concept)))
        (find neg-concept
          (node-model-expanded-concepts b))))
      (node-model-expanded-concepts a)))
    (not (some #'(lambda (all)
      (find-if #'(lambda (rp)
        (implies-p (role-profile-role rp)
          (role all)))
          (node-model-role-profiles b)))
      (node-model-alls a))))
```

```
(not (some #'(lambda (all)
  (find-if #'(lambda (rp)
    (implies-p (role-profile-role rp)
      (role all)))
    (node-model-role-profiles a)))
  (node-model-alls b)))))
```

Die Bedingung `(implies-p (role-profile-role rp) (role all))` trägt auch schon einer evtl. vorhandenen $\mathcal{ALCH}$ -Rollenhierarchie Rechnung, da hier $\forall R.C \in all\_concepts$ und $(S, n) \in role\_profiles'$ interagieren, falls $S \sqsubseteq R$ gilt (natürlich gilt auch $R \sqsubseteq R$).

Für $\mathcal{ALCHf}$ muss die Bedingung erneut verschärft werden, da hier auch die Rollenprofile selbst auf (wechselseitige) Interaktionen miteinander überprüft werden müssen, denn $(S, m) \in role\_profiles$ und $(R, m) \in role\_profiles'$ interagieren miteinander, falls sie eine gemeinsame funktionale Oberrolle F haben ($\exists F \in \mathcal{N}_{\mathcal{F}}. S \sqsubseteq F, R \sqsubseteq F$):

```
(defmethod models-mergeable-p ((language alchf) a b)
  (and (models-mergeable-p +alch+ a b)

    (not (some #'(lambda (rp-a)
      (let ((r-a (role-profile-role rp-a)))
        (find-if
          #'(lambda (rp-b)
            (let ((r-b (role-profile-role rp-b)))
              (has-common-parent-feature r-a r-b)))
            (node-model-role-profiles b))))
      (node-model-role-profiles a)))))
```

Entsprechende Verschärfungen sind für zusätzliche Konstruktoren ($\mathcal{N}$, $\mathcal{Q}$) erforderlich. Diese Zusatzbedingungen wollen wir hier nicht weiter erläutern.

Kernmodelle werden hingegen, wie erläutert, für einen (unvollständigen) *offensichtlichen Unerfüllbarkeitstest* benötigt. Dementsprechend unterscheidet sich auch der Modellverschmelzungstest für Kernmodelle vom soeben diskutierten Modellverschmelzungstest. Beispielsweise wird für zwei $\mathcal{ALC}$ -Kernmodelle geprüft, ob $expanded\_concepts \cup expanded\_concepts' \cup all\_concepts \cup all\_concepts'$ einen Widerspruch enthält. Nur in diesem Falle fällt für Kernmodelle der Verschmelzbarkeitstest negativ aus; das negative Ergebnis wird dann als offensichtliche Unerfüllbarkeit interpretiert. Der Test ist also eine Aufweichung bzw. Abschwächung des soeben diskutierten Verschmelzbarkeitstests (liefert öfter TRUE, da das Ergebnis FALSE nun als *offensichtlich unerfüllbar* und nicht als *nicht offensichtlich erfüllbar* interpretiert wird).

6.4. Vorstellung einiger MIDeLoRA-Instantiierungen

Die Flexibilität und Generizität von MIDeLoRA illustrieren wir durch Vorstellung von „Proof of Concept“-Implementationen für eine Reihe von Beweisern. Wir stellen daher ausgewählte Punkte

$$(T, DL, K) \in \mathcal{T} \times \mathcal{DL} \times \mathcal{K}$$

im MIDeLoRA-Raum vor, um Flexibilität und Variabilität bzgl. der Achsen $\mathcal{T}$, $\mathcal{DL}$ und $\mathcal{K}$ zu demonstrieren:

1. Dazu gehören eine Reihe von ABox-Erfüllbarkeitsbeweisern für Standard-DLs, von $\mathcal{ALCH}$ bis $\mathcal{ALCHfN}_{\mathcal{R}^+}$, von $\mathcal{ALCHI}$ bis zu $\mathcal{ALCHIfN}_{\mathcal{R}^+}$. Diese Beweiser entsprechen daher den Punkten $\mathcal{T} \times \mathcal{DL} \times \mathcal{K} = \{\text{abox\_consistent?}\} \times \{\mathcal{ALCH}, \dots, \mathcal{ALCHfN}_{\mathcal{R}^+}, \mathcal{ALCHI}, \dots, \mathcal{ALCHIfN}_{\mathcal{R}^+}\} \times \{\text{ABox}\}$. Hier wird also die $\mathcal{DL}$ -Koordinate variiert.
Die zentrale Rolle des ABox-Erfüllbarkeitsbeweisers hatten wir bereits diskutiert; nahezu alle anderen Beweiser werden seine Dienste direkt oder indirekt nutzen müssen. Der `abox\_consistent?`-Beweiser ist daher zentral für die entspr. $\mathcal{DL} \times \mathcal{K}$ -Ebene bzw. -Familie.
2. Wir geben einen Beweiser für einen anderen Aufgabentyp als $T = \text{abox\_consistent?}$ an – einen (generischen) Instanzenretrieval-Beweiser, $T = \text{concept\_instances}$. Der Beweiser wird zudem inkrementell arbeiten, damit die BAA Generator-ABFen inkrementell arbeiten können, s. Kap. 6.1.7, S. 411.
3. Zudem werden für eine MIDeLoRA-BAA auch noch Beweiser für $\mathcal{T} = \{\text{role\_pairs}, \text{individual\_role\_successors}, \text{individual\_role\_predecessors}\}$ benötigt. Die zur Bereitstellung der BAA erforderlichen Implementationstätigkeiten im MIDeLoRA-Rahmenwerk werden dargestellt. Hier wird also die $\mathcal{T}$ -Koordinate variiert.
4. Zur Illustration der Variabilität bzgl. der $\mathcal{K}$ - und $\mathcal{DL}$ -Achse geben wir ABox-Erfüllbarkeitsbeweiser für die Spezial-DLs $\mathcal{ALCI}_{\mathcal{RA}^\ominus}$, $\mathcal{ALCI}_{\mathcal{RA}}$ und $\mathcal{ALCI}_{\mathcal{RCC}}$ an. Diese Beweiser benötigen spezialisierte ABoxen, nämlich Instanzen der Klasse `RBox-ABox` (`RBox-ABox`) bzw. `JEPD-ABox` (`JEPD-ABox`), s. Kap. 6.2.7.2, S. 474. Wir hatten erwähnt, dass $\mathcal{ALCI}_{\mathcal{RCC}1}$ bis $\mathcal{ALCI}_{\mathcal{RCC}3}$ entscheidbar sind, und es existieren auch entscheidbare Fragmente von $\mathcal{ALCI}_{\mathcal{RA}^\ominus}$ - und $\mathcal{ALCI}_{\mathcal{RA}}$.
5. Ebenfalls zur Illustration der Variabilität bzgl. der $\mathcal{K}$ -Achse beschreiben wir die Anpassungen, die an den Standard-Beweisern erforderlich werden, wenn räumliche oder persistente ABoxen verwendet werden sollen. Wir dokumentieren somit die Realisierung virtueller Assertionen im MIDeLoRA-Rahmenwerk.

6.4.1. ABox-Erfüllbarkeitsbeweiser für Standard-DLs

Der vom MiDeLoRA-Baukasten in Form von Standard-Tableau-Regeln (s. Kap. 6.2.7.1, S. 467) und syntaktischer Basis-Funktionalität (s. Kap. 6.2.4, S. 447) bereitgestellte Funktionsumfang ist prinzipiell ausreichend zur Implementation von Beweisern für DLs von $\mathcal{ALCH}$ bis zu $\mathcal{ALCHIfN}_{\mathcal{R}^+}$. Zusätzlich werden generelle TBoxen, Bild- und Urbildrestriktionen für Rollen, Rollenkompositionsaxiome, reflexive Rollen, sowie disjunktive und konjunktive Rollen prinzipiell unterstützt. Wir beschreiben nun, wie diese vorhandene umfangreiche Basis-Funktionalität zur Komposition von Beweisern für Standard-DLs genutzt werden kann. Wir geben daher ABox-Erfüllbarkeitsbeweiser für $\mathcal{ALCH}$ bis $\mathcal{ALCHIfN}_{\mathcal{R}^+}$ an.

Die Beweiser für DLs mit inversen Rollen unterscheiden sich in wesentlichen Punkten von den Beweisern für DLs ohne inverse Rollen, s. S. 473: Beweiser für DLs ohne inverse Rollen verwenden eine kombinierte $\exists\forall$ -Regel. Zudem sind Optimierungen wie das Sub-Tableau-Caching anwendbar. Aufgrund der kombinierten $\exists\forall$ -Regel muss in der Vorbereitungsphase die Regel *initial-abox-saturation* angewendet werden. Diese Unterschiede motivieren die Schaffung zweier „getrennter“ Stränge von Beweisern, einen für die $\mathcal{ALCH}$ -Familie, und einen für die $\mathcal{ALCHI}$ -Familie.

Die im Folgenden vorgestellten Beweiser verwenden die Regeln, die wir in Kap. 6.2.7, S. 466 entworfen haben. Die Regeln sind wie dort erläutert implementiert.

Wir beschreiben nun zunächst den $\mathcal{ALCH}$ -Strang, und dann den $\mathcal{ALCHI}$ -Strang.

6.4.1.1. Der $\mathcal{ALCH}$ -Strang

Der $\mathcal{ALC}$ -Strang ist durch Beweiser für $\mathcal{ALCH}$, $\mathcal{ALCHf}$, $\mathcal{ALCHf}_{\mathcal{R}^+}$, und $\mathcal{ALCHfN}_{\mathcal{R}^+}$ gegeben. Die Wahl dieser Punkte auf der $\mathcal{DL}$ -Achse ist durch die für diese DLs anwendbaren Optimierungen motiviert. Natürlich wäre bereits der $\mathcal{ALCHfN}_{\mathcal{R}^+}$ -Beweiser ausreichend, um alle diese Sprachen abzudecken. Die „kleineren“ Beweiser sind jedoch zum einen verständlicher und somit leichter wart- und anpassbar. Zum anderen können die eingeschränkten Beweiser auch effizienter arbeiten, da sie weniger Fälle abprüfen müssen; beispielsweise muss der $\mathcal{ALCH}$ -Beweiser keinen Blockierungstest durchführen.¹⁷ Wir haben somit erläutert, wann eine DL hier als „relevant“ anzusehen ist und somit zu einer entspr. Sprachklasse in Abb. 6.4 führt.

¹⁷Der verwendete Beweiser wird letztlich anhand von *get-language*, s. Kap. 6.2.4.2.1, S. 449 ausgewählt; hierbei werden auch die TBox-Axiome berücksichtigt. Enthält eine $\mathcal{ALCH}$ -TBox Meta Constraints, so wird *get-language* mindestens $\mathcal{ALCH}_{\mathcal{R}^+}$ zurückgeben.

6.4.1.1.1. *ALCH* Der *ALCH*-Beweiser unterscheidet sich von dem in Kap. 6.3.4, S. 494 diskutierten Beweiser nur in wenigen Details.

Zum einen kann auf die Herstellung des Prä-Tableaus mittels *initial-abox-saturation* verzichtet werden, sollte die ABox nur einen Knoten enthalten; hierdurch werden Konzepterfüllbarkeitstests leicht beschleunigt.

Zum anderen wurde die Regelanwendungskette in der Hauptphase um die Regelanwendungen der Regeln *cache-and-delete* und *model-merging* (s. auch Kap. 6.2.8, S. 484) erweitert:

```
(define-prover ((abox-sat alch abox))
  (:init
    (if (zerop (1- (no-of-nodes abox)))
      (progn
        (loop-over-abox-nodes (node abox)
          (register-label-is-stable node))
        (start-main))
      (perform (initial-abox-saturation)
        (:body (start-main))))))
  (:main
    (perform (deterministic-expansion)
      (:body
        (if clashes
          (handle-clashes)
          (perform (or-expansion)
            (:positive
              (if clashes
                (handle-clashes)
                (restart-main)))
            (:negative
              (perform (cache-and-delete)
                (:body
                  (perform (some-expansion)
                    (:positive
                      (if clashes
                        (handle-clashes)
                        (perform (model-merging :node new-node)
                          (:body
                            (continue-main))))))
                    (:negative
                      (success))))))))))
      (:success
        (completion-found)))
```

Die Regel *cache-and-delete* wird also nach vollständig erfolgter aussagenlo-

6.4 Vorstellung einiger MIDeLoRA-Instantiierungen

gischer Expansion angewendet, d.h., in diesem Zustand gibt es in der ABox keine offenen Disjunktionen mehr. Diese Regel hat, wie bereits in Kap. 6.2.8, S. 484 diskutiert, zwei Aufgaben:

1. Das Löschen von Knoten aus dem Tableau.
2. Das Sub-Tableau-Caching.

Ein Knoten kann dann gelöscht und zuvor sein Knotenmodell angelegt und abgespeichert werden, wenn er als *erfüllbar* bekannt und entsprechend markiert ist. Dies ist der Fall, wenn sein Label keine unexpandierten $\exists R.C$ -Konzepte mehr enthält, und der Knoten entweder

- keine Nachfolger hat, oder aber
- alle seine Nachfolger *erfüllbar* sind (im Sinne dieser rekursiven Definition).

Das Knotenmodell wird dann als Konzeptmodell für das initiale $\sqcap$ -Konzept des Knotens abgespeichert, allerdings nur dann, wenn der Knoten nicht bereits als erfüllbar bekannt war und daher bereits seine Sub-Tableau-Expansion eingespart wurde. Dies ist möglich, da *some-expansion* eine kombinierte $\exists\forall$ -Regel verwendet, und das Label von *new-node* daher stabil ist. Das initiale $\sqcap$ -Konzept eines Knotens mit stabilem Label wird beim Erzeugen des Knotens automatisch angelegt. Für dieses wird sofort (wie beschrieben) Erfüllbarkeit anhand der Caches bzw. durch paarweise Modellverschmelzungstest der Konjunkte des initialen $\sqcap$ -Konzeptes überprüft. Wurde der Knoten so als erfüllbar nachgewiesen, so wird er von der *model-merging*-Regel sofort deaktiviert und mit dem Status „erfüllbar“ versehen.

Mittels (*restart-main*) wird, wie bereits in Kap. 6.3.4, S. 494 beschrieben, die unter *:main* spezifizierte Regelanwendungskette erneut durchlaufen. Dies entspricht bei Verwendung der Kellerrepräsentation des Suchbaums einem Aufruf der *prover-main*-Methode. Es wird also eine neue Inkarnation bzw. ein neuer Knoten im Suchbaum erzeugt, und somit insbesondere der lokale Kontext der *or-expansion*-Regel bewahrt. Durch (*continue-main*) wird ebenfalls die im *:main*-Abschnitt spezifizierte Regelanwendungskette erneut durchlaufen, doch wird der lokale Kontext *nicht* bewahrt. Wird der Keller zur Suchbaum-Repräsentation verwendet, so können hier Bindungsrahmen bzw. rekursive Aufrufe von *prover-main* eingespart werden.

6.4.1.1.2. *ALCHf* *ALCHf* bietet funktionale Rollen. Wir geben hier einen Beweis an, der die UNA verwendet. In der Initialisierungsphase (*:init*) wird auf das Vorhandensein von Konflikten der Art $\mathcal{A} \rightsquigarrow \{(i, j) : F, (i, k) : F\}$ für eine funktionale Rolle F ($F \in \mathcal{N}_{\mathcal{F}}$) geprüft. Offensichtlich handelt es sich hier um eine Verletzung der Funktionalität von F ; ein solcher *Feature Clash* kann in einer UNA-ABox nicht behoben werden. Die ABox ist daher unerfüllbar, wenn

ein solcher Widerspruch gefunden wird. Auf die Herstellung des Prä-Tableaus sowie den Feature Clash-Test wird verzichtet, sollte die ABox nur einen Knoten enthalten. Die hier implementierte globale Regelanwendungsstrategie wendet zunächst feature-expansion, und dann some-expansion an. Nach jeder Anwendung einer dieser Regeln wird wie im $\mathcal{ALCH}$ -Beweiser der generierte Nachfolger durch die model-merging-Regel auf Erfüllbarkeit getestet. Die feature-expansion-Regel ist für die Expansion von $\exists F.C$ -Konzepten, $F \in \mathcal{N}_{\mathcal{F}}$ verantwortlich und wurde in Kap. 6.2.7.1, S. 470 entworfen und erläutert.

```
(define-prover ((abox-sat alchf abox))
  (:init
    (if (zerop (1- (no-of-nodes abox)))
        (progn
          (loop-over-abox-nodes (node abox)
            (register-label-is-stable node))
          (start-main))
        (progn

          (announce "Looking for initial feature clashes")
          (let ((features (and *cur-tbox*
                               (get-all-features *cur-tbox*))))
            (loop-over-abox-nodes (node abox)
              (when (some
                    #'(lambda (f)
                        (when (cdr (get-role-successors node f))
                          (announce "Found feature clash ~A : ~A"
                                    node f)
                          t)))
                    features)
                (return-from prover-init nil))))

          (perform (initial-abox-saturation)
            (:body (start-main)))))))

  (:main
    (perform (deterministic-expansion)
      (:body
        (if clashes
            (handle-clashes)
            (perform (or-expansion)
              (:positive
                (if clashes
                    (handle-clashes)
                    (restart-main)))))))))
```

6.4 Vorstellung einiger MIDeLoRA-Instantiierungen

```
(:negative
  (perform (cache-and-delete)
    (:body
      (perform (feature-expansion)
        (:positive
          (if clashes
            (handle-clashes)
            (perform (model-merging :node new-node)
              (:body
                (continue-main))))))
      (:negative
        (perform (some-expansion)
          (:positive
            (if clashes
              (handle-clashes)
              (perform (model-merging :node new-node)
                (:body
                  (continue-main))))))
          (:negative
            (success))))))))))
```

Die `:success`-Phase des Beweisers wurde vom $\mathcal{ALCH}$ -Beweiser ererbt ($\mathcal{ALCHF}$ ist Unterklasse von $\mathcal{ALCH}$, s. Abb. 6.4).

6.4.1.1.3. $\mathcal{ALCH}_{f_{\mathcal{R}^+}}$ Die Initialisierungs- und Endphase können ererbt werden. Da nun transitiv abgeschlossene Rollen ($\mathcal{R}^+$) geboten werden, wird in der Hauptphase eine Blockierungsregel benötigt: `block-nodes`. Ein neu generierter Knoten `new-node` wird von `block-nodes` auf Blockierung direkt nach Generierung getestet. Hier ist Teilmengen-Blockieren ausreichend. Fällt der Test positiv aus, so deaktiviert `block-nodes` den Knoten. Da der Erfüllbarkeitsstatus eines blockierten Knotens direkt mit dem Erfüllbarkeitsstatus des blockierenden Knotens gekoppelt ist, dürfen Knoten, die *blockierte Nachfolger haben oder selbst blockiert sind*, nicht ohne weiteres von der Regel `cache-and-delete` als erfüllbar betrachtet werden. Eine Verfeinerung der `cache-and-delete`-Regel sorgt dafür, dass solche Knoten nicht als erfüllbar markiert werden können; eine weitergehende Behandlung ist in [Haarslev u. Möller 2000, Horrocks 2003b] beschrieben, die wir hier jedoch nicht implementiert haben. Prinzipiell ist die dort beschriebene Zusatzlogik ohne Weiteres in die Regeln integrierbar; die Definition des Beweisers bleibt also unverändert:


```
(define-prover ((abox-sat alchf-rplus abox))
  (:main
    (perform (deterministic-expansion)
      (:body
        (if clashes
          (handle-clashes)
          (perform (or-expansion)
            (:positive
              (if clashes
                (handle-clashes)
                (restart-main)))
            (:negative
              (perform (cache-and-delete)
                (:body
                  (perform (feature-expansion)
                    (:positive
                      (if clashes
                        (handle-clashes)
                        (perform (block-nodes :blocked new-node)
                          (:body
                            (perform (model-merging :node new-node)
                              (:body
                                (continue-main))))))))
                    (:negative
                      (perform (some-expansion)
                        (:positive
                          (if clashes
                            (handle-clashes)
                            (perform (block-nodes :blocked new-node)
                              (:body
                                (perform (model-merging :node new-node)
                                  (:body
                                    (continue-main))))))))
                        (:negative
                          (success))))))))))))))
```

6.4.1.1.4. $\mathcal{ALCHfN}_{\mathcal{R}^+}$ Die Initialisierungs- und Endphase können wieder erbt werden. Die Unterstützung von $\mathcal{N}$ erfordert nun zusätzlich die Einbindung der beiden Regeln *at-least-expansion* (s. S. 471) und *at-most-merging* (s. S. 472) in die Regelanwendungskette. Die globale Regelanwendungsstrategie aus [Baader u. Sattler 2001] fordert, dass *generierende Regeln* (also die Regeln *some-expansion*, *feature-expansion*, *at-least-expansion*) nur dann angewendet werden, wenn keine nichtgenerierende Regel (und insbesondere nicht die Regel *at-most-merging*) anwendbar ist. Der angegebene Bewei-

ser realisiert diese Strategie:

```
(define-prover ((abox-sat alchfn-rplus abox))
  (:main
    (perform (deterministic-expansion)
      (:body
        (if clashes
          (handle-clashes)
          (perform (or-expansion)
            (:positive
              (if clashes
                (handle-clashes)
                (restart-main)))
            (:negative
              (perform (at-most-merging)
                (:positive
                  (restart-main))
                (:negative
                  (perform (cache-and-delete)
                    (:body
                      (perform (feature-expansion)
                        (:positive
                          (if clashes
                            (handle-clashes)
                            (perform (block-nodes :blocked new-node)
                              (:body
                                (perform (model-merging)
                                  (:body
                                    (continue-main))))))))
                        (:negative
                          (perform (some-expansion)
                            (:positive
                              (if clashes
                                (handle-clashes)
                                (perform (block-nodes :blocked new-node)
                                  (:body
                                    (perform (model-merging)
                                      (:body
                                        (continue-main))))))))
                            (:negative
                              (perform (at-least-expansion)
                                (:positive
                                  (if clashes
                                    (handle-clashes)
                                    (perform (block-nodes :blocked new-node)
                                      (:body
```

```
(perform (model-merging)
  (:body
    (continue-main))))))
(:negative
  (success)))))))))
```

6.4.1.2. *ALCHI*-Strang

Der *ALCHI*-Strang ist, wie bereits erläutert, im Wesentlichen unoptimiert. Die Performanz dieser Beweiser wird somit wesentlich schlechter ausfallen, als im soeben diskutierten *ALCH*-Strang.

6.4.1.2.1. *ALCHI* Da nun keine kombinierte $\exists\forall$ -Regel mehr verwendet werden kann, kann auf eine Anwendung von *initial-abox-saturation* in der *:init*-Phase des Beweisers verzichtet werden, s. Kap. 6.2.7.1, S. 473. Da zudem keine Optimierungsregeln anwendbar sind, ist der resultierende Beweiser einfacher als der entspr. *ALCH*-Beweiser:

```
(define-prover ((abox-sat alchi abox))
  (:init
    (start-main))
  (:main
    (perform (deterministic-expansion)
      (:body
        (if clashes
          (handle-clashes)
          (perform (or-expansion)
            (:positive
              (if clashes
                (handle-clashes)
                (restart-main)))
            (:negative
              (perform (some-expansion)
                (:positive
                  (if clashes
                    (handle-clashes)
                    (restart-main)))
                (:negative
                  (success))))))))))
  (:success
    (completion-found)))
```

Wir verzichten auf die Darstellung der Beweiser für *ALCHIf* und *ALCHIf_{R+}* und fahren mit dem *ALCHIf_{N_{R+}}*-Beweiser fort.

6.4.1.2.2. $\mathcal{ALCHIf}\mathcal{N}_{\mathcal{R}^+}$ Wie auch im $\mathcal{ALCHf}$ -Beweiser muss die Eingabe-ABox zunächst in der `:init`-Phase auf das Vorhandensein von *Feature Clashes* überprüft werden. Im Gegensatz zur Initialisierungsphase des $\mathcal{ALCHf}$ -Beweisers können Knoten nun jedoch nicht mehr mittels `register-label-is-stable` als stabil markiert werden, und die `initial-abox-saturation`-Anwendung ist wieder überflüssig. In der Hauptphase kann die Überprüfung der Blockierungsbedingung nun zudem nicht mehr ausschließlich auf neu erzeugte Nachfolger begrenzt werden. Die Blockierungsbedingung muss für alle Knoten mit noch unexpandierten generierenden Konzepten ausgewertet werden, und Blockierung entspr. stattfinden. Die Regel `block-nodes` wird daher nach erfolgter `at-most-merging`-Anwendung angewendet; sie betrachtet dann alle relevanten Knoten. Eine Fokussierung von `block-nodes` auf neu erzeugte Knoten durch Angabe der Argumente `:node new-node` wie im $\mathcal{ALCH}$ -Strang ist nun nicht mehr möglich. Die Regel sorgt zudem dafür, dass das Blockieren *dynamisch* erfolgt; Blockierungen werden automatisch (vom Kern) wieder aufgehoben, sollte sich das Label eines bereits blockierten oder blockierenden Knotens ändern (die Methode `register-as-unexpanded` stellt dies per Seiteneffekt sicher). Zudem wird nun das *paarweise Blockieren* benötigt. All diese Details sind in den Regeldefinition sauber abstrahiert:

```
(define-prover ((abox-sat alchifn-rplus abox))
  (:init ...) ; s. Anmerkung im Text
  (:main
    (perform (deterministic-expansion)
      (:body
        (if clashes
          (handle-clashes)
          (perform (or-expansion)
            (:positive
              (if clashes
                (handle-clashes)
                (restart-main)))
            (:negative
              (perform (at-most-merging)
                (:positive
                  (restart-main))
                (:negative
                  (perform (block-nodes)
                    (:body
                      (perform (feature-expansion)
                        (:positive
                          (if clashes
                            (handle-clashes)
                            (continue-main))))))))))))))
```

```
(:negative
  (perform (some-expansion)
    (:positive
      (if clashes
        (handle-clashes)
        (continue-main))))
  (:negative
    (perform (at-least-expansion)
      (:positive
        (if clashes
          (handle-clashes)
          (continue-main))))
    (:negative
      (success)))))))))
```

6.4.2. Ein generischer inkrementeller Instanzenretrieval-Beweiser

Der im Folgenden angegebene Beweiser verwendet *lineares Instanzenretrieval* [Haarslev u. Möller 2002]. Offensichtlich handelt es sich hier nicht um einen Tableau-Beweiser, sodass auch die primär zum Bau von Tableau-Beweisern bereitgestellten höherwertigen Abstraktionen (Tableau-Regeln und Regelanwendungsketten) nicht mehr anwendbar sind. Der Beweiser muss daher unter Verwendung der bereitgestellten Basis-Abstraktionen implementiert werden.

Wir hatten in Kap. 6.2.8, S. 485 die (konzeptionellen) Wächter `obvious_instance` und `obvious_non_instance` zur Optimierung des Instanzentests vorgestellt, und auch die Idee erläutert, dass die originale Eingabe-ABox um zusätzliche abgeleitete *deterministische Assertionen* erweitert wird bzw. dass diese nach Berechnung einer Vervollständigung *beibehalten werden* (s. Kap. 6.2.8, S. 487). Der hier angegebene Beweiser implementiert diese Optimierungen.

Wir definieren den Beweiser zudem so, dass er für die BAA zur Implementation inkrementeller Generatoren verwendet werden kann. Wie auch für die ABFen (s. Kap. 5.2.2.3) verwenden wir wieder Fortsetzungen (Downward Continuations). Sobald eine *C*-Instanz `node` vom Beweiser gefunden wurde, wird die Fortsetzung mittels (`funcall continuation node`) hierüber informiert; der Beweiser blockiert dann solange, bis die Kontrolle aus `continuation` zurückkehrt. So werden vom Instanzenretrieval nur die Instanzen berechnet, die auch vom Nutzer angefordert werden.

Die drei operationalen Phasen des Beweisers nehmen folgende Aufgaben wahr:

`:init` Wird der Instanzenretrieval-Beweiser mit Argumenten *C* und *A* gerufen, so wird zunächst überprüft, ob $\neg C$ erfüllbar ist. Als Seiteneffekt dieses Tests werden das Konzeptmodell und das Kernmodell für $\neg C$ berechnet

6.4 Vorstellung einiger MIDeLoRA-Instantiierungen

(falls diese nicht schon existieren).

- Ist $\neg C$ erfüllbar, so wird geprüft,
 - ob $\neg C \equiv \top$ gilt, denn dann gilt $C \equiv \perp$ und das Ergebnis ist leer.
 - * Ist $\mathcal{A}$ erfüllbar, so wird die Endphase mit leerer Ergebnisliste gerufen (s.u.).
 - * Ist $\mathcal{A}$ hingegen unerfüllbar, so wird eine Fehlermeldung erzeugt.
 - Andernfalls gilt $C \not\equiv \perp$. Die ABox $\mathcal{A}$ wird (bzw. wurde bereits) auf Erfüllbarkeit geprüft; hierbei werden (bzw. wurden) wie beschrieben nun die *deterministischen Assertionen beibehalten*. Für jedes Individuum ist nun sowohl ein Individuumsmodell als auch ein Kern-Individuumsmodell vermerkt.
 - * Ist die ABox erfüllbar, so wird die Hauptphase gestartet. Zuvor werden die zu betrachtenden Knoten identifiziert; diese können auch explizit durch das Argument `nodes` als zu betrachtende Kandidatenmenge spezifiziert werden.
 - * Ist die ABox hingegen unerfüllbar, so wird eine Fehlermeldung erzeugt.
- Ist $\neg C$ hingegen unerfüllbar, so gilt $C \equiv \top$.
 - Ist die ABox erfüllbar, so ist das Ergebnis durch *alle* Individuen der ABox gegeben. Für jedes Individuum wird dann die Fortsetzung gerufen; am Ende wird die Hauptphase mit der Liste aller Individuen als Ergebnisliste gerufen.
 - Ist die ABox hingegen unerfüllbar, so wird eine Fehlermeldung erzeugt.

:main In der Hauptphase wird `linear` über alle mittels `nodes` übergebenen Knoten iteriert. Wie beschrieben, werden die Wächter `obvious-instance-p` und `obvious-non-instance-p` verwendet, um überflüssige Aufrufe des ABox-Erfüllbarkeitstesters zu vermeiden (die benötigten Modelle stehen nun alle bereit, s. `:init`-Phase). Da der nun gerufene ABox-Erfüllbarkeitsbeweiser den originalen Zustand der ABox nicht automatisch durch Zurückrollen wieder herstellt, muss dieses explizit mittels `rollback-to` geschehen. Wurde eine Instanz gefunden, so wird die Berechnung mittels (`funcall continuation node`) fortgesetzt. Die Instanz wird zudem in der Ergebnisliste vermerkt bzw. akkumuliert. Wurden schließlich alle Instanzen berechnet, so wird die Endphase mit der Ergebnisliste als Argument (`... :res res`) gerufen.

:success Die Endphase gibt lediglich die übergebenen akkumulierten Ergeb-

nisinstanzen zurück.

Nach diesen Erläuterungen sollte der folgende Beweiser auch für Leser ohne COMMON LISP-Kenntnisse verständlich sein:

```
(define-prover ((concept-instances dl abox)
                &key concept continuation nodes res)
  (:init
    ;; ist (not concept) erfüllbar?
    ;; als Seiteneffekt wird ein Konzeptmodell sowie ein
    ;; Kern-Konzeptmodell für (not concept) angelegt

    (if (let ((*store-ind-retrieval-models-p* t))
          (sat-p (get-negated-concept concept)))

        ;; (not concept) ist erfüllbar
        ;; ist concept äquivalent zu BOTTOM?

        (if (concept-equivalent-p concept (parse-concept 'bottom))

            ;; Concept ist äquivalent zu BOTTOM
            ;; -> Ergebnis ist leer

            (if (or *continuation-active-p* (abox-sat-p abox))
                (prover-success :res nil)
                (error-abox-inconsistent abox))

            ;; Concept ist nicht äquivalent zu BOTTOM ->
            ;; ist die ABox erfüllbar?

            (if (or *continuation-active-p* (abox-sat-p abox))

                ;; die ABox ist erfüllbar ->
                ;; Instanzenretrieval-Hauptphase starten

                (progn (unless nodes
                        (loop-over-abox-nodes (node abox)
                          (when (old-p node) (push node nodes))))

                      (prover-main :nodes nodes))

                ;; ABox ist unerfüllbar -> Fehlermeldung

                (error-abox-inconsistent abox)))
```

6.4 Vorstellung einiger MIDeLoRA -Instantiierungen

```
;; (not concept) ist unerfüllbar ->
;; concept ist äquivalent zu TOP,
;; Ergebnis ist durch ALLE Individuen gegeben
;; (sofern die ABox erfüllbar ist, sonst Fehler)

(if (or *continuation-active-p* (abox-sat-p abox))
    (let ((res nil))
      (loop-over-abox-nodes (node abox)
        (when (old-p node)
          (when continuation (funcall continuation node))
          (push node res)))
      (prover-success :res res))
    (error-abox-inconsistent abox)))

(:main
  (let ((negated (get-negated-concept concept))
        (action (get-current-action)))

    (with-clash-handler

      ;; alle Knoten aktivieren

      (dolist (node nodes)
        (setf (active-p node) t))

      ;; lineares Instanzenretrieval

      (dolist (node nodes)
        (setf clashes nil)

        (when

          ;;; ist node eine Instanz von concept?

          (cond ((obvious-instance abox node concept) t)
                ((obvious-non-instance abox node concept) nil)
                (t
                 (let ((added (register-as-unexpanded negated
                                                         :node node :new-choice-point 0)))
                   (check-for-clash node added))
                 (prog1
                  (or clashes
                     (not (prover-init 'abox-sat
                                         language abox
                                         :debug-p t)))))))))
```

```

                                :prepare-abox-p nil
                                :prepare-tbox-p nil
                                :maintain-prover-state-p t)))
      (rollback-to abox action))))

    ;;; node ist Instanz von concept

    (push node res)

    (unwind-protect
      (when continuation
        (let ((*continuation-active-p* t))
          (reset-sat-status abox)
          (funcall continuation node)))

      (rollback-to abox action))))

    (prover-success :res res))))

(:success (return res)))

```

6.4.3. Die BAA für MiDELORa

Zusätzlich zu `concept_instances` und `individual_instance?` werden für eine MiDELORa-BAA zudem noch Beweiser für die Aufgaben $\mathcal{T} = \{\text{individuals\_related?}, \text{role\_pairs}, \text{individual\_role\_successors}, \text{individual\_role\_predecessors}\}$ benötigt. Diese verwenden ebenfalls Fortsetzungen. Die Implementationen arbeiten zudem *zweiphasig* (s. Kap. 5.3.2.1.1, S. 323) und nutzen Optimierungsverfahren wie Lemma 5, S. 323. Im Unterschied zu den Generator-ABFen für die RACERPRO-BAA kann hier das Problem der potentiellen *Mehrfachberechnung von Ergebnistupeln* bereits auf der Ebene der Basis-ABox-Retrieval-Funktionen gelöst werden (lediglich die RACERPRO-Funktion `concept_instances` erlaubt die explizite Kandidatenspezifikation).

Die von NRQL genutzten Caches waren zum einen zur Implementation der zweiphasigen und unvollständigen Modi, zur Maximierung der Nebenläufigkeit im Rahmen des Mehrbenutzerbetriebes, und zur Bereitstellung entsprechender (ehemals nicht von RACERPRO bereitgestellter) Indexstrukturen erforderlich. Zum anderen hatten wir die Notwendigkeit von Caches jedoch auch kritisiert, da sie a) verwaltet werden müssen, und b) zu einem erhöhten Speicherbedarf führen, s. Kap. 5.3.3.6.1, S. 367. Da die hier implementierten Basis-ABox-Retrievaldienste direkt auf dem Tableau bzw. der ABox arbeiten, ergibt sich das Problem hier nicht. Das Tableau ist zudem bereits so ausgelegt, dass es die effiziente Traversierung (s. Lemma 5) direkt ermöglicht.

Allerdings ist nun die tupelweise BAA-Anfragebeantwortung direkt mit den erforderlichen Beweisprozessen gekoppelt; die Kontrolle kehrt erst aus der Fortsetzung zurück, wenn das nächste BAA-Ergebnistupel angefordert wird. Die ABox wird daher durch den Beweis- bzw. Anfragebeantwortungsprozess gesperrt und steht somit anderen Prozessen nicht zur Verfügung. Im Gegensatz zu NRQL erlaubt die MIDeLoRA-BAA daher momentan keine nebenläufige Anfragebeantwortung. Wiederum zeigt sich, dass die gleichzeitige Befriedigung mehrerer scheinbar orthogonaler Anforderungen durch einen Entwurf ein nichttriviales Unterfangen ist.

6.4.4. ABox-Erfüllbarkeitsbeweiser für DLs mit Rollenboxen

Zur Implementation von Beweisern für DLs wie $\mathcal{ALCI}_{\mathcal{RA}^\Theta}$, $\mathcal{ALCI}_{\mathcal{RCC}}$, für Sprachen wie SWRL, oder auch für Modallogiken mit komplexen Rahmen-Bedingungen hatten wir die ABox-Klassen RBox-ABox (RBox-ABox) und JEPD-ABox (JEPD-ABox) definiert, die über eine assoziierte RBox verfügen. In Kap. 6.2.7.2 hatten wir spezialisierte Tableau-Regeln namens *abox-enumeration*, *rolebox-application* und *abox-role-completion* speziell zur Implementierung von Beweisern für derartige DLs vorgestellt.

Wir demonstrieren nun, wie diese Regeln zur Definition von ABox-Erfüllbarkeitsbeweisern für die Sprache $\mathcal{ALCI}_{\mathcal{RA}^\Theta}$ und $\mathcal{ALCI}_{\mathcal{RA}^\Theta \text{ JEPD}}$ (diese Sprache umfasst die $\mathcal{ALCI}_{\mathcal{RCC}}$ -Familie) verwendet werden können. Da diese Sprachen i.A. unentscheidbar sind, terminieren die Beweiser nicht für jede Eingabe-ABox.

Zuvor möchten wir jedoch einen wichtigen Implementationsaspekt dieser Spezial-Tableau-Regeln nachreichen – für das DDJ müssen die durch Anwendung dieser Regeln hinzugefügten Rollen-Assertionen mit *Abhängigkeiten* versehen werden:

- Die Regel *abox-enumeration* hatten wir zur Auflösung disjunktiver Rollen-Assertionen bzw. zur Enumeration der Konfigurationen einer ABox mit disjunktiven Rollen-Assertionen entworfen. Da die Regel nichtdeterministisch ist, müssen die in einer Konfiguration hinzugefügten Rollen-Assertionen (die Disjunkten entsprechen) mit Abhängigkeitsinformation versehen werden. Zur Illustration betrachten wir eine ABox mit disjunktiver Rollen-Assertion: $\{i : \forall R.C, (i, j) : R \sqcup S, j : \neg C\}$. Diese ABox hat zwei Konfigurationen, die durch *abox-enumeration* inkrementell hergestellt werden: $\{i : \forall R.C, (i, j) : R, j : \neg C\}$ und $\{i : \forall R.C, (i, j) : S, j : \neg C\}$. Offensichtlich ist nur die erste ABox erfüllbar. Da es sich um eine nichtdeterministische Regel handelt, wird die *abox-enumeration*-Inkarnation eine neue Verzweigungspunkt-ID mittels *get-new-choice-point* vom Kern, hier 1, anfordern, und sowohl $\text{choice\_points}((i, j) : R) := \{1\}$ in Konfiguration 1 als auch $\text{choice\_points}((i, j) : S) := \{1\}$ in Konfiguration 2 setzen.

Wird nun zunächst die Konfiguration $\{i : \forall R.C, (i, j) : R, j : \neg C\}$ hergestellt, so wird der enthaltene Widerspruch $\{j : C, j : \neg C\}$ explizit. DDJ wird nun zu der Regelinkarnation, die den Verzweigungspunkt mit der ID 1 aufgesetzt hat zurückspringen, da $\text{choice\_points}(j : \neg C) \cup \text{choice\_points}(j : C) = \{0\} \cup \{1, 0\} = \{1, 0\}$ ist. Offensichtlich ist $\text{choice\_points}(j : C) := \text{choice\_points}(i : \forall R.C) \cup \text{choice\_points}((i, j) : R) = \{0\} \cup \{1\}$. Dem Verzweigungspunkt 1 entspricht die diskutierte abox-enumeration-Inkarnation. Nach Rückerhalt der Kontrolle muss die Inkarnation also eine *alternative Konfiguration* herstellen. Um Thrashing zu vermeiden, sollte die alternative Konfiguration den Widerspruch nicht mehr enthalten, d.h., die an dem Widerspruch beteiligten Rollen-Assertionen sollten in der Alternativkonfiguration nicht mehr auftauchen. Die am Widerspruch beteiligten Kanten bzw. Rollen-Assertionen werden daher identifiziert, und es wird versucht, eine alternative Konfiguration herzustellen, in der die entspr. Kanten ihre Beschreibung wechseln. Hier würde also die Kante (i, j) bzw. $(i, j) : R \sqcup S$ identifiziert, und das R -Label der Kante würde gegen ein S -Label ausgewechselt, was einem Ersetzen der Assertion $(i, j) : R$ gegen $(i, j) : S$ entspricht.

Wir hatten bereits in Kap. 5.1.2.6.3, S. 255 die von der Substratklasse Disjunktives-Kanten-Substrat (Disjunctive-Edges-Substrate) bereitgestellte (und hier ererbte) Methode `get-next-configuration` so entworfen, dass sie eben diese Funktionalität bietet. Per Argument können diejenigen Kanten spezifiziert werden, deren Label sich ändern soll.

- Nicht nur die Abhängigkeiten der durch abox-enumeration hinzugefügten (disjunktiven) Rollen-Assertionen müssen korrekt berechnet werden, sondern auch die Abhängigkeiten der Rollen-Assertionen, die durch Anwendung der Regel `rolebox-application` entstehen. Diese Regel hatten wir zur Berücksichtigung von Rollenkompositionssaxiomen der Form $S \circ T \sqsubseteq R_1 \sqcup \dots \sqcup R_n$ entworfen. Betrachten wir die ABox $\{(i, j) : R, (j, k) : S\}$ mit assoziierter RBox $\{R \circ S \sqsubseteq T\}$. Durch `rolebox-application`-Anwendung wird $(i, k) : T$ hinzugefügt. Für die Abhängigkeiten bzw. Verzweigungspunkte-IDs gilt $\text{choice\_points}((i, k) : T) := \text{choice\_points}((i, j) : R) \cup \text{choice\_points}((j, k) : S)$; d.h., die „stützenden“ Rollen-Assertionen müssen identifiziert werden. Wir hatten bereits in Kap. 5.1.2.6.4, S. 257 diskutiert, dass die von der Substratklasse RBox-Substrat (RBox-Substrate) bereitgestellte Methode `get-support` diese Kanten bzw. stützenden Rollen-Assertionen identifizieren kann.
- Eine von der Regel `abox-role-completion` hinzugefügte Assertion $(i, j) : \bigsqcup_{R \in \text{roles}(\text{RBox}(\mathcal{A}))}$ bekommt ebenfalls Abhängigkeitsinformationen zugeordnet, die wie folgt berechnet wird:

$$\text{choice\_points}((i, j) : \bigsqcup_{R \in \text{roles}(\text{RBox}(\mathcal{A}))}) := \bigcup_{k \in \{i, j\}} \text{choice\_points}(\text{created\_by}(k)).$$

Die Funktion `created_by` (s. Attribut `created-by`, Klasse ABox-Knoten (ABox-

6.4 Vorstellung einiger MIDeLoRA-Instantiierungen

Node)) gibt die für die Existenz des Knotens verantwortliche *generierende Assertion* zurück. Die Verzweigungspunkte-IDs der generierenden Assertionen werden vereinigt, denn offensichtlich ist die reine Existenz der Knoten i, j für die Etablierung der Assertion $(i, j) : \bigsqcup_{R \in \text{roles}(\text{RBox}(\mathcal{A}))}$ verantwortlich. Somit sind indirekt die ihrerseits für die Existenz der Knoten $\{i, j\}$ verantwortlichen generierenden Assertionen für die Existenz der Rollen-Assertion verantwortlich.

Um nun eine mögliche Zusammenschaltung dieser Spezial-Tableau-Regeln zu illustrieren, geben wir zunächst einen $\mathcal{ALCT}_{\mathcal{RA}^\oplus}$ -Beweiser an:

```
(define-prover ((abox-sat alci-ra-minus rolebox-abox))
  (:init
    (start-main))
  (:main
    (perform (rolebox-application)
      (:body
        (perform (abox-enumeration)
          (:body
            (perform (deterministic-expansion)
              (:body
                (if clashes
                  (handle-clashes)
                  (perform (or-expansion)
                    (:positive
                      (if clashes
                        (handle-clashes)
                        (restart-main)))
                    (:negative
                      (perform (some-expansion)
                        (:positive
                          (if clashes
                            (handle-clashes)
                            (restart-main)))
                        (:negative
                          (success))))))))))
          (:after-successful-clash-repair
            (restart-main))))))
    (:success
      (completion-found)))
```

Die ersichtliche Strategie ist also, zunächst stets die Rollenkompositionssaxiome anzuwenden, und dann die Disjunktionen aufzulösen. Erst dann werden die anderen bekannten Tableau-Regel angewendet; aus deren Sicht liegt also eine ABox ohne disjunktive Rollen-Assertionen vor.

Die Regel abox-enumeration ist mit einem speziellen Ausgang namens *after-successful-clash-repair* versehen, der nach *erfolgreicher* Herstellung einer angeforderten alternativen Konfiguration nach (Wieder-) Betreten des Rückkehr-Einganges genommen wird. Falls das „lokale Reparieren“ hingegen nicht erfolgreich war, so wird die Regel ihrerseits die Kontrolle zurückgeben.

Die nachfolgenden Regelanwendungen sind aus dem $\mathcal{ALCHI}$ -Beweiser bekannt. Aufgrund der inversen Rollen kann wieder keine kombinierte $\exists\forall$ -Regel verwendet werden. Eine Blockierungsbedingung bzw. -Regel wird nicht verwendet, da die DL unentscheidbar ist. Auch Beweiser für unentscheidbare DLs können nützlich sein: Beispielsweise wurde die Taxonomie aus Abb. 3.6 mit dem weiter unten angegebenen $\mathcal{ALCI}_{RAJEPD}$ -Beweiser berechnet.

Für DLs mit JEPD-Eigenschaft wie die $\mathcal{ALCI}_{RCC}$ -Familie verwenden wir eine andere Strategie zur Berechnung der abgeleiteten Rollen-Assertionen, die wir bereits in Kap. 6.2.7.2 diskutiert hatten, und die ohne Anwendung der Regel *rolebox-completion* auskommt. Stattdessen wird die ABox zunächst durch Anwendung von *abox-role-completion* vervollständigt, und dann werden durch *abox-enumeration* lediglich solche Konfigurationen hergestellt, die die Rollenaxiome erfüllen. Dies ist aufgrund der JEPD-Eigenschaft möglich. Die von der Regel gerufene Methode *role-consistent-p* wurde also entspr. überschrieben, sodass relationale Konsistenz der hergestellten Konfiguration sichergestellt wird.

Der angegebene Beweiser arbeitet auf ABoxen der Klasse JEPD-ABox (JEPD-ABox) und ist für die Sprachklasse $\mathcal{ALCIRAJEPD}$ definiert. $\mathcal{ALCI}_{RAJEPD}$ umfasst die $\mathcal{ALCI}_{RCC}$ -Familie; die Sprachklasse $\mathcal{ALCIRAJEPD}$ ist eine Oberklasse von $\mathcal{ALCIRCC}$, sodass der Beweiser für $\mathcal{ALCI}_{RCC}$ bzw. $\mathcal{ALCIRCC}$ anwendbar ist:

```
(define-prover ((abox-sat alci-ra-jepd jepd-abox))
  (:init
    (start-main))

  (:main
    (perform (abox-role-completion)
      (:body
        (perform (abox-enumeration)
          (:body
            (perform (deterministic-expansion)
              (:body
                (if clashes
                  (handle-clashes)
                  (perform (or-expansion)
                    (:positive
                      (if clashes
```

```

        (handle-clashes)
      (restart-main)))
    (:negative
      (perform (some-expansion)
        (:positive
          (if clashes
            (handle-clashes)
            (restart-main))))
      (:negative
        (success)))))))))
  (:after-successful-clash-repair
    (restart-main))))))

```

```

  (:success
    (completion-found)))

```

6.4.5. Ein Beweiser für räumliche ABoxen mit virtuellen Assertionen

Wie in den Anforderungen in Kap. 6.1.5, S. 405 und Kap. 6.1.6, S. 409 motiviert, ermöglicht MIDeLoRA die Definition von ABox-Beweisern für ABoxen mit virtuellen Assertionen. Wir hatten in Kap. 6.2.3.1, S. 430 beschrieben, wie der MIDeLoRA-Entwurf die Berücksichtigung von virtuellen Assertionen durch ABox-Beweiser mit Hilfe von Iteratoren prinzipiell ermöglicht.

Wir beschreiben nun die benötigten Anpassungen für die diskutierten ABox-Erfüllbarkeitsbeweiser im *ALC_{HI}*-Strang, damit diese die in einer räumlichen ABox der Klasse MIDeLoRA-Implicite-Relationen-Karten-Substrat (MIDeLoRA-Implicit-Relations-Map-Substrate) vorhandenen virtuellen räumlichen (RCC-, Δ - und ϵ -) Rollen-Assertionen vom Typ **E2** und **E3** (s. Kap. 6.1.6, S. 409) berücksichtigen. Virtuelle Rollen-Assertionen dieses Typs sind ausreichend, um anfragefokussierte ABoxen (s. Kap. 6.1.5, S. 405) zu realisieren. Prinzipiell sind auch virtuelle Konzept-Assertionen vom Typ **V2** einfach zu implementieren.

Wir hatten in Kap. 4 erwähnt, dass für Instanzen der Klasse Geometrisches-Substrat (Geometric-Substrate) und somit auch für Instanzen von Unterklassen wie MIDeLoRA-Implicite-Relationen-Karten-Substrat (MIDeLoRA-Implicit-Relations-Map-Substrate) momentan lediglich eine Hauptspeicherimplementation existiert. Daher sind alle ABox-Individuen materialisiert. Dies ist eine *wichtige Einschränkung*, denn somit können virtuelle Assertionen vom Typ **V3**, **E4**, **E5** nicht auftreten (diese fordern virtuelle Individuen). In Kap. 6.2.3.2, S. 432 hatten wir analysiert, dass Unvollständigkeit durch „übersehene (virtuelle) Assertionen“ dann nicht auftreten kann, da ausreichend viele Anzeichen für die virtuellen Assertionen in Form der explizit vorhandenen Knoten gegeben sind. Enthält das Label eines Individuums beispielsweise ein $\forall R.C$ - oder $\leq_n R$ -Konzept, so werden automatisch die entspr. R -Nachfolger von der Regel über die Iteratoren `loop-over-role-successors`, `loop-over-role-predecessors` bzw.

über die listenwertigen Methoden `get-role-successors`, `get-role-predecessors` angefordert.

Zur Implementierung der virtuellen Rollen-Assertionen (**E2**, **E3**) überschreiben wir die Iteratoren `loop-over-role-successors`, `loop-over-role-predecessors` für die Klasse `MiDeLoRA-Implicite-Relationen-Karten-Substrat` (`MiDeLoRA-Implicit-Relations-Map-Substrate`) so, dass zunächst die nichtvirtuellen Rollen-Assertionen (vom Typ **E1**) bzw. Nachfolger geliefert werden. Sind diese erschöpft, so werden in einer 2. Phase die von der Klasse `Geometrisches-Substrat` (`Geometric-Substrate`) ererbten räumlichen Iteratoren (s. Kap. 5.1.2.6.9, S. 269 und Kap. 5.1.4, S. 275) zur Berechnung der Rollen-Assertionen vom Typ **E2**, **E3** bzw. der entspr. Nachfolger oder Vorgänger verwendet. Die räumlichen DIS-KQL-Atome, s. Kap. 5.3.4.6.2, S. 397, sind letztlich genauso implementiert. Die Assertionen vom Typ **E2**, **E3** werden hier auch automatisch materialisiert und somit in Assertionen vom Typ **E1** umgewandelt, wodurch Mehrfachberechnung von virtuellen Rollen-Assertionen vermieden wird. Prinzipiell können diese Assertionen bzw. Kanten auch wieder dematerialisiert werden; die einfache „Proof of Concept“-Implementation behält diese jedoch bei.

Nach diesen einfachen Änderungen sind alle Beweiser im *ALC^{HI}*-Strang ohne weitere Änderungen unmittelbar lauffähig auf ABoxen der Klasse `MiDeLoRA-Implicite-Relationen-Karten-Substrat` (`MiDeLoRA-Implicit-Relations-Map-Substrate`). Die `MiDeLoRA-BAA` ist nun auf Instanzen der Klasse `MiDeLoRA-Implicite-Relationen-Karten-Substrat` (`MiDeLoRA-Implicit-Relations-Map-Substrate`) lauffähig, arbeitet inkrementell, nutzt die Räumlichkeit der ABox, etc. Wir haben somit die durch Leitdomäne 1 aufgetretenen Anforderungen an einen DL-Beweiser in dieser Domäne realisiert.

Der oben angegebene generische Instanzenretrieval-Beweiser arbeitet ebenfalls ohne zusätzliche Änderungen auf Instanzen der Klasse `MiDeLoRA-Implicite-Relationen-Karten-Substrat` (`MiDeLoRA-Implicit-Relations-Map-Substrate`). Bei der Berechnung des *Individuumsmodells* eines Knotens zur Ausnutzung durch `obvious-non-instance` muss jedoch sichergestellt werden, dass tatsächlich *alle* den Knoten betreffende Rollen-Assertionen materialisiert wurden, also vom Typ **E1** sind, denn andernfalls ist das mittels `make-model-from-node` erzeugte Knotenmodell unvollständig, und der Modellverschmelzungstest `obvious-non-instance` zu *optimistisch* (gibt also zu oft `TRUE` zurück). Hierdurch würde `concept_instances` unvollständig. Da idealerweise für jedes Individuum (mindestens) ein Individuumsmodell existieren sollte, läuft dies der Idee der „anfragefokussierten ABox“ zunächst entgegen. Natürlich können die Individuumsmodelle aus einer von einem ABox-Erfüllbarkeitstest hergestellten Vervollständigung errechnet werden, wenn *alle* virtuellen Assertionen materialisiert wurden. Dies ist jedoch nur dann möglich, wenn die Menge der virtuellen Assertionen *endlich* und somit *prinzipiell vorausberechenbar* ist, s. Kap. 6.1.5, S. 405, wie z.B. im

Falle von RCC-Relationen. Für virtuelle ϵ - oder Δ -Assertionen ist dies jedoch nicht der Fall, da es hiervon eine überabzählbare Anzahl gibt, und die Anfrage steht ja nicht fest (die *anfragespezifische Aufbereitung der ABox* soll ja gerade vermieden werden), sodass nicht klar ist, welche Assertionen materialisiert werden müssen (dies entscheidet sich erst zur Anfragebeantwortungszeit), s. Diskussion in Kap. 6.1.5, S. 405.

Aus diesem Grund ist der obvious-instance-Test nur für eingeschränkte Arten von räumlichen ABoxen anwendbar, z.B. räumliche ABoxen, in denen die virtuellen Assertionen stets RCC-Assertionen sind. Andererseits muss auf den Test verzichtet werden, wodurch die Effizienz des Instanzenretrieval-Beweisers herabgesetzt wird. Die *Kernmodelle* und somit der obvious-instance-Test sind von dieser Problematik jedoch nicht betroffen, wie man sich leicht klar machen kann (denn der Test wird lediglich *pessimistischer*, gibt also öfter FALSE zurück).

6.4.6. Ein Beweiser für persistenten ABoxen mit virtuellen Assertionen

Die Klasse DB-ABox bietet nun im Gegensatz zum MiDeLoRA-Implicite-Relationen-Karten-Substrat (MiDeLoRA-Implicit-Relations-Map-Substrate) alle Arten von virtuellen Assertionen: **V1**, **V2**, **V3**, **E1**, **E2**, **E3**, **E4**, und **E5**, s. Kap. 6.1.6, S. 409. Dementsprechend sind die in Kap. 6.2.3.2, S. 432 identifizierten Probleme immanent. Wir hatten dort jedoch auch pragmatische Lösungen identifiziert. Notfalls müssen einfach alle virtuellen Individuen bzw. erforderlichen virtuellen Assertionen materialisiert werden; durch entspr. Markierung der Knoten kann sichergestellt werden, dass die entspr. Assertionen vom Beweiser nicht übersehen werden.

Wir überschreiben also wieder die Iteratoren entsprechend, nun für die Klasse DB-ABox. Es müssen nun jedoch auch die *Knoten-Iteratoren* angepasst werden. Die zentrale Methode ist hier die Methode `find-node`, die u.a. vom Iterator `loop-over-abox-nodes` und von `get-abox-nodes` gerufen wird. Wie auch die Kanten-Iteratoren, so werden `loop-over-abox-nodes` und `get-abox-nodes` *zweiphasig* arbeiten: Zunächst werden die expliziten Knoten geliefert, dann wird eine SQL-Datenbank-Abfrage zur Ermittlung der virtuellen Knoten gestartet, die dann i.d.R. materialisiert werden müssen. Die Methode `find-node` befragt also die Datenbank und materialisiert den Knoten dann:

```
(defmethod find-node ((abox db-abox) name &key error-p
                     (look-into-db-p t) &allow-other-keys)
  (let ((node (call-next-method)))
    (or node
```

```

(when look-into-db-p
  (with-abox* (abox)
    (let* ((name-label
            (convert-db-tuples
             (first
              (with-db-connection (db abox)
                (clsql::query
                 (format nil "SELECT name, label
                             FROM NODES
                             WHERE NAME = '~A'"
                           (encode-value name))
                 :database db))))))
      (name (first name-label))
      (label (second name-label)))
    (when name
      (let ((node (create-abox-node abox name
                                     :type 'db-abox-node
                                     :look-into-db-p nil
                                     :update-db-p nil)))
        (dolist (concept label)
          (node-instance node concept :update-db-p nil))
        node))))))
(when error-p
  (error "Node ~A neither found in materialized ABox nor in DB!"
        name))))

```

Auch die Kanten-Iteratoren setzen nun entsprechende SQL-Anfragen ab. Wir sparen uns eine Darstellung.

Alle definierten Beweiser sind nach dieser Änderung auch auf einer DB-ABox lauffähig, mit den diskutierten Einschränkungen. Es sind keine Änderungen an den Definitionen der Beweiser notwendig.

Betrachten wir den Instanzenretrieval-Beweiser, so fällt auf, dass dieser bereits durch den initialen ABox-Erfüllbarkeits alle Individuen anfordert und somit ein vollständiges Materialisieren der virtuellen Individuen in den Hauptspeicher veranlasst. Die Lösung ist nicht optimal, da somit nach erfolgreichem Erfüllbarkeitstest keine virtuellen Assertionen vom Typ **V3**, **E4**, **E5** mehr existieren; es handelt sich damit nicht wirklich um einen „anfragefokussierten Instanzenretrieval-Beweiser“, der nur die erforderlichen Assertionen materialisiert. Dafür ist der Instanzenretrieval-Beweiser jedoch wieder automatisch vollständig, genau wie im Fall der ABox der Klasse `MIDeLoRA-Implicit-Relations-Map-Substrate`. Eine mit den Tableau-Beweisverfahren gekoppelte *informierte* Speicherverwaltung diesbez. ist Gegenstand zukünftiger Untersuchungen, die im Rahmen dieser Arbeit nicht abgeschlossen werden konnten.

6.5. Zusammenfassung

Wir haben das MIDELORA-Baukastensystem zur Konstruktion von aufgaben-, sprach- und substratklassenspezifischen DL-Beweisern entworfen und implementiert. Die Anforderungen an sowie die Architektur von MIDELORA wurden sorgfältig motiviert und von bestehenden DL-Systemen abgegrenzt. MIDELORA ist das (nach Kenntnis des Autors) erste DL-System, für dessen Entwurf die Anforderungen *offene und erweiterbare ABox-Repräsentation*, *Aufgaben- und Sprachspezifität* konsequent berücksichtigt wurden. Der Begriff des MIDELORA-Raums stellt die zentrale Entwurfsmetapher zur Definition von aufgaben-, sprach- und ABox-klassenspezifischen Beweisern bereit.

Zu den Potentialen der MIDELORA-Architektur gehören:

- Die Bereitstellung der technischen Voraussetzungen zur Berücksichtigung virtueller Assertionen.
- Die Bereitstellung der technischen Voraussetzungen zur Realisierung echten dependenzgesteuerten Zurücksetzens.
- Die Bereitstellung der technischen Voraussetzungen zur Realisierung räumlicher und persistenter ABoxen.
- Die Bereitstellung domänenspezifischer Software-Abstraktionen bzw. programmiersprachlicher Abstraktionen zur Realisierung von Beweisern mit inneren Qualitätsmerkmalen, wie: Überschaubarkeit, Verständlichkeit, Wartbarkeit, Anpassbarkeit und Konfigurierbarkeit.

Die anvisierte Flexibilität, Konfigurierbarkeit, Offenheit und konzeptionelle Verständlichkeit und somit Wartbarkeit der mit dem MIDELORA-Baukasten definierten Beweiser wurde durch eine Vielzahl von Beweisern mit (nach Meinung des Autors) sehr konziser und sauberer Struktur demonstriert.

7. Evaluation und Ausblick

Inhaltsangabe

7.1	Evaluation der Anfragebeantwortungsmaschine	525
7.1.1	Effektivität der Basis-Optimierungstechniken	525
7.1.2	Betrachtungen zur Skalierbarkeit	532
7.1.3	Fazit	541
7.2	Evaluation von MiDeLoRA	542
7.2.1	Evaluation des Konzepterfüllbarkeitstesters und der TBox	542
7.2.2	Effektivität der Basis-ABox-Retrieval-Optimierungen	544
7.2.3	Betrachtungen zur ABox-Skalierbarkeit	545
7.2.4	Fazit	551
7.3	Schlusswort und Ausblick	552

Softwaretechnische Flexibilität ist ein wesentliches Leitmotiv in dieser Arbeit. Die erstellten Entwürfe und Implementationen zeichnen sich durch eine hohe Generizität zur Realisierung dieser Flexibilität aus. Nach Meinung des Autors wurde die erzielte Flexibilität hinreichend durch die vorgestellten unterschiedlichen Ausprägungen (die verschiedenen Substrat-Klassen und -Anfragesprachen sowie unterschiedlichen MiDeLoRA-Beweiser) belegt. Es ist wohlbekannt, dass Generizität in der Softwaretechnik oftmals mit Laufzeiteinbußen verbunden ist, die z.B. durch dynamisches Binden bei konsequenter Nutzung der OOP, Verwendung schwergewichtiger Datenstrukturen („Objektgraphen“), etc., erkauft werden muss.

Wir wollen in diesem Kapitel daher evaluieren, ob ontologiebasierte prototypische ISe, die auf dem Rahmenwerk basieren (wie die DLMAPS-Anwendung oder der OWL TREE), auch eine ansprechende Performanz erreichen. Dies ist insbesondere deswegen relevant, da wir in dieser Arbeit *datenintensive* ontologiebasierte IS-Domänen betrachten. Es ist wohlbekannt, dass *Skalierbarkeit* (bzgl. der Größe der extensionalen Wissensbasis bzw. Datenbestandes) bei gleichzeitig hoher Expressivität (und somit Berechnungskomplexität) der verwendeten Ontologiesprache nicht einfach zu erzielen ist.

Wir werden daher in diesem Kapitel die Performanz des Gesamt-Rahmens stichprobenartig evaluieren. Der Gesamt-Rahmen ist durch die Substrat-Klassen zur Daten- bzw. Informationsrepräsentation, die Substrat-Anfragebeantwortungsmaschine zur ontologiebasierten Anfragebeantwortung, und schließlich die in Kap. 6 vorgestellten MiDeLoRA-Beweiser zur Lösung der anfallen-

den Basis-Inferenzprobleme gegeben. Je nach IS-Domäne sind unterschiedliche Substrat-Klassen, Substrat-Anfragesprachen und Ontologiesprachen mit unterschiedlicher Expressivität nutzbar. Sowohl die Komplexität der verwendeten Ontologie- und Daten-Beschreibungssprachen, als auch die Größe der extensionalen Wissensbasis kann daher „skaliert“ werden.

Wir evaluieren zunächst die Performanz der Substrat-Anfragebeantwortungsmaschine. Dabei verwenden wir *zunächst ausschließlich Substrate, die keine DL-Inferenz zur Anfragebeantwortungszeit bedingen*, um ausschließlich die von der Anfragebeantwortungsmaschine benötigte Zeit zu messen. Wir evaluieren zunächst stichprobenartig die Effektivität der Basis-Optimierungstechniken. Schließlich betrachten wir den LUBM-Benchmark [Guo u. a. 2004a] sowie die DISK und die DISK-Referenzanfragen, um die Skalierbarkeit der Anfragebeantwortung zu evaluieren. Die LUBM-TBox findet sich im Anhang A.1 (in KRSS-Syntax). Dabei wählen wir bewusst den LUBM und die DISK-Referenzanfragen, da diese prototypisch für unsere Leitdomänen stehen. Zudem variieren wir für die DISK die Substrat-Repräsentation und demonstrieren hierdurch, dass unterschiedliche Substrat-Repräsentationen unterschiedliche Anfragebeantwortungszeiten bedingen können.

Dann evaluieren wir MIDELORA. Wir evaluieren stichprobenartig sowohl die Performanz der TBox als auch der ABox. Wiederum verwenden wir die LUBM-Anfragen und die DISK-Referenzanfragen, die nun von der MIDELORA-BAA (im vollständigen Modus) beantwortet werden, sodass MIDELORA-Inferenz zur Anfragebeantwortungszeit notwendig wird. Obwohl diese Anfragen auch mit NRQL und somit RACERPRO evaluiert werden könnten, halten wir RACERPRO bewusst aus dieser Evaluation heraus, da RACERPRO nicht Teil der Leistung des Autors ist. Zur Evaluierung reichhaltiger ABoxen evaluieren wir die DISK-Anfragen zudem auf einer räumlichen ABox, und die LUBM-Anfragen abschließend auch auf einer Datenbank-ABox.

7.1. Evaluation der Anfragebeantwortungsmaschine

Wir evaluieren zunächst die Effektivität der in der Anfragebeantwortungsmaschine implementierten Basis-Optimierungen, und dann die Skalierbarkeit (bzgl. der Größe der extensionalen Wissensbasis).

Für die ersten Tests messen wir auch die Stichprobenvarianzen der Anfragebeantwortungszeiten, um einen Eindruck von der Streubreite der Zeiten zu bekommen. Die hierfür erforderlichen Mehrfachevaluationen der Anfragen erhöhen den Zeitbedarf für die Tests jedoch erheblich. Da die meisten gemessenen Varianzen im Bereich von 10^{-5} Sekunden liegen, verzichten wir daher später auf deren Ermittlung. Es ist die Meinung des Autors, dass die gemessenen Zeiten hinreichend signifikant bzw. robust sind.

7.1.1. Effektivität der Basis-Optimierungstechniken

In Kap. 5.2.2.2 hatten wir zwischen semantischen und syntaktischen Basis-Optimierungstechniken unterschieden. Zu den semantischen Techniken gehört die Anfrageklassifikation (Verwendung der QBox als Index), während wir die heuristische Optimierung (Herstellung bzw. Selektion eines effizienten ABPs) als syntaktische Technik bezeichnet hatten. Zudem werden existentielle Variablen (s. Kap. 5.2.2.2.2) identifiziert und während der Anfragebeantwortung sonderbehandelt (lediglich *eine* Bindung wird von Generatoren solcher Variablen etabliert).

Wir demonstrieren die Effektivität jeder dieser Techniken durch Evaluation einer Beispielanfrage mit und ohne diese Technik. Wie bereits begründet, verwenden wir die Maschine zunächst in einem Modus, der keine beschreibungslogische Inferenz zur Anfragebeantwortungszeit erfordert, also beispielsweise NRQL im Modus 1 (s. Kap. 5.3.3.5.1), sodass wir ausschließlich die durch die Anfragebeantwortungsmaschine konsumierte Zeit messen.

7.1.1.1. Codeschablonenbasierte Anfrageübersetzung und -bearbeitung

Wir hatten in Kap. 5.2.2.3.2 die Idee des codeschablonenbasierten Anfrageübersetzer erläutert. Durch die Verwendung von Codeschablonen für die ABFen kann u.U. eine sehr große Anzahl von (funcall continuation)-Funktionsaufrufen und die Traversierung des ASGs zur Anfragebeantwortungszeit eingespart werden. Es existiert dann lediglich eine einzige ABF, nämlich für den Wurzelknoten des ASGs, in der die ABFen aller anderen ASG-Knoten der Anfrage (durch Ineinanderschachtelung) zusammengefasst sind.

Zur Evaluation der Effektivität dieser Idee wurden die Menge der 14 LUBM-

7.1 Evaluation der Anfragebeantwortungsmaschine

Anfragen 5 mal auf einem LUBM-Datenbestand mit 6 Universitätsabteilungen im NRQL Modus 1 ausgeführt. Der NRQL-Modus 1 impliziert, dass ein (durch die ABFen der Konzept- bzw. Rollen-Atome abgesetzter) Instanzen- bzw. Rollenprüfungstest unabhängig vom verwendeten Anfragekonzept bzw. unabhängig von der verwendeten Anfragerolle annähernd die gleich Zeit benötigt. Wir sagen annähernd, da ja zur Evaluierung der Rollen-Atome die $\leadsto$ -Relation bzw. Lemma 5 angewendet wird, sodass die Eigenschaften der Rollen berücksichtigt und somit eine Art einfacher Inferenz stattfindet. Insbesondere ist das Object Property bzw. die Rolle `subOrganizationOf` transitiv, etc. Vereinfachend können wir dennoch annehmen, dass der Zeitbedarf der Generatoren (nach einer gewissen „Einschwingzeit“) nahezu ausschließlich durch die Kardinalitäten der Extensionen bestimmt wird; dies liegt u.a. auch an den verwendeten Caches. Der verwendete Modus ist zudem mit einer Ausnahme (Anfrage 12, s. Kap. 3.2.4.1) vollständig für den LUBM.

Es wurde nun jede einzelne LUBM-Anfrage für den Test 5 mal ausgeführt, und sowohl die gemittelten Anfrage-Vorbereitungszeiten, als auch die gemittelten Anfrage-Ausführungszeiten der 14 Anfragen wurden aufsummiert. Dies wurde einmal mit und einmal ohne codeschablonenbasierte Anfrageübersetzung durchgeführt: Die arithmetischen Mittel der gemittelten und aufsummierten Beantwortungszeiten sind 0.471 Sekunden bei einer Stichprobenvarianz von $4.4999917 * 10^{-5}$ Sekunden bzw. 0.7375 Sekunden mit Stichprobenvarianz $9.062483 * 10^{-5}$ Sekunden. Die entspricht einem Faktor von 1.568. Diese Basis-Optimierungstechnik ist daher nicht so effektiv wie erhofft.

7.1.1.2. Heuristische Optimierung

Zur Bewertung der Effektivität der heuristischen Optimierung betrachten wir die LUBM-Anfrage 9:

```
(retrieve (?x ?y ?z)
  (and (?x Student)
    (?y Faculty)
    (?z Course)
    (?x ?y advisor)
    (?x ?z takesCourse)
    (?y ?z teacherOf)))
```

Offensichtlich gibt es $6! = 720$ verschiedene ABPs. Wir messen die Laufzeit jedes ABPs. Hierzu stellen wir zunächst die 720 Permutations-Anfragen her und schalten den heuristischen Optimierer zur Ausführung einer Permutations-Anfrage aus. Die Reihenfolge der Konjunkte der Anfrage entspricht dann auch ihrem ABP (da der Optimierer ja ausgeschaltet wurde und somit keine Umordnung der Konjunkte vornimmt).

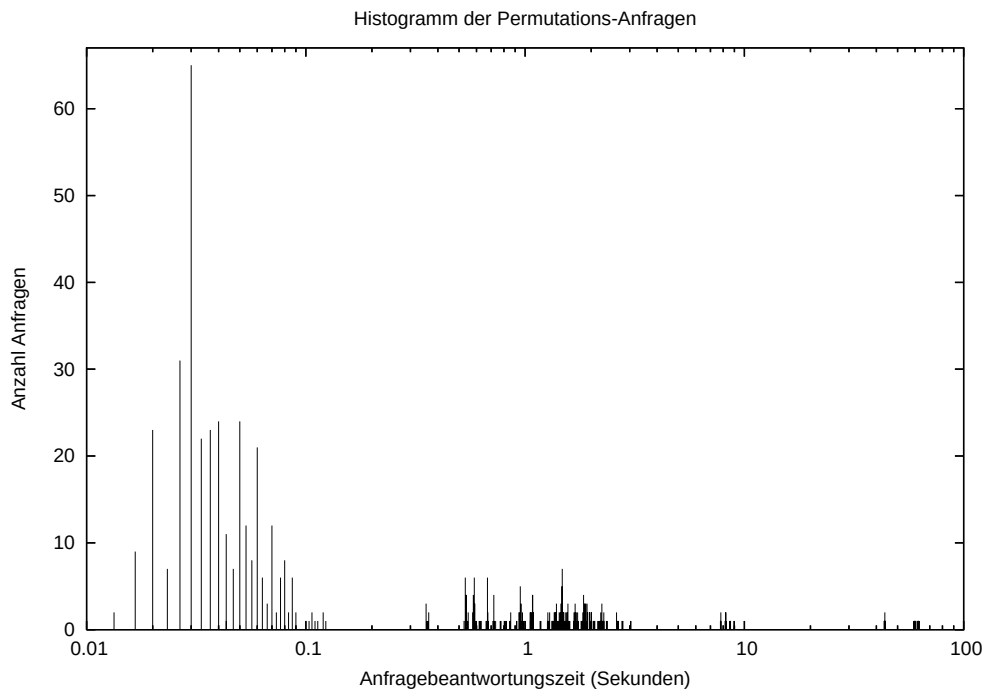


Abbildung 7.1.: Histogramm der Permutations-Anfragen.

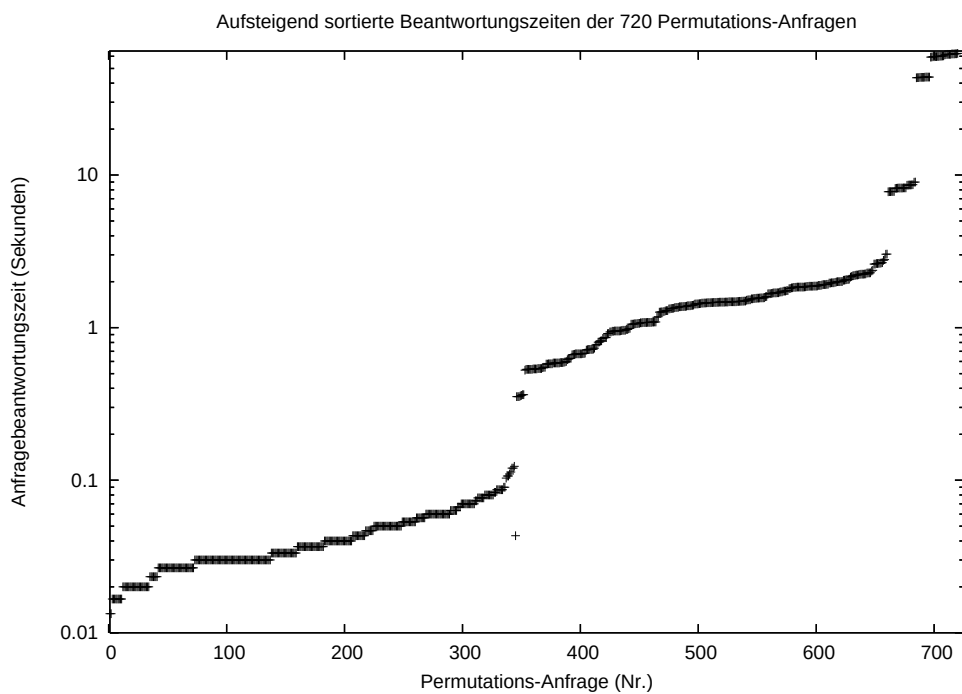


Abbildung 7.2.: Aufsteigend sortierte Anfragebeantwortungszeiten der Permutations-Anfragen.

7.1 Evaluation der Anfragebeantwortungsmaschine

Wir laden nun den LUBM-Datenbestand, jedoch nur eine einzige Universitätsabteilung, da der Test andernfalls zu aufwendig wäre. Die verwendete ABox enthält 2753 Individuen bei 7835 Konzept-Assertionen und 7308 Rollen-Assertionen. Wir nehmen auch vorweg, dass wir aufgrund des Zeitbedarfs des Tests auf die Messung von Stichprobenvarianzen verzichtet haben (der durchgeführte Test benötigte bereits über 24 Stunden auf einer 64-Bit-Maschine mit 4 GB Hauptspeicher, da stets eine vollständige Neuinitialisierung vor Auswertung einer der Permutations-Anfragen erforderlich ist, um vergleichbare Zeiten zu erhalten).

Die aufsteigend sortierten gemessenen Laufzeiten der 720 Permutations-Anfragen sind in Abb. 7.2; Abb. 7.1 zeigt diese als Histogramm (dabei ist auf der x -Achse die Zeit, und auf der y -Achse die Anzahl der Anfragen eingetragen, die in dieser Zeit beantwortet werden konnten).

Die schnellste Permutations-Anfrage

```
(retrieve (?x ?y ?z)
  (and (?y Faculty)
    (?x ?y advisor)
    (?x ?z takesCourse)
    (?y ?z teacherOf)
    (?z Course)
    (?x Student)))
```

benötigt 0.013 Sekunden, die langsamste Permutations-Anfrage

```
(retrieve (?x ?y ?z)
  (and (?z Course)
    (?x Student)
    (?y Faculty)
    (?y ?z teacherOf)
    (?x ?z takesCourse)
    (?x ?y advisor)))
```

benötigt 62.723 Sekunden. Dies entspricht einem Faktor von 4824.846. Somit handelt es sich um den maximalen Beschleunigungsfaktor, der durch die heuristische Optimierung für diese Anfrage und diesen Datenbestand erzielt werden kann.

Schalten wir den Optimierer nun ein, so selektiert dieser für Anfrage 9 den ABP, der der Permutations-Anfrage


```
(retrieve (?x ?y ?z)
  (and (?x Student)
    (?x ?y advisor)
    (?y Faculty)
    (?x ?z takesCourse)
    (?z Course)
    (?y ?z teacherOf)))
```

entspricht. Diese Anfrage benötigt 0.03 Sekunden und ist „nur“ auf Platz Nr. 72 in der Liste der 720 (aufsteigend) nach Anfragebeantwortungszeit sortierten Permutations-Anfragen. Tatsächlich treten unter diesen 72 schnellsten jedoch nur 5 verschiedene Zeiten auf; dementsprechend hat der Optimierer hier die fünftschnellste Anfrage gewählt. Wird nun noch die *kardinalitätsbasierte verfeinerte Optimierungsheuristik* hinzugeschaltet (s. Kap. 5.3.3.6.3), so selektiert der Optimierer den ABP bzw. die Permutations-Anfrage

```
(retrieve (?x ?y ?z)
  (and (?y faculty)
    (?x ?y advisor)
    (?x student)
    (?x ?z takesCourse)
    (?z course)
    (?y ?z teacherOf))),
```

da weniger (bekannte) Fakultäten als (bekannte) Studenten existieren. Diese Anfrage ist mit einer benötigten Beantwortungszeit von 0.023 auf Platz 11 und somit nahe am Optimum; tatsächlich zeigt sie die drittschnellste gemessene Anfragebeantwortungszeit. Da die gemessenen Zeiten in diesen Bereichen jedoch lediglich um Bruchteile von Zehntelsekunden differieren, kann es sich genauso gut um Rauschen handeln, und wir können mit einigem Recht behaupten, dass der Optimierer eine optimale Anfrage gewählt hat.

Die heuristische Optimierung ist also hocheffektiv; ohne den Optimierer würde die langsamste Permutations-Anfrage daher 2727.101 mal langsamer ablaufen. Der Optimierer ist für unsere Zwecke also „gut genug“. Verfeinerte Heuristiken erscheinen hier zunächst nicht erforderlich.

Wie bereits erwähnt, verhindert der Zeitbedarf des Permutationstests bedauerlicherweise eine umfangreichere Evaluation auf größeren Datenbeständen und auch eine Mehrfachevaluation zur Gewinnung der Varianzen. Werden beispielsweise drei Universitätsabteilungen geladen, so benötigen die langsamsten Permutations-Anfragen nun Minuten oder gar Stunden, während die schnellsten Permutations-Anfragen weiterhin nur Zehntelsekunden benötigen.

7.1 Evaluation der Anfragebeantwortungsmaschine

7.1.1.3. Sonderbehandlung existentieller Variablen

Zur Evaluation der Effektivität dieser Optimierungstechnik (s. Kap. 5.2.2.2.2) betrachten wir die Anfrage

```
(retrieve (?x)
  (and (?x Student)
    (?x ?y takesCourse)
    (?y Course)))
```

Wir laden eine Universität und evaluieren die Anfrage 5 mal; die Anfrage liefert 7790 Ergebnisse. Es ergeben sich dann die Mittelwerte 0.606 (Varianz 0.05888) Sekunden vs. 0.614 (Varianz 0.05958) Sekunden bei eingeschalteter gegenüber ausgeschalteter Sonderbehandlung existentieller Variablen. Dies entspricht einem Faktor von 1.013. Die Technik erzielt hier also nur eine unbedeutende Performanzsteigerung.

Offensichtlich ist die Technik für diese Anfrage umso effektiver, je größer die durchschnittliche Anzahl der takesCourse-Nachfolger einer Student-Instanz ist, deren Enumerierung dann für eine bereits an ?x gebundene Student-Instanz eingespart werden kann: Entfernen wir aus der Anfrage das Rollen-Atom, so benötigt die Anfrage ohne existentiell markiertes ?y durchschnittlich 86.843 Sekunden (Varianz 1.2578244) gegenüber durchschnittlichen 0.248 Sekunden (Varianz $2 * 10^{-5}$). Dies entspricht einem Faktor von 350.173.

Allerdings werden solche „Kreuzprodukt“-Anfragen von miteinander unkorrelierten Variablen in der Praxis wohl eher selten gestellt (wahrscheinlich hat der Ersteller der Anfrage einen Fehler gemacht). Es mag jedoch Datenbestände geben, in denen die durchschnittliche Anzahl der Nachfolger eines Individuums eines bestimmten Typs (bzw. einer bestimmten Rolle) sehr groß ist (dies ist beispielsweise in den K_n -artigen RCC-ABoxen der Fall!), sodass diese Technik dann sehr effektiv ist.

7.1.1.4. Anfrageklassifikation

Die QBox kann als hierarchischer Cache angesehen werden. Zur Demonstration der Effektivität der QBox wählen wir eine Klasse von Anfragen, die anders als die bisher ausgewerteten LUBM-Anfragen im Modus 1 (bei eingeschaltetem Optimierer) noch nicht in Sekundenbruchteilen ausgewertet werden können: Die eingeschränkten DISKQL-Anfragen (s. Def. 100, Kap. 5.3.4.5) mit *virtuellen* RCC-Atomen.¹ Diese Atome müssen zur Anfragebeantwortungszeit anhand

¹Der Zeitbedarf zur Auswertung der virtuellen RCC-Atome kann natürlich auch als Schwäche der momentan vorhandenen ABF-Implementationen dieser Atome angesehen werden. Die Atome stehen hier jedoch stellvertretend für alle Atome, deren Evaluation aufwendig ist.

der Geometrie der Karte ausgewertet werden, wozu aufwendige geometrische Inspektionsmethoden notwendig sind. Kann eine solche Anwendung einer Inspektionsmethode aufgrund der gespeicherten Tupel einer in der QBox vorhandenen Eltern-Anfrage der Anfrage vermieden werden, so ist der Beschleunigungsfaktor beträchtlich.

So benötigt die Beantwortung der eingeschränkte DISKQL-Anfrage

```
Q1 = (retrieve (?x ?y) (?x ?y :ppi))
```

auf der DISK mit 2694 (Wurzel-) Kartenobjekten zunächst ganze 155 Sekunden; 1093 Tupel werden zurückgeliefert. Wird nun die Anfrage

```
Q2 = (retrieve (?x ?y) (and (?*x flaeche) (?x ?y :ppi)
                             (?*y flaeche)))
```

gestellt, so wird Q1 als Eltern-Anfrage erkannt. Die Berechnung der 120 Tupel dauert dann nur noch 17 Sekunden. Wird schließlich

```
Q3 = (retrieve (?x ?y) (and (?*x wohnen) (?x ?y :ppi)
                             (?*y teich-nicht-schiffbar)))
```

gestellt, so wird Q2 als Eltern-Anfrage erkannt, und die Berechnung des einen Ergebnistupels dauert 0.38 Sekunden. Schließlich kann

```
Q4 = (retrieve (?x ?y) (and (?*x wohnen) (?x ?y :ppi)
                             (?*y teich)))
```

zwischen Q2 und Q3 einklassifiziert und innerhalb von 0.72 Sekunden beantwortet werden; die Anfrage liefert ebenfalls ein Ergebnistupel.

Werden diese Anfragen hingegen (in dieser Reihenfolge) ohne Anfrageklassifikation beantwortet, so werden die Zeiten 161, 123, 45, und 41 (in Sekunden) benötigt. Der Zeitersparnis-Faktor ist daher $\frac{161+123+45+41}{155+17+0.38+0.72} = 2.1$. Je mehr Anfragen gestellt werden, desto höher ist die Zeitersparnis, entsprechende Subsumptionsbeziehungen vorausgesetzt. Natürlich spielt auch die Reihenfolge, in der die Anfragen gestellt werden, eine wichtige Rolle. Die Technik ist hier also durchaus als effektiv einzustufen.

7.1.2. Betrachtungen zur Skalierbarkeit

Skalierbarkeit ist bzgl. verschiedener Dimensionen wünschenswert:

1. Skalierbarkeit bzgl. der Expressivität der verwendeten Ontologie- und Datenbeschreibungssprachen.
2. Skalierbarkeit bzgl. der Größe der extensionalen Daten- bzw. Informationsbestände.

Die unter 1. genannte Skalierbarkeit ist im vorgestellten Rahmenwerk unmittelbar realisiert, wie hinreichend deutlich geworden sein dürfte. Im Folgenden bezeichne „Skalierbarkeit“ daher die unter 2. genannte Skalierbarkeit. Wir untersuchen diesbez. die 14 LUBM-Anfragen und die 7 DISK-Referenzanfragen:

1. Mit dem LUBM kann Skalierbarkeit für einfache (größtenteils deterministische), aber umfangreiche Wissensbasen nachgewiesen werden. Der LUBM ist hier ideal, da die Größe der extensionalen Wissensbasis beliebig skaliert werden kann (da die Daten von einem Generator-Programm erzeugt werden).

Wir verwenden NRQL wiederum im Modus 1, um RACERPRO aus der Evaluation herauszuhalten. Der Modus ist, wie bereits erwähnt, mit Ausnahme von Anfrage 12 vollständig für den LUBM. Im MIDeLoRA-Evaluationskapitel werden wir diesen Skalierbarkeitstest wiederholen, indem wir anstelle von NRQL im Modus 1 nun die MIDeLoRA-BAA (im vollständigen Modus) verwenden. Vergleichbare Tests wurden auch mit RACERPRO bzw. NRQL im Modus 3 durchgeführt [Haarslev u. a. 2004a, Möller u. a. 2006a;b]. Wie motiviert, wollen wir später jedoch MIDeLoRA verwenden, da RACERPRO nicht Teil der Leistung des Autors ist.

2. Mit den DISK-Referenzanfragen können wir Skalierbarkeit auf einer gänzlich anderen Art von Datenbestand prüfen. Leider stehen nur zwei DISK-Karten zur Evaluierung zur Verfügung, s. Anhang A.3 und Anhang A.4. Wir evaluieren die DISK-Referenzanfragen auf beiden Karten, wobei wir nun auch verschiedene Substrat-Repräsentationen für die Karten verwenden (unterschiedliche Karten-Substrate). Wir können so auch den Einfluss der verwendeten Substrat-Repräsentation auf die Anfragebeantwortungszeit demonstrieren.

7.1.2.1. LUBM ohne DL-Inferenz

Die (intuitive) Bedeutung der folgenden LUBM-Anfragen in NRQL-Syntax sollte offensichtlich sein. Die mit „www“ beginnenden Symbole bezeichnen Individuen; bei subOrganizationOf handelt es sich um eine transitive Rolle, und name, telephone, emailAddress sind OWL Datatype Properties. Erneut sei betont,

dass selbst im Modus 1, der ja keine DL-Laufzeitinferenz erfordert, die Rolleneigenschaften vollständig berücksichtigt werden. $\mathcal{ALCH}_{\mathcal{R}^+}(Dn)$ ist ausreichend für die LUBM-TBox, die in Anhang A.1 in KRSS-Syntax angegeben ist. Die 14 LUBM-Anfragen sehen in NRQL nun wie folgt aus.

- Q1** (retrieve (?x)
 (and (?x GraduateStudent)
 (?x www.Department0.University0.edu/GraduateCourse0
 takesCourse)))
- Q2** (retrieve (?x ?y ?z)
 (and (?x GraduateStudent) (?y University) (?z Department)
 (?x ?z memberOf) (?z ?y subOrganizationOf)
 (?x ?y undergraduateDegreeFrom)))
- Q3** (retrieve (?x www.Department0.University0.edu/AssistantProfessor0)
 (and (?x Publication)
 (?x www.Department0.University0.edu/AssistantProfessor0
 publicationAuthor)))
- Q4** (retrieve (?x (:datatype-fillers (name ?x))
 (:datatype-fillers (emailAddress ?x))
 (:datatype-fillers (telephone ?x)))
 (and (?x Professor)
 (?x www.Department0.University0.edu worksFor)
 (?x (a name)) (?x (an emailAddress)) (?x (a telephone)))))
- Q5** (retrieve (?x)
 (and (?x Person)
 (?x www.Department0.University0.edu memberOf)))
- Q6** (retrieve (?x) (?x Student))
- Q7** (retrieve (?x ?y)
 (and (?x Student) (?y Course) (?x ?y takesCourse)
 (www.Department0.University0.edu/AssociateProfessor0 ?y
 teacherOf)))
- Q8** (retrieve (?x ?y (:datatype-fillers (emailAddress ?x)))
 (and (?x Student) (?y Department) (?x ?y memberOf)
 (?y www.University0.edu subOrganizationOf)
 (?x (a emailAddress)))))
- Q9** (retrieve (?x ?y ?z)
 (and (?x Student) (?y Faculty) (?z Course)
 (?x ?y advisor) (?x ?z takesCourse) (?y ?z teacherOf)))
- Q10** (retrieve (?x)
 (and (?x Student)
 (?x www.Department0.University0.edu/GraduateCourse0
 takesCourse)))

7.1 Evaluation der Anfragebeantwortungsmaschine

- Q11** (retrieve (?x)
 (and (?x ResearchGroup)
 (?x www.University0.edu subOrganizationOf)))
- Q12** (retrieve (?x ?y)
 (and (?x Chair) (?y Department) (?x ?y memberOf)
 (?y www.University0.edu subOrganizationOf)))
- Q13** (retrieve (?x)
 (and (?x Person) (www.University0.edu ?x hasAlumnus)))
- Q14** (retrieve (?x) (?x UndergraduateStudent))

Wir haben die 14 LUBM-Anfragen für 1 bis 4 Universitäten evaluiert. Mehr als 4 Universitäten können auf einem Standard-PC im Jahre 2006 (AMD XP 3000 mit 1 GB Hauptspeicherausbau) nicht geladen werden. Vor der Auswertung von **Q1** wird stets eine Initialisierungsanfrage (**Q-Init**) der Art (retrieve (?x) (?x top)) evaluiert, wodurch einige weitere Initialisierungsvorgänge durchgeführt werden (Vorbereitung und Indizierung des Substrates etc.), die sonst (unfairerweise) **Q1** in Rechnung gestellt würden. Die Initialisierungsanfrage offenbart auch die Anzahl der Individuen. Wir haben nun folgende Werte gemessen, s. auch Abb. 7.3 (hier ist insbesondere „Summe“ bzw. $\sum_i Q_i$ wichtig):

Anfrage	1 Univ.		2 Unis		3 Unis		4 Unis	
	Tupel	Zeit	Tupel	Zeit	Tupel	Zeit	Tupel	Zeit
Ladezeit	–	7.0	–	27.36	–	49.19	–	68.58
Q-Init	17174	5.16	38334	60.88	55664	120.3	78579	200.59
Q1	4	0.01	4	0.015	4	0.0215	4	0.0285
Q2	0	0.044	0	0.389	2	2.0334	6	3.402
Q3	6	0.0095	6	0.014	6	0.0195	6	0.029
Q4	34	0.071	34	0.018	34	0.023	34	0.031
Q5	719	0.009	719	0.037	719	0.044	719	0.053
Q6	7790	0.0045	17878	1.286	26258	1.91	37114	2.8405
Q7	67	0.051	67	0.014	67	0.022	67	0.029
Q8	7790	1.9305	7790	0.604	7790	0.5965	7790	0.5525
Q9	208	0.6675	449	1.188	692	1.817	951	1.1175
Q10	4	0.0065	4	0.012	4	0.0205	4	0.028
Q11	224	0.0135	224	0.019	224	0.027	224	0.035
Q12	0	0.077	0	0.0745	0	0.0865	0	0.0895
Q13	1	0.0065	4	0.013	8	0.02	16	0.029
Q14	5916	0.196	13559	0.394	19868	0.651	28020	0.822
$\sum_i Q_i$	–	4.247	–	4.0775	–	7.292	–	9.0865

Auf einer AMD-64-Bit-Maschine mit 4 GB Hauptspeicher können hingegen bis zu 10 Universitäten geladen werden. Die auf dieser Maschine gemessenen und summierten Beantwortungszeiten sind in Abb. 7.4 dargestellt; der Verlauf ist augenscheinlich linear, was für die Implementierung spricht.

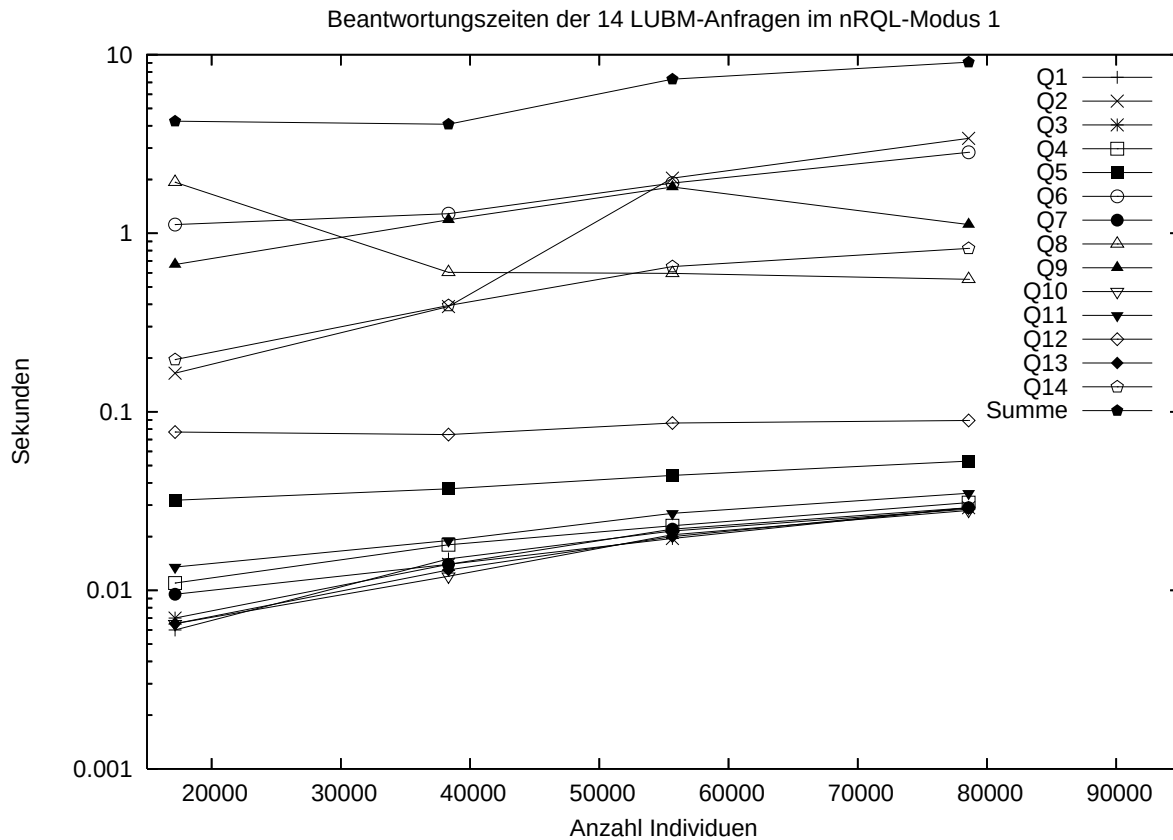


Abbildung 7.3.: Individuelle LUBM-Beantwortungszeiten auf einem Standard-Linux-PC (AMD XP 3000, 1 GB RAM) im Frühling 2006.

Die gemessenen Zeiten liegen für den LUBM also nach Meinung des Autors in durchweg akzeptablen bis guten Größenordnungen; auch relationale Datenbanksysteme erreichen für diese Anfragen keine wesentlich höhere Performanz, s. auch [Rizk 2005] für eine (vom Autor betreute) vergleichende Studie. Das Bild mag sich jedoch ändern, wenn wir entweder die verwendete Expressivität erhöhen, den vollständigen nRQL-Modus 3, oder aber eine Klasse von Anfragen verwenden, für die die Auswertung der ABFen wesentlich aufwendiger ist. Eine solche Klasse von Anfragen sind z.B. die DISKQL-Anfragen mit virtuellen RCC-Atomen (wie bereits dargestellt), deren Performanz wir nun untersuchen wollen.

7.1.2.2. Die DISK-Referenzanfragen

In Kap. 3.1.3.3 bzw. Kap. 5.3.4.4 hatten wir 7 „DISK-Referenzanfragen“ definiert, die wir hier aus Gründen der Leserfreundlichkeit in konkreter DISKQL-Syntax erneut aufführen:

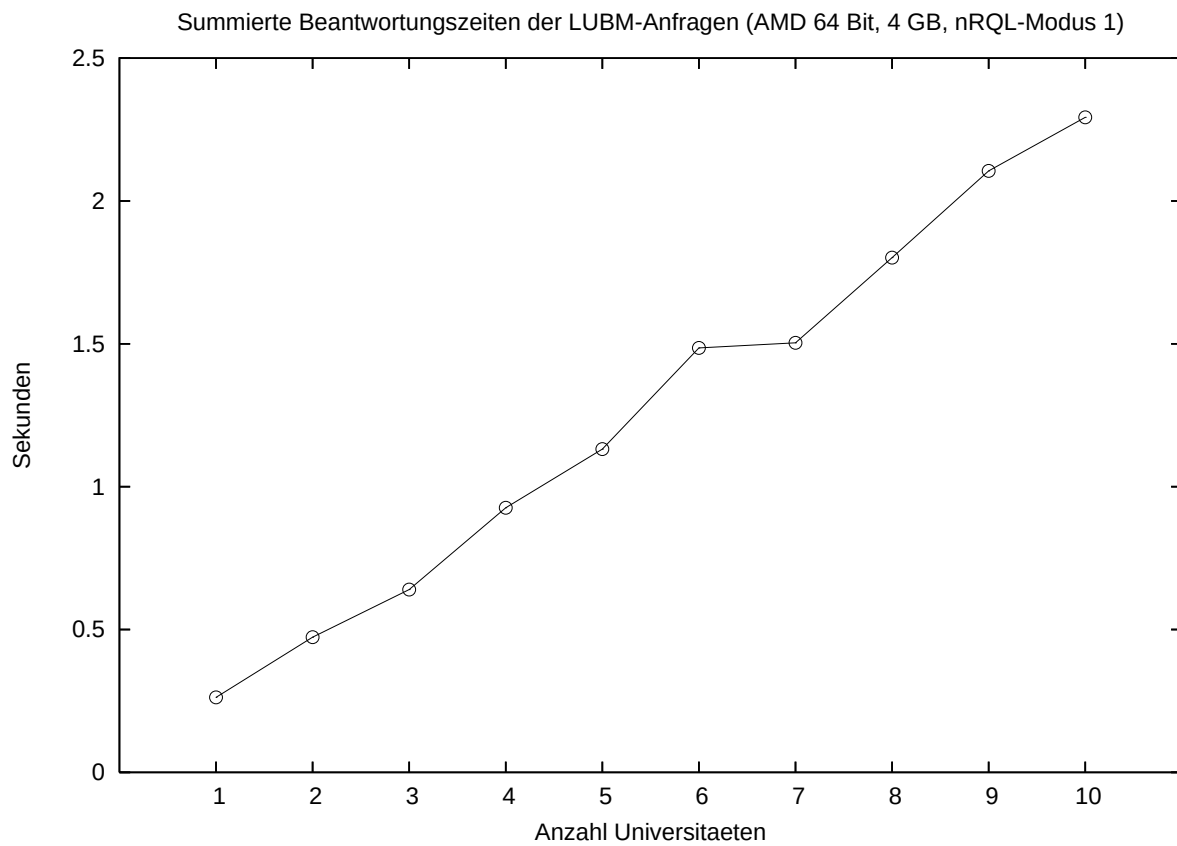


Abbildung 7.4.: Summierte LUBM-Beantwortungszeiten auf einem High-End-Linux-PC (AMD 64 BIT, 4 GB RAM) im Fr hling 2006.

```

Q1 (dlmaps-retrieve (?x) (?x wohngebiet))
Q2 (dlmaps-retrieve (?x ?y)
      (and (?x wohngebiet) (?x ?y :adjacent) (?y autobahn)))
Q3 (dlmaps-retrieve (?w)
      (and (?w wohngebiet) (?g gruenflaeche)
            (?e gewerbegebiet) (?t (or teich see))
            (?w ?g :adjacent) (?w ?e :adjacent) (?g ?t :contains)))
Q4 (dlmaps-retrieve (?x)
      (and (?*x (and wohngebiet (at-least 8 ppi gebaeude)))
            (?x (:area (> area 30000)))))
Q5 (V1) (dlmaps-retrieve (?x)
            (and (?x wohnen)
                  (neg (project-to (?x)
                                   (and (?x ?y :adjacent)
                                         (?y (or industrie gewerbe)))))))
Q5 (V2) (dlmaps-retrieve (?x)
            (and (?x (and wohnen
                          (all ec (not (or industrie gewerbe)))))))
Q6 (dlmaps-retrieve (?x ?y ?z ?u ?f)
      (and (?x gewaesser) (?y naturschutzgebiet) (?z bach)
            (?u gewerbegebiet) (?f fabrik) (?u ?f :contains)
            (?x ?y :inside) (?z ?x :flows-in)
            (?z ?u (:or :touches :crosses))))
Q7 (dlmaps-retrieve (?x ?y)
      (and (?x wohngebiet) (?y ubahn-station)
            (?y ?x (:inside-distance 750))))

```

Wir messen die Beantwortungszeiten der DISK-Referenzanfragen auf den beiden vorliegenden DISK-Karten. Zählen wir in diesen Karten lediglich die Wurzel-Objekte (also keine Komponenten-Objekte) und die nichttrivialen RCC-Relationen zwischen diesen (Kanten der Art $DC(a, b)$ und $EQ(a, a)$ werden nicht mitgezählt), so enthält

1. Karte 1 2693 Objekte und 1104 RCC-Kanten, und
2. Karte 2 18040 Kartenobjekte und 19992 RCC-Kanten.

Wir verwenden stets ein hybrides Karten-Substrat (mit assoziierter RACER-PRO-ABox). Nun können die RCC-Kanten im geometrischen Substrat dieses hybriden Substrates jedoch explizit sein oder nicht. Im ersten Fall werden die RCC-Atome auf *einfache binäre Anfrageatome* (s. Kap. 5.2.2.8) abgebildet, und im zweiten Fall auf *virtuellen räumlichen (RCC-) Atomen* (s. Kap. 5.3.4.3.2). Prinzipiell können hierfür auch Rollen-Atome verwendet werden, sofern die ABox die RCC-Kanten als RCC-Rollen-Assertionen enthält, allerdings zu dem

Preis, dass dann die in den Atomen verwendeten Nicht-Basis-RCC-Relationen zu zusätzlichen Disjunktionen in den Anfragen führen, da in Rollen-Atomen momentan keine disjunktiven Rollen zugelassen sind. So müsste dann beispielsweise das Atom $(?x ?y (:or :ntppi :tppi))$ in die Disjunktion $(or (?*x ?*y :ntppi) (?*x ?*y :tppi))$ mit zwei Rollen-Atomen umgeschrieben werden. Die `:inside-distance`- und `:area`-Atome müssen hingegen stets anhand der Geometrie der Karte durch Inspektionsmethoden ausgewertet werden und sind somit stets „virtuell“. Für die thematischen Atome werden BAA-Konzept-Atome (s. Def. 94) verwendet, so dass beispielsweise das Atom in Endnutzersyntax $(?x \text{ wohngebiet})$ in das Konzept-Atom $(?*x \text{ wohngebiet})$ umgeschrieben wird.

Um zunächst wieder ausschließlich die Performanz der Anfragebeantwortungsmaschine ohne Interferenz durch beschreibungslogische Inferenz zu messen, evaluieren wir die Anfragen wieder in einem Modus, der NRQL Modus 1 entspricht. Dementsprechend werden die Anfragen **Q3**, **Q4** und **Q5** keine Ergebnistupel liefern, denn diese Anfragen enthalten Konzept-Atome mit Disjunktionen bzw. Anzahlrestriktionen. Wir können die Disjunktion im Konzept-Atom $(?t (or \text{ teich see}))$ gegen den (stärkeren) Rumpf $(or (?t \text{ teich}) (?t \text{ see}))$ ersetzen und nennen die so modifizierte Anfrage nun **Q3 (V2)**. Die originale Anfrage **Q3** nennen wir **Q3 (V1)**.

Schließlich nehmen wir noch eine weitere, hinreichend komplizierte Anfrage in die Liste der Anfragen auf, um die „Grenzen des Systems“ auszuloten:

```
Q8 (dlmaps-retrieve (?w ?g ?e ?t ?u)
      (and (?w wohnen) (?t teich) (?u u-bahn-station)
            (or (?g wiese-weide) (?g park))
            (?w ?g :adjacent) (?w ?e :adjacent) (?g ?t :contains)
            (?u ?w (:inside-distance 750))
            (neg (project-to (?w)
                           (and (?w wohnen) (?x autobahn) (?w ?x :adjacent))
```

Wir verwenden nun zunächst ein Substrat mit expliziten RCC-Kanten. Dies impliziert, dass die RCC-Kanten zunächst in einer Initialisierungsphase (beim Anlegen des Substrates) errechnet werden müssen. Wiederum wird vor Auswertung von **Q1** eine Initialisierungsanfrage (Anfrage $(dlmaps-retrieve (?x) (?x \text{ top}))$) evaluiert. Die folgenden Zeiten wurden gemessen:

DISKQL-Anfrage	Karte 1		Karte 2	
	Anzahl Tupel	Zeit (Sek.)	Anzahl Tupel	Zeit (Sek.)
Lade- und Indizierungszeit	–	72.492	–	900.168
Initialisierungsanfrage	2693	1.6	18040	17.642
Q1	76	0.02	490	0.044
Q2	4	0.26	24	2.624
Q3 (V1)	0	0.249	0	5.216
Q3 (V2)	12	1.048	6	18.441
Q4	0	0.192	0	0.828
Q5 (V1)	76	0.304	490	12.533
Q5 (V2)	0	0.02	0	0.036
Q6	1	1.416	1	23.006
Q7	69	0.316	250	3.764
Q8	102	4.068	71	47.099
$\Sigma_i \mathbf{g}_i$	–	7.893	–	113.591

Insbesondere für Karte 2 ist die Lade- und Indizierungszeit, welche die für die Vorausberechnung und Materialisierung aller RCC-Relationen als Kanten benötigte Zeit mit einschließt, nun nicht mehr zu vernachlässigen. Ansonsten halten wir eine Gesamtbeantwortungszeit von 7.893 Sekunden bzw. 113.591 Sekunden für akzeptabel.

Verwenden wir nun ein Substrat ohne explizite RCC-Kanten, so müssen nicht nur die :inside-distance- und :area-Atome, sondern auch die RCC-Atome in den Anfragen durch Inspektionsmethoden zur Anfragebeantwortungszeit ausgewertet werden. Dafür kann die zeitaufwendige Indizierungszeit entfallen. Es ergeben sich dann folgende Zeiten:

DISKQL-Anfrage	Karte 1		Karte 2	
	Anzahl Tupel	Zeit (Sek.)	Anzahl Tupel	Zeit (Sek.)
Lade- und Indizierungszeit	–	50.759	–	400.689
Initialisierungsanfrage	2693	0.536	18040	21.409
Q1	76	0.02	490	0.032
Q2	4	11.197	24	40.211
Q3 (V1)	0	10.145	0	147.985
Q3 (V2)	12	90.613	6	174.767
Q4	0	0.148	0	1.448
Q5 (V1)	76	0.364	490	15.985
Q5 (V2)	0	0.012	0	0.048
Q6	1	8.125	1	74.129
Q7	69	0.136	250	2.196
Q8	102	139.208	71	466.473
$\Sigma_i \mathbf{g}_i$	–	259.968	–	923.274

Es wird deutlich, dass die Auswertung der räumlichen Beziehungen durch Inspektionsmethoden zur Anfrageauswertungszeit der virtuellen RCC-Atome sehr

7.1 Evaluation der Anfragebeantwortungsmaschine

teuer ist; die Implementationen sind nicht als effizient einzustufen. Dies wird insbesondere anhand von **Q3 (V2)** auf Karte 1 deutlich, wo ein Verlangsamungsfaktor von 86.46 gemessen wurde.

Umso wichtiger ist also, dass unser Rahmen verschiedene Substrat-Klassen für eine Repräsentationsaufgabe zur Verfügung stellt, sodass die für die Anwendung geeignete Repräsentation gewählt werden kann.

7.1.2.2.1. Zur Effektivität des für DISKQL-Anfragen verfeinerten heuristischen Optimierers Der generische heuristische Optimierer wurde für DISKQL-Anfragen verfeinert, s. Kap. 5.3.4.6.3. Wie dort erläutert, wurden die get-score-Bewertungsmethoden des Optimierers für die räumlichen Atome so verfeinert, dass Wissen über die im räumlichen Atom referenzierte räumliche Beziehung in die Bewertungsheuristik mit eingeht. Diese Verfeinerung ermöglicht erst die soeben gemessenen Zeiten: Mit den Standardbewertungsmethoden würden z.B. für **Q8** die beiden ABPs

<pre>(or (and (?*ubahn u-bahn-station) (?ubahn ?w (:inside-distance 750)) (?*w wohnen) (?w ?g :ec) (?*g wiese-weide) (?g ?t (:or :ntppi :tppi)) (?*t teich) (?w ?e :ec) (neg (:project-to (?w) (and (?*x-ano1-ano2 autobahn) (?w ?x-ano1-ano2 :ec) (?*w wohnen)))))) ...)</pre>	<pre>(or (and (?*w wohnen) (?w ?g :ec) (?*g wiese-weide) (?g ?t (:or :ntppi :tppi)) (?*t teich) (?w ?e :ec) (?ubahn ?w (:inside-distance 750)) (?*ubahn u-bahn-station) (neg (:project-to (?w) (and (?*x-ano1-ano2 autobahn) (?w ?x-ano1-ano2 :ec) (?*w wohnen)))))) ...)</pre>

vom Optimierer gleichbewertet werden (unter ... steht jeweils die gleiche Konjunktion noch einmal, jedoch mit dem Atom (?*G PARK) statt (?*G WIESE-WEIDE)). Der links stehende ABP kann auf einem Substrat mit expliziten RCC-Kanten für Karte 1 in ca. 4 Sekunden ausgewertet werden. Im Gegensatz hierzu benötigt der rechte ABP über 30 Sekunden. Die für die DISKQL angepassten, verfeinerten und daher informierteren Bewertungsheuristiken stellen daher sicher, dass der linke ABP vom Optimierer gewählt wird, was hier wieder einen Beschleunigungsfaktor von 7.5 garantiert.

7.1.3. Fazit

Wir haben die Effizienz der Substrat-Anfragebeantwortungsmaschine evaluiert. Hierbei haben sich die Basis-Optimierungstechniken als (je nach Anfrage- und verwendeter Substrat-Klasse) unterschiedlich effektiv erwiesen. Die heuristische Optimierung ist als wichtigste unbedingt notwendige Optimierungstechnik einzustufen; ohne sie können selbst einfache Anfragen auf vergleichsweise kleinen Datenbeständen (z.B. einer Universitätsabteilung) nicht in akzeptablen Zeitspannen beantwortet werden. Die Streubreite der Permutationsanfragen kann um einen Faktor von bis zu 10.000 variieren. Andere Techniken, wie die Anfrageklassifikation, sind nur für bestimmte Klassen von Anfragen effektiv. Die Art der Substrat-Repräsentation hat einen großen Einfluss auf die benötigten Beantwortungszeiten, wie anhand der DISK-Anfragen auf einem Substrat mit und ohne explizite RCC-Kanten verdeutlicht. Skalierbarkeit wurde für bestimmte Klassen von Anfragen und Substraten in gewissen Grenzen erreicht (sofern der Hauptspeicher ausreicht), wie anhand des LUBMs belegt. Skalierbarkeit ist schwieriger zu erreichen für die DISK. Momentan sind entweder die für die Vorausberechnung aller RCC-Relationen benötigten Zeiten zu hoch, oder aber die Auswertungszeiten der virtuellen RCC-Atome. Deren ABF-Implementationen müssen noch verbessert werden. Für eine „Proof of Concept“-Implementation kann jedoch gesagt werden, dass die hier erzielten Zeiten ausreichend performant sind (zumal es in dieser Arbeit nicht primär um die Erstellung maximal effizienter Inspektionsmethoden für RCC-Atome geht). Das vorliegende Rahmenwerk kann also durchaus für derartige IS-Domänen mit guter bis ausreichender Performanz bei der Anfragebeantwortung genutzt werden.

Dieses Ergebnis mag sich jedoch ändern, wenn nun auch noch beschreibungslogische Inferenz während der Anfragebeantwortung notwendig wird. Dann wird es wesentlich von der Performanz des verwendeten DL-Systems abhängen, ob die ABFen der Atome effizient sind. Wir verwenden im Folgenden MIDELORA, um die (im Sinne unserer Definitionen) vollständigen Anfragebeantwortungs-Modi zu evaluieren.

7.2. Evaluation von MiDeLoRA

MiDeLoRA wurde nicht mit dem Primärziel „Erreichung maximaler Effizienz“ entworfen und implementiert, sondern unter den Gesichtspunkten der *Offenheit, Flexibilität, Erweiterbarkeit, Konfigurierbarkeit und Wartbarkeit* (s. Kap. 6). Natürlich ist aber auch für MiDeLoRA eine gewisse Effizienz erforderlich, da die MiDeLoRA-BAA sonst nicht im vollständigen Modus für ontologiebasierte ISe mit großen extensionalen Wissensbasen verwendet werden kann (Skalierbarkeit wäre nicht gegeben). Ziel der hier durchgeführten Arbeiten sollte es jedoch sein, MiDeLoRA für Leitdomäne 1 und Leitdomäne 2 nutzen zu können, um die gestellten Anforderungen umzusetzen.

Wir evaluieren nun zunächst die Effizienz der intensionalen Komponenten von MiDeLoRA, also den Konzept-Erfüllbarkeitsbeweiser und die TBox. Schließlich testen wir die Effizienz und Skalierbarkeit der ABox bzw. der MiDeLoRA-BAA im vollständigen Modus.

7.2.1. Evaluation des Konzepterfüllbarkeitstesters und der TBox

In [Karabaev u. Lutz 2004] wurde ein Benchmark-Wert zum Vergleich der Effizienz des Konzepterfüllbarkeitstesters und der Taxonomie-Berechnungsalgorithmen unterschiedlicher DL-Systemen vorgeschlagen, der auf der „Tableaux '98 (T98) Benchmark Suite“ basiert. Bei der „T98 Benchmark Suite“ handelt es sich um eine Menge von *ALC*-Konzepten, die jeweils innerhalb von 100 Sekunden auf Erfüllbarkeit bzw. Unerfüllbarkeit geprüft werden müssen. Die Menge ist wiederum in problemspezifische (und mit Namen wie *k_branch*, *k_d4*, ... versehene) Untermengen bzw. Klassen von Konzepten partitioniert. Die Konzepte in den Klassen sind dabei nach zunehmender Komplexität geordnet. Es handelt sich um *ALC*-Konzepte. Jede Klasse enthält 21 Konzepte. Dem Evaluationsschema aus [Karabaev u. Lutz 2004] folgend, wird MiDeLoRA nun für jedes Konzept ein Zeitrahmen von 100 Sekunden zur Entscheidung der Erfüll- bzw. Unerfüllbarkeit zugestanden. Pro Klasse wird das schwierigste bzw. maximale Konzept der Klasse, welches innerhalb des Zeitrahmens noch getestet werden kann, ermittelt (die Konzepte pro Klasse sind ja nach zunehmender Schwierigkeit geordnet).

Die MiDELORA-Messergebnisse sind in folgender Tabelle wiedergegeben:

k_branch_n	k_branch_p	k_d4_n	k_d4_p	k_dum_n	k_dum_p
9	21	21	21	21	21
k_grz_n	k_grz_p	k_lin_n	k_lin_p	k_path_n	k_path_p
21	21	21	21	21	21
k_ph_n	k_ph_p	k_poly_n	k_poly_p	k_t4p_n	k_t4p_p
9	7	21	21	21	21

Die Summe dieser Zahlen ergibt den Benchmark-Wert: 340. Für (inzwischen leicht veraltete) Vergleichszahlen mit anderen Beweisern s. [Karabaev u. Lutz 2004].

In [Karabaev u. Lutz 2004] wurde zudem noch drei weitere Benchmark-Kennzahlen vorgeschlagen. Zwei hiervon verwenden die Galen1- und Galen2- $\mathcal{ALC}$ -TBoxen. Hier wird gemessen, wieviel Prozent der Konzepte dieser TBoxen innerhalb von 100 Sekunden auf Erfüllbarkeit getestet werden können. Wir nehmen, um auch eine $\mathcal{ALCHf}_{\mathcal{R}^+}$ -Wissensbasis zu testen, noch die Galen-TBox in diese Liste mit auf, und messen zudem auch die Klassifikationszeiten. Es ergeben sich folgende Messwerte:

TBox	Sprache	Anzahl Konzepte	Erfüllbarkeit (Sek.)	Klassifikation (Sek.)
Galen1	$\mathcal{ALC}$	2728	32 ($\rightarrow$ 100 %)	264
Galen2	$\mathcal{ALC}$	3926	18 ($\rightarrow$ 100 %)	248
Galen	$\mathcal{ALCHf}_{\mathcal{R}^+}$	2749	42 ($\rightarrow$ 100 %)	258

Diese Werte sind vergleichsweise schlecht; dennoch können alle Konzepte ($\rightarrow$ 100%) der entspr. TBoxen innerhalb der 100-Sekunden-Schranke auf Erfüllbarkeit geprüft werden. Der in MiDELORA implementierte Klassifikationsalgorithmus muss noch weiter optimiert werden, denn MiDELORA macht bei der TBox-Klassifikation noch zu viele „echte“ Subsumptionstests (und nutzt syntaktisch gegebene, offensichtliche Subsumptions- und offensichtliche Nicht-Subsumptionsbeziehungen beim Klassifikationsprozess noch nicht maximal aus). Auch scheinen die berechneten GCI-Absorptionen noch nicht optimal zu sein. Wir stellen fest, dass in Systeme wie RACERPRO diesbez. wesentlich mehr Arbeit geflossen ist. Da der Schwerpunkt dieser Arbeit jedoch nicht auf der intensionalen, sondern auf den extensionalen Komponente inkl. Anfragesprachen liegt, können diese Zahlen zunächst toleriert werden. Für kleine bis mittlere TBoxen ist MiDELORA „schnell genug“, denn beispielsweise benötigt die Klassifikation der LUBM-TBox nur Sekundenbruchteile.

7.2.2. Effektivität der Basis-ABox-Retrieval-Optimierungen

Nachdem die Effizienz der BAA (bzw. von NRQL) im unvollständigen Modus anhand des LUBMs demonstriert wurde, sollen nun die vollständigen Modi der MIDeLoRA-BAA evaluiert werden. Nun wird nahezu die gesamte benötigt Anfragebeantwortungszeit durch die ABFen der Konzept- und Rollen-Atome verursacht werden. Die Basis-ABox-Retrieval-Dienste müssen daher maximal effizient implementiert werden. Hierzu wurden entsprechende Techniken vorgestellt, sowohl für die Instanz- und Rollen-Prüfungsprobleme (*individual_instance?*, *individuals_related?*), als auch für die Instanz-, Rollenfüller-, und Rollenpaare-Retrieval-Probleme (*concept_instances*, *role_fillers*, *role_pairs*): Für die Rollen-Atome werden die durch die Lemmata 5 und 6 implizierten Optimierungen angewendet, und für die Konzept-Atome werden die Tests zur Erkennung *offensichtlicher Instanzen* (*obvious_instance*) und *offensichtlicher Nicht-Instanzen* (*obvious_non_instance*) angewendet.

Zur Illustration, wie essentiell diese unvollständigen Verfahren sind, betrachten wir die LUBM-Anfrage **Q9**, für die der Optimierer folgenden ABP herstellt:

```
Q9 (ABP)    (retrieve (?x ?y ?z)
               (and (?y Faculty)
                    (?y ?z teacherOf)
                    (?z Course)
                    (?x ?z takesCourse)
                    (?x Student)
                    (?x ?y advisor)))
```

Diese Anfrage liefert auf einer Universitätsabteilung (diese enthält 1555 Individuen) 13 Ergebnistupel, und die Beantwortung dauert im vollständigen Modus der MIDeLoRA-BAA ca. 1.3 Sekunden. Der initiale ABox-Erfüllbarkeitstest (in dem auch die benötigten Individuums- und Kernmodelle erzeugt werden) dauert ca. 12 Sekunden. Zur Beantwortung der Anfrage fallen 1555 (durch die Tester- bzw. Generator-ABFen der Konzept-Atome abgesetzte) Instanzentests an; aufgrund der Caches werden Doppelprüfungen vermieden (die Zahl wäre sonst höher). Glücklicherweise können *alle* diese Instanzentests durch *obvious_instance* (41 Treffer) und *obvious_non_instance* (1514 Treffer) entschieden werden, sodass kein einziger ABox-Erfüllbarkeitstest erforderlich wird.

Verzichten wir nun auf die Nutzung von *obvious_instance* und *obvious_non_instance* bei der Auswertung der Anfrage, so fallen also 1555 ABox-Unerfüllbarkeitstests an. Ein einziger Test dauert durchschnittlich 12 Sekunden. Die Beantwortung der Anfrage würde dann bereits 18660 Sekunden bzw. 5.18 Stunden benötigen. Durch die *Beibehaltung deterministischer Assertionen* während der Anfragebearbeitung (s. Kap. 6.2.8) kann die für einen durchschnittlichen ABox-Unerfüllbarkeitstest benötigte Zeit auf durchschnittlich ca. 3.7 Sekunden re-

duziert werden. Die Beantwortung der Anfrage würde dann jedoch immer noch ca. 1.6 Stunden benötigen. Es wird deutlich, dass die unvollständigen Wächter unbedingt notwendig sind.

Schließlich können wir noch die verwendeten Rollen-Atome betrachten. Da $\mathcal{ALCH}_{\mathcal{R}}^+$ ausreichend ist für den LUBM, werden zur (vollständigen) Berechnung der Extensionen der Rollen-Atome keine ABox-Unerfüllbarkeitstests erforderlich (s. Lemma 5). Schalten wir diese Optimierung aus, so werden zur Beantwortung der Anfrage auf diesem Datenbestand 4133 Rollenprüfungstests erforderlich, wobei diese Zahl schon aufgrund von Lemma 6 minimiert wurde. Ein naiver Algorithmus würde schlechtestenfalls sogar $1555^2 = 2418025$ Tests durchführen. Unter der Annahme, dass all diese Tests auf ABox-Unerfüllbarkeitstest reduziert werden müssten, würden schlechtestenfalls zusätzliche $4133 * 3.7$ Sekunden = 4.25 Stunden anfallen. Die Berücksichtigung von Lemma 5 in den Rollen-ABFen spart hier daher 4.25 Stunden; analog spart Lemma 6 gegenüber dem naiven Verfahren $2418025 * 3.7$ Sekunden – 4.25 Stunden = 2485.19 Stunden – 4.25 Stunden = 2480.94 Stunden bzw. 103.37 Tage. Natürlich handelt es sich bei diesen Zahlen um (fundierte) Schätzungen. Doch ist das Fazit aus diesen Zahlen sehr deutlich: Ohne diese Optimierungen sind die LUBM-Anfragen selbst auf einem Datenbestand mit einer einzigen Universitätsabteilung nicht beantwortbar.

7.2.3. Betrachtungen zur ABox-Skalierbarkeit

Wiederum betrachten wir die LUBM- und die DISK-Referenzanfragen.

7.2.3.1. Der LUBM mit MIDeLoRA

MIDeLoRA kann keine OWL-Dateien verarbeiten. Daher wurde RACERPRO verwendet, um die LUBM-OWL-Dateien in Dateien in KRSS-Syntax zu transformieren (s. Anhang A.1). Die Datatype Properties `name`, `emailAdress` etc. werden als normale Rollen behandelt. Die 14 Anfragen wurden auf Datenbeständen zunehmender Größe evaluiert, beginnend mit einer einzelnen Universitätsabteilung, bis zu einer kompletten Universität mit 15 Universitätsabteilungen. Aus Speicherplatzgründen verzichteten auf die Evaluation von mehr als einer Universität mit MIDeLoRA (1 GB ist dann nicht mehr ausreichend).

Die wichtigsten Kennzahlen sind in Abb. 7.5 dargestellt: Die Ladezeit, die für den initialen ABox-Erfüllbarkeitstest benötigt Zeit, die für die Ausführung der Initialisierungsanfrage benötigte Zeit, sowie die sog. *Zurückroll-Zeit* (s.u.). *Der augenscheinlich sub-lineare Verlauf der Kurven (insbesondere bzgl. der Zeit für die ABox-Erfüllbarkeit) lässt Skalierbarkeit bzw. ein nichtexponentielles Wachs-*

7.2 Evaluation von MiDeLoRA

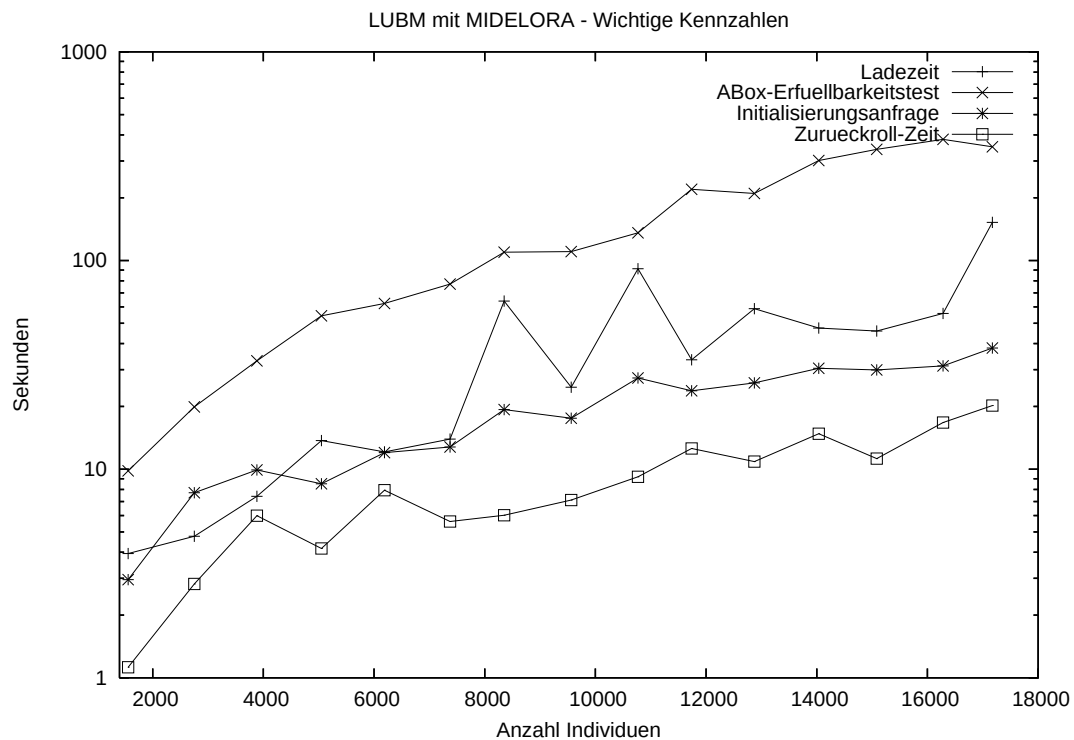


Abbildung 7.5.: Der LUBM mit MiDeLoRA– Wichtige Kennzahlen.

tum vermuten.² Schließlich sind in Abb. 7.6 die Anfragebeantwortungszeiten der LUBM-Anfragen dargestellt; insbesondere auch die Gesamt-Anfragebeantwortungszeit.

Für eine ganze Universität ergeben sich folgende Kennzahlen:

Anzahl Individuen	17174
Lade- und Indizierungszeit	152.458 Sekunden
ABox-Erfüllbarkeitstest	350.777 Sekunden
Initialisierungsanfrage	38.107 Sekunden
Summierte Beantwortungszeiten ($\sum_i g_i$)	90.63 Sekunden
Angefallene Instanzen-Tests	68711
obvious_instance-Treffer	14261 (20.76 %)
obvious_non_instance-Treffer	54435 (79.22 %)
Benötigte ABox-Unerfüllbarkeitstests	15 (0.02 %)
Globale Zurückroll-Zeit	20.173 Sekunden

Demnach benötigt die Auswertung aller 14 LUBM-Anfragen 90.63 Sekunden. Von den während der Anfragebeantwortung angefallenen 68711 Instanzentests

²Da es sich um eine logarithmische Skala handelt, würde ein linearer Verlauf exponentielles Wachstum bedeuten.

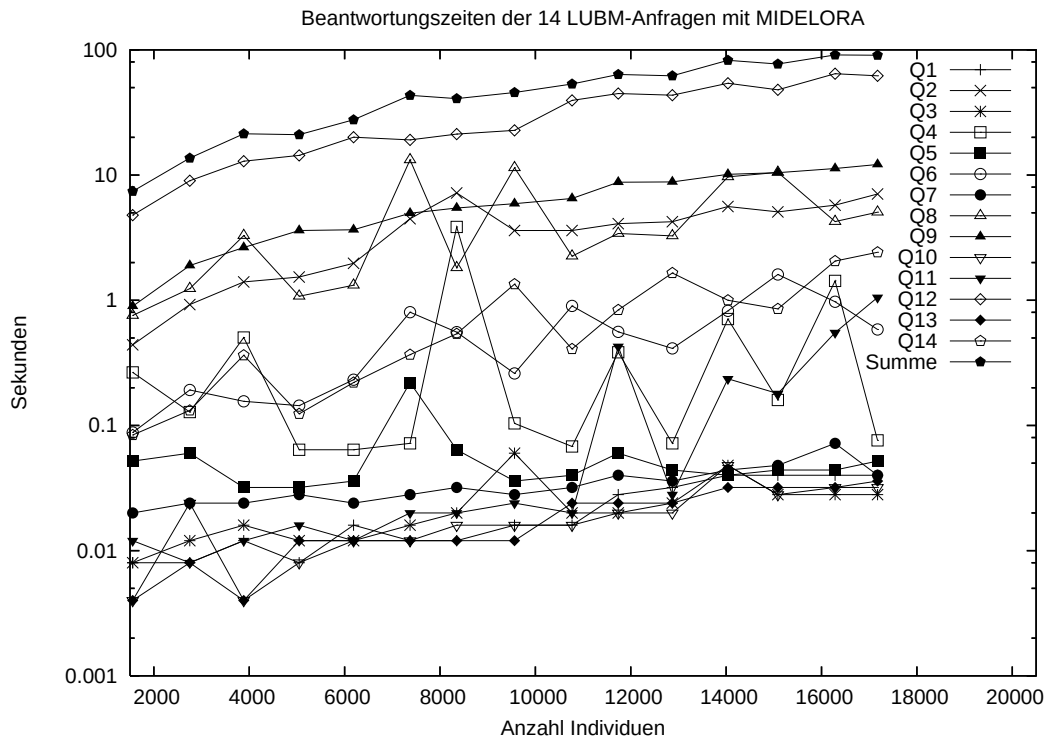


Abbildung 7.6.: Beantwortungszeiten der 14 LUBM mit MIDELORA.

konnten 99.98 % durch die unvollständigen Methoden entschieden werden. Lediglich 15 ABox-Unerfüllbarkeitstests wurden zur Entscheidung der 0.02 % restlichen Instanzentests erforderlich; für diese wurde ein Großteil der 90.63 Sekunden benötigt (die von `obvious_instance` und `obvious_non_instance` benötigte Zeit kann vernachlässigt werden, bzw. liegt – summiert – im Millisekundenbereich).

Bei der sog. *globalen Zurückroll-Zeit* handelt es sich um die Zeit, die nach durchgeführter Anfragebeantwortung benötigt wird, um die ABox wieder in den originalen Zustand zu bringen (Zurückrollen der Befehlshistorie, s. Kap. 6.2.2.3). Zwischen den einzelnen Anfragen wird auf dieses Zurückrollen der Befehlshistorie verzichtet. Stattdessen werden die *deterministischen Assertionen* beibehalten, s. Kap. 6.2.8. Da die deterministischen Assertionen logisch impliziert sind, muss der originale ABox-Zustand nicht zwingend wieder hergestellt werden. Dennoch ist die gemessene Zeit interessant, da sie verdeutlicht, dass das Zurückrollen einer Befehlshistorie mit mehreren hunderttausend bis einigen millionen Einträgen in MIDELORA eine zeitaufwendige Operation ist. Zurücksetzen der Suche zu weit oben im Suchbaum liegenden Knoten (mit niedrigen Verzweigungspunkt-IDs) ist in MIDELORA daher prinzipiell eine recht teure Operation. Dies ist eine wichtige Einsicht.

Die Einzelzeiten für die 14 Anfragen auf einer Universität sind folgende:

7.2 Evaluation von MiDeLoRA

	Q1	Q2	Q3	Q4	Q5	Q6	Q7
Tupel	4	0	6	34	719	7790	67
Zeit (Sek.)	0.04	7.056	0.028	0.076	0.052	0.584	0.04
	Q8	Q9	Q10	Q11	Q12	Q13	Q14
Tupel	7790	208	4	224	15	1	5916
Zeit (Sek.)	5.049	12.14	0.032	1.053	62.027	0.036	2.417

Die Zeiten sind nach Meinung des Autors akzeptabel. Im Gegensatz zum unvollständigen NRQL-Modus 1 kann **Q12** nun vollständig beantwortet werden, jedoch zum Preis einer um den Faktor $\frac{90.63}{9.09} = 9.97$ langsameren Gesamt-Beantwortungszeit.

7.2.3.1.1. Der LUBM mit einer Datenbank-ABox MiDeLoRA arbeitet sowohl auf transienten als auch auf persistenten ABoxen. Obwohl noch nicht alle Probleme bzgl. persistenter ABoxen in MiDeLoRA gelöst sind (s. Diskussion in Kap. 6.2.3.2), so ermöglicht die Architektur doch unmittelbar die Verwendung aller definierten Beweiser auf persistenten ABoxen. Dabei geschieht das Nachladen und Materialisieren der virtuellen Assertionen aus der Datenbank transparent, d.h., es muss kein zusätzlicher Code zum „Laden und Konvertieren der Datenbank in Hauptspeicher-Objektgraphen“ geschrieben werden. Wir hatten in Kap. 6 erläutert, wie mittels Iteratoren und entspr. Zugriffsmethoden diese Transparenz aus der Perspektive der Beweiser erreicht wird. Es handelt sich daher um eine *tiefe Integration* der Beweiser mit der Datenbank, die somit prinzipiell über ein einfaches „Laden und Konvertieren aus einer Datenbank“ anstatt dem „Laden und Konvertieren aus einer KRSS-Datei“ hinausgeht.

Wie in Kap. 6.4.6, S. 520 erläutert, führt momentan jedoch bereits der initiale ABox-Konsistenztest mangels einer definierten Speicherverwaltungsregel zu einer vollständigen Materialisierung aller (virtuellen) Datenbank-Assertionen. Da keine Speicherverwaltungsregel definiert wurde, werden diese ab dann nicht mehr dematerialisiert. Dieses würde jedoch erforderlich, sollte die Datenbank-ABox nicht mehr vollständig zu einem Zeitpunkt in den Hauptspeicher passen. Somit testen wir hier eigentlich ebenfalls nur ein einfaches „Laden einer ABox aus einer Datenbank“, gleichwohl die Architektur die technischen Voraussetzungen für darüber hinausgehende Nutzungsmöglichkeiten bietet (die momentan jedoch nicht implementiert sind).

Zum Test der Ladezeit einer DB-ABox wird zunächst eine LUBM-ABox aus einer KRSS-ABox-Datei in das MiDeLoRA-System geladen. Die KRSS instance- und related-Einträge in dieser Datei führen nun automatisch zur Bevölkerung der Datenbank. Starten wir nun den ABox-Erfüllbarkeitsbeweiser auf der persistenten DB-ABox, so führt das inkrementelle Anfordern der virtuellen Assertionen aus der Datenbank zum Absetzen von SQL-Abfragen. Die so erhaltenen

Assertionen werden als Substrat-Knoten und -Kanten mit entspr. Beschriftungen im Hauptspeicher materialisiert. Der Erfüllbarkeitstest benötigt auf diese Weise nun 16.013 Sekunden gegenüber 9.765 Sekunden, was einem Faktor von 1.64 entspricht. Da nach erfolgtem Einladen der Assertionen aus der Datenbank alle Substrat-Objekte materialisiert wurden, weichen die Zeiten für die nachfolgend gestellten LUBM-Anfragen nicht von den bereits gemessenen Zeiten ab.

7.2.3.2. Die DISK-Referenzanfragen

Für den LUBM hatten wir ermittelt, dass die Verwendung eines vollständigen BAA-Modus zu einer um ca. den Faktor 10 langsameren Anfragebeantwortung führt, wenn MIDeLoRA verwendet wird. Den gleichen Test wiederholen wir nun mit den DISK-Referenzanfragen.

Statt eines hybriden Karten-Substrates mit assoziierter RACERPRO-ABox verwenden wir nun eine räumliche MIDeLoRA-ABox (Instanz der Klasse MIDeLoRA-Explizite-Relationen-Karten-Substrat (MIDeLoRA-Explicit-Relations-Map-Substrate)). Wie bereits zuvor werden die thematischen Atome auf Konzept-Atome abgebildet. Für die RCC-Atome werden nun jedoch Rollen-Atome verwendet, da die RCC-Kanten als RCC-Rollen-Assertionen explizit gemacht wurden. Dies führt zu zusätzlichen Disjunktionen in den Anfragen, da beispielsweise das RCC-Atom (`?x ?y (:or :ntppi :tppi)`) nun in die Disjunktion (`(or (?*x ?*y :ntppi) (?*x ?*y :tppi))`) zweier Rollen-Atome umgeschrieben werden muss (Rollen-Atome mit disjunktiven Rollen sind momentan nicht implementiert). So hat die Anfrage **q6** nun 16 (komplexe) Disjunkte, da `:inside` für (`:or :tpp :ntpp`), `:flows-in` für (`:or :ec :po`), `:contains` für (`:or :tppi :ntppi`) und (`:or :touches :crosses`) für (`:or :ec :po`) steht. Die umgeschriebenen Anfragen sind nun also deutlich komplexer. Wir hätten jedoch auch virtuelle RCC-Atome verwenden können, da die ABox-Individuen ja geometrische Objekte sind, und auf RCC-Rollen-Assertionen in der ABox daher verzichten können. Aufgrund der vergleichsweise schlechten Performanz der virtuellen RCC-Atome, und da diese prinzipiell bereits in Kap. 7.1.2.2 evaluiert wurden, haben wir die Kanten hier explizit gemacht, damit primär die in MIDeLoRA durch beschreibungslogische Inferenz konsumierte Zeit gemessen wird. Natürlich müssen die `:inside-distance`- und `:area`-Atome etc. jedoch nach wie vor durch Inspektionsmethoden evaluiert werden; hier kann jetzt also die Räumlichkeit der ABox *direkt* genutzt werden.

Wir erwarten nun auch Antworten für **q3 (v1)**, **q4** und **q5 (v2)**, da wir den vollständigen Modus verwenden. Da zudem (`?y (or industrie gewerbe)`) in **q5 (v1)** zuvor nicht vollständig beantwortet werden konnte, erwarten wir nun *weniger* Ergebnistupel für **q5 (v1)** (da die Disjunktion innerhalb eines

7.2 Evaluation von MiDeLoRA

NAF-negierten Rumpfes in der Anfrage auftritt). Die vollständige Beantwortung von **Q5 (V2)** erfordert aufgrund der verwendeten universellen Rollenrestriktion ($\text{all ec (not (or industrie gewerbe))}$) zudem, dass die RCC-Rollen abgeschlossen werden (s. Kap. 3.1.5). Alternativ könnten wir MiDeLoRA nun jedoch auch in einem Modus betreiben, der dieses Abschließen überflüssig macht, indem der Beweiser so parametrisiert wird, dass zur Expansion der Assertion $i : \exists EC.(industrie \sqcup gewerbe)$ (die aufgrund der Reduktion des Instanzentests für i auf den ABox-Unerfüllbarkeitstests von $\mathcal{A} \cup \{i : \exists EC.(industrie \sqcup gewerbe)\}$ entsteht) die in Kap. 3.1.5 diskutierte *spezialisierte* $\exists$ -Regel verwendet wird. Da MiDeLoRA nur den $\mathcal{N}$ -, nicht jedoch den $\mathcal{Q}$ -Operator bietet, müssen wir in **Q4** zudem $(? * x \text{ (and wohngebiet (at-least 8 ppi gebaeude)))}$ gegen $(? * x \text{ (and wohngebiet (at-least 8 ppi top)))}$ ersetzen.

Die für Karte 1 erzeugte ABox enthält dann 2693 Individuen, 5936 Konzept-Assertionen (inkl. RCC-Rollenabschluss-Assertionen), und 1104 RCC-Rollen-Assertionen. Für Karte 1 wurden die folgenden Werte ermittelt:

Anzahl Individuen	2693
Lade-, Indizierungszeit und Zeit für ABox-Erfüllbarkeitstest	75.384 Sekunden
Initialisierungsanfrage	20.973 Sekunden
Summierte Beantwortungszeiten ($\sum_i \mathbf{g}_i$)	431.771 Sekunden
Angefallene Instanzen-Tests	51665
obvious_instance-Treffer	553 (1.07 %)
obvious_non_instance-Treffer	50659 (98.05 %)
Benötigte ABox-Unerfüllbarkeitstests	453 (0.88 %)
Globale Zurückroll-Zeit	4.3680 Sekunden

DISKQL-Anfrage	Karte 1	
	Anzahl Tupel	Zeit (Sek.)
Q1	76	0.196
Q2	4	0.616
Q3 (V1)	12	1.184
Q3 (V2)	12	1.064
Q4	5	13.405
Q5 (V1)	74	0.796
Q5 (V2)	68	215.546
Q6	1	21.225
Q7	69	0.384
Q8	102	177.355

Auf Karte 2 führen die DISK-Referenzanfragen momentan zu nicht akzeptablen Antwortzeiten für die Anfragen **Q4** und **Q8**; der initiale ABox-Erfüllbarkeitstest benötigt bereits 636 Sekunden. Die Anfragen **Q4**, **Q8** konnten innerhalb von 1000 Sekunden nicht vollständig beantwortet werden.

Die Nutzung von MiDeLoRA bedingt also einen Faktor von $\frac{431.77}{7.89} = 54.32$. Offensichtlich werden hier mehr echte ABox-Unerfüllbarkeitstests benötigt als für den LUBM (0.88 % gegenüber 0.02 %), was den deutlich schlechteren Faktor erklärt. Zudem sind die Anfragen aber in gewisser Weise auch komplexer als die LUBM-Anfragen, da nun beispielsweise **96** 16 komplexe Disjunkte enthält.

7.2.4. Fazit

MiDeLoRA hat den LUBM-Benchmark und somit den „Skalierbarkeitstest für Leitdomäne 2“ nach Meinung des Autors bestanden. Es gehört damit zu den schnellsten heutzutage (Stand: Sommer 2006) verfügbaren DL-Systemen. Für Leitdomäne 1 bzw. die DLMAPS-Anwendung müssen wir noch mit kleineren Karten vorlieb nehmen. Der Autor ist jedoch optimistisch, auch für Karte 2 ansprechende Zeiten erzielen zu können. MiDeLoRA ermöglicht die (im Sinne unserer Definitionen) vollständige ontologiebasierte Anfragebeantwortung mittels der MiDeLoRA-BAA in diesen Domänen.

Allerdings sind die erzielten Erfolge in gewisser Weise „zerbrechlich“ und nicht ohne Weiteres auf beliebige Wissensbasen, Anfragen oder beliebige IS-Domänen übertragbar. In den beiden Leitdomänen haben wir vielversprechende Anfangserfolge erzielt, die aber nur aufgrund der unvollständigen Verfahren möglich wurden. Sobald jedoch die unvollständigen Verfahren nicht anwendbar sind, werden teure ABox-Unerfüllbarkeitstests erforderlich. Diese können u.U. mehrere Sekunden oder gar hunderte von Sekunden dauern. Leider ist es oftmals purer Zufall bzw. „Glück“, ob die durch den initialen ABox-Erfüllbarkeitstest als Seiteneffekt akquirierten Individuums- und Kernmodelle während der Anfragebeantwortung zu obvious_instance- oder obvious_non_instance-Treffern führen. Selbst wenn die unvollständigen Verfahren – wie im Falle der DISK-Referenzanfragen – 99.12 % aller anfallenden ABox-Unerfüllbarkeitstests einsparen, so verursachen die verbleibenden 0.88 % bei großen ABoxen dennoch hunderte von ABox-Unerfüllbarkeitstests (für Karte 1: 453 Tests). Kann ein einzelner Unerfüllbarkeitstest nicht im Millisekundenbereich beantwortet werden, so ist performante Anfragebeantwortung nicht in jedem Fall möglich. Solche Zeiten jedoch zu *garantieren*, ist selbst für einfache ABoxen sehr schwierig.

Der ABox-Erfüllbarkeitstest bzw. Instanzentest in MiDeLoRA müsste noch um ein bis zwei Größenordnungen beschleunigt werden, um eine höhere Robustheit der Ergebnisse garantieren zu können. Vielversprechend erscheint die Verwendung des sog. *binären bzw. partitionsbasierten Instanzenretrievals* [Haarslev u. Möller 2002; 2004] anstatt des in MiDeLoRA verwendeten linearen Instanzenretrievals. Durch diese Technik kann u.U. eine große Anzahl von Unerfüllbarkeitstests eingespart werden. Dies konnte in dieser Arbeit leider nicht mehr untersucht werden.

7.3. Schlusswort und Ausblick

Die in dieser Arbeit entworfene und implementierte Rahmenarchitektur bietet pragmatische und innovative Lösungen für die drei Hauptproblemgruppen, die sich beim Bau ontologiebasierter ISe ergeben (vgl. auch [Schefe 1991]):

1. das Repräsentationsproblem,
2. das Anfragebeantwortungsproblem.
3. das Inferenzproblem.

Das Rahmenwerk verfügt über eine formale Semantik, sodass die Leistungen eines mit dem Rahmen implementierten Systems klar definiert sind. Das Rahmenwerk ist stark durch Beschreibungslogiken beeinflusst, ist jedoch bzgl. der Daten- bzw. Informationsrepräsentation flexibler, da es u.a. hybride, multimodale und räumliche Repräsentationsstrukturen sowie Repräsentationsstrukturen speziell zur Datenrepräsentation bietet. Zudem integriert und „subsumiert“ es Beschreibungslogiken und bietet mittels einer domänenspezifischen Sprache zur Konstruktion von Beschreibungslogikbeweisen eine hohe Erweiterbarkeit, Flexibilität und Konfigurierbarkeit. Wir haben die erzielte Flexibilität durch Angabe mehrere Ausprägungen in den unterschiedlichen Schichten (den obigen Problemgruppen 1., 2. und 3. entsprechend) belegt.

Sowohl mit dem Substrat-Datenmodell, als auch dem Anfragesprachenrahmenwerk und dem MIDELORA-Rahmenwerk haben wir Aspekte adressiert, die in Standard-DL-Systemen heutzutage noch nicht adäquat berücksichtigt werden, u.a.

- reichhaltige, virtuelle und persistente ABoxen,
- hybride Repräsentationen,
- unvollständige und zweiphasige (und somit ressourcenkonfigurierbare) Anfragebeantwortungsmodi,
- inkrementelle und nebenläufige Anfragebeantwortung,
- expressive und innovative Konstrukte für SEMANTIC WEB- bzw. OWL-Anfragesprachen (Negation as Failure, negierte Anfragerollen, Constraint-Atome, etc.),
- domänenspezifische Sprachen zur Konstruktion bzw. Konfiguration von Beweisen.

So ist beispielsweise auch nRQL nach Wissen des Autors nach wie vor die einzig existierende (und von Anwendern) genutzte ABox- bzw. SEMANTIC WEB-Anfragesprache, welche über derartige Merkmale verfügt. Die für die BAA bzw.

NRQL entwickelten Optimierungstechniken haben sich als unbedingt erforderlich herausgestellt.

Für die in NRQL realisierten unvollständigen und zweiphasigen Anfragebeantwortungsmodi stellt sich hingegen die Frage nach der „Halbwertszeit“: Tatsächlich erreicht RACERPRO – und somit auch NRQL im vollständigen Modus – inzwischen eine Performanz und ABox-Skalierbarkeit, die vorher den unvollständigen Modi vorbehalten war. Dies ist insbesondere für den LUBM der Fall. Ist der ganze Aufwand zur Realisierung unvollständiger und zweiphasiger Modi also überflüssig, sobald DL-Systeme schnell genug werden? Die Antwort auf diese Frage ist Jein: Auf der einen Seite wird es sicherlich immer ABoxen geben, für die die im DL-System implementierten (Retrieval-) Optimierungen nicht anwendbar sind, sodass die während der Anfragebeantwortung dann anfallenden ABox-Erfüllbarkeitstests Sekunden oder Minuten benötigen. Wie dargestellt, ist dies dann „tödlich“ für die vollständige Anfragebeantwortung auf halbwegs großen ABoxen. Die unvollständigen, zweiphasigen oder auch inkrementellen Anfragebeantwortungsmodi können dann Abhilfe bzw. Linderung schaffen. Dies ist insbesondere auch für MIDELORA der Fall, welches ja *inkrementelle Versionen* der zur Auswertung von Konzept- und Rollen-Atomen benötigten Basis-ABox-Retrieval-Funktionen (concept_instances, role_fillers etc.) bietet. Zum anderen wurde die Zweiphasigkeit und Inkrementalität *per se* als sinnvoll und wünschenswert für das SEMANTIC WEB-Retrieval dargestellt, da hier „Vollständigkeit der Antwortmenge“ prinzipiell nicht zu erwarten ist.

Die in diesem Kapitel durchgeführte Evaluation hat gezeigt, dass der Rahmen tatsächlich zur Realisierung (prototypischer) ontologiebasierter ISe für kleine bis mittelgroße Daten- bzw. Informationsbasen verwendet werden kann. Dabei erreichen die unvollständigen Modi noch einen hohen Performanzvorsprung (gemessene Faktoren 10 für den LUBM, und 55 für die DISK-Referenzanfragen); zudem ist der Hauptspeicherbedarf geringer. Gleichwohl z.B. Systeme wie RACERPRO hier inzwischen in der aktuellsten Version sehr viel schneller als MIDELORA sind, so schmälert dies dennoch nicht die Leistungen von MIDELORA oder die Notwendigkeit von MIDELORA für das Gesamt-rahmenwerk.

Die Architektur von MIDELORA kann als innovativ für DL-Systeme bezeichnet werden. Zu Beginn der Arbeiten an MIDELORA (im Jahre 2003) war nicht klar, ob mit einer derartigen Tableau-Repräsentation, der Idee der destruktiven Veränderungen und der Befehlshistorie überhaupt ein annähernd performantes DL-System realisiert werden könnte. Diese Arbeit hat gezeigt, dass der Aufwand zwar erheblich ist, ein effizientes DL-System mit diesen Techniken jedoch dennoch realisiert werden kann. Zumal hat die MIDELORA-Architektur langfristig unübersehbare Vorteile, die wir in Kap. 6.2.3, S. 430 ausführlich dargelegt und diskutiert haben.

7.3 Schlusswort und Ausblick

Nichtsdestotrotz sind noch weitergehende Entwicklungsarbeiten erforderlich: So sind die in MIDeLoRA implementierten Beweiser für DLs mit inversen Rollen weitgehend unoptimiert; auch die Implementation des $\mathcal{N}$ -Operators ist noch verbesserungswürdig. Aus Platzgründen konnten wir in diesem Kapitel zudem nicht jede von MIDeLoRA bereitgestellte DL evaluieren.

Aus der Perspektive des Autors wäre eine Weiterführung der begonnenen Arbeiten insbesondere unter den Gesichtspunkten *Realisierung von Persistenz und Transaktionalität* vielversprechend, zumal die benötigten Schnittstellen und Basis-Abstraktionen in MIDeLoRA prinzipiell bereits realisiert sind, und Anfangserfolge im Rahmen dieser Arbeit erzielt wurden.

Literaturverzeichnis

- [Abiteboul u. a. 1995] ABITEBOUL, S. ; HULL, R. ; VIANU, V.: *Foundations of Databases*. Addison-Wesley, 1995
- [Acciarri u. a. 2005] ACCIARRI, A. ; CALVANESE, D. ; DE GIACOMO, G. ; LEMBO, D. ; LENZERINI, M. ; PALMIERI, M. ; ROSATI, R.: *QUONTO: Querying ONtologies*. In: *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI 2005)*, 2005
- [Alonso u. a. 2004] ALONSO, G. ; CASATI, F. ; KUNO, H. ; MACHIRAJU, V.: *Web Services – Concepts, Architectures and Applications*. Springer-Verlag, 2004
- [Antoniou u. v. Harmelen 2004] ANTONIOU, G. ; VAN HARMELEN, F. : *A Semantic Web Primer*. The MIT Press, 2004
- [Baader 1996] BAADER, F.: Logik-basierte Wissensrepräsentation. In: *KI 3/96* (1996)
- [Baader 1999] BAADER, F.: Logic-Based Knowledge Representation. In: WOOLDRIDGE, M.J. (Hrsg.) ; VELOSO, M. (Hrsg.): *Artificial Intelligence Today, Recent Trends and Developments*, Springer-Verlag, 1999
- [Baader u. a. 2005] BAADER, F. ; BRANDT, S. ; LUTZ, C.: Pushing the $\mathcal{EL}$ Envelope. In: *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI 2005)*, 2005
- [Baader u. a. 2003a] BAADER, F. (Hrsg.) ; CALVANESE, D. (Hrsg.) ; MCGUINNESS, D. (Hrsg.) ; NARDI, D. (Hrsg.) ; PATEL-SCHNEIDER, P. (Hrsg.): *The Description Logic Handbook – Theory, Implementation and Applications*. Cambridge University Press, 2003
- [Baader u. a. 1994] BAADER, F. ; FRANCONI, E. ; HOLLUNDER, B. ; NEBEL, B. ; PROFITLICH, H.J.: An Empirical Analysis of Optimization Techniques for Terminological Representation Systems or: Making KRIS get a move on. In: *Applied Artificial Intelligence. Special Issue on Knowledge Base Management 4* (1994)
- [Baader u. Hanschke 1991a] BAADER, F. ; HANSCHKE, P.: A Scheme for Integrating Concrete Domains into Concept Languages. In: *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI 1991)*, 1991

- [Baader u. Hollunder 1991] BAADER, F. ; HOLLUNDER, B.: KRIS: Knowledge Representation and Inference System, System Description. In: *ACM SIGART Bulletin* 2 (1991)
- [Baader u. a. 2002] BAADER, F. ; HORROCKS, I. ; SATTLER, U.: Description Logics for the Semantic Web. In: *KI – Künstliche Intelligenz* 4 (2002)
- [Baader 1990] BAADER, F.: Terminological Cycles in KL-ONE-based Knowledge Representation Languages. In: *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI 1990)*, 1990
- [Baader 1991] BAADER, F.: Augmenting Concept Languages by Transitive Closure of Roles: An Alternative to Terminological Cycles. In: *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI 1991)*, 1991
- [Baader u. Hanschke 1991b] BAADER, F. ; HANSCHKE, P.: A Scheme for Integrating Concrete Domains into Concept Languages / Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI). 1991 (RR-91-10). Forschungsbericht
- [Baader u. a. 2003b] BAADER, F. ; HORROCKS, I. ; SATTLER, U.: Description Logics as Ontology Languages for the Semantic Web. In: HUTTER, D. (Hrsg.) ; STEPHAN, W. (Hrsg.): *Festschrift in honor of Jörg Siekmann*, Springer-Verlag, 2003
- [Baader u. Sattler 2000] BAADER, F. ; SATTLER, U.: Tableau Algorithms for Description Logics. In: *Proceedings of the 9th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2000)*, 2000
- [Baader u. Sattler 2001] BAADER, F. ; SATTLER, U.: An Overview of Tableau Algorithms for Description Logics. In: *Studia Logica* 69 (2001), Nr. 1
- [Baeza-Yates u. a. 1999] BAEZA-YATES, R. ; RIBEIRO-NETO, B.: *Modern Information Retrieval*. Addison-Wesley, 1999
- [Bartelme 1995] BARTELME, N.: *Geoinformatik – Modelle, Strukturen, Funktionen*. Springer-Verlag, 1995
- [Bass u. a. 1998] BASS, L. ; CLEMENTS, P. ; KAZMAN, R.: *Software Architecture in Practice*. Addison-Wesley, 1998
- [Bechhofer u. a. 2006a] BECHHOFFER, Sean ; VAN HARMELEN, F. ; HENDLER, J. ; HORROCKS, I. ; MCGUINNESS, D. ; PATEL-SCHNEIDER, P. ; STEIN, L.: *OWL Web Ontology Language Reference / World Wide Web Consortium*, 2006. Forschungsbericht

- [Bechhofer u. a. 2006b] BECHHOFFER, S. ; VAN HARMELEN, F. ; HENDLER, J. ; HORROCKS, I. ; MCGUINNESS, D. ; PATEL-SCHNEIDER, P. ; STEIN, L.: OWL Web Ontology Language Reference / World Wide Web Consortium, 2006. Forschungsbericht
- [Bechhofer u. a. 2003] BECHHOFFER, S. ; MÖLLER, R. ; CROWTHER, P.: The DIG Description Logic Interface. In: *Proceedings of the International Workshop on Description Logics (DL 2003)*, 2003
- [Berners-Lee u. a. 2001] BERNERS-LEE, T. ; HENDLER, J. ; LASSILA, O.: The Semantic Web. In: *Scientific American* (2001), Mai
- [Berners-Lee u. a. 2004] BERNERS-LEE, T. ; FIELDING, R. ; IRVINE, U.C. ; MASINTER, L.: *Uniform Resource Identifiers (URI): Generic Syntax*.
- [Bibel 1993] BIBEL, W.: *Wissensrepräsentation und Inferenz*. Vieweg & Sohn Verlagsgesellschaft, 1993
- [Bill u. Fritsch 1991] BILL, R. ; FRITSCH, D.: *Grundlagen der Geo-Informationssysteme*. Wichmann, 1991
- [Blackburn u. a. 2001] BLACKBURN, P. ; RIJKE, M. de ; VENEMA, Y.: *Modal Logic*. Cambridge Tracts in Theoretical Computer Science, No 53, 2001
- [Borgida u. a. 2003] BORGIDA, A. ; LENZERINI, M. ; ROSATI, R.: Description Logics for Databases. In: **[Baader u. a. 2003a]**, Kapitel 16
- [Borgida 1992] BORGIDA, A.: On the Relationship between Description Logics and Predicate Logic Queries / Rutgers University. New Brunswick (NJ, USA), 1992 (LCSR-TR-295-A). Forschungsbericht
- [Borgida 1995] BORGIDA, A.: Description Logics in Data Management. In: *IEEE Transactions on Knowledge and Data Engineering* 7 (1995), Nr. 5
- [Borgida u. Patel-Schneider 1994] BORGIDA, A. ; PATEL-SCHNEIDER, P.: A Semantics and Complete Algorithm for Subsumption in the CLASSIC Description Logic. In: *Journal of Artificial Intelligence Research* 1 (1994)
- [Brachman u. Levesque 1984] BRACHMAN, R. J. ; LEVESQUE, H. J.: The Tractability of Subsumption in Frame-Based Description Languages. In: *Proceedings of the 4th National Conference on Artificial Intelligence (AAAI 1984)*, 1984
- [Brachman u. Levesque 2004] BRACHMAN, R. J. ; LEVESQUE, H. J.: *Knowledge Representation and Reasoning*. Morgan Kaufmann Publishers, Inc., 2004
- [Brachman u. Schmolze 1985] BRACHMAN, R. J. ; SCHMOLZE, J. W.: An Overview of the KL-ONE Knowledge Representation System. In: *Cognitive Science* 9 (1985), Nr. 2

- [Brachman u. a. 1989] BRACHMAN, R. ; BORGIDA, A. ; MCGUINNESS, D. ; RESNICK, L.: *The CLASSIC Knowledge Representation System, or, KL-ONE: the Next Generation*. Preprints of the Workshop on Formal Aspects of Semantic Networks, 1989
- [Brickley u. Guha 2000] BRICKLEY, D. ; GUHA, R. V.: Resource Description Framework (RDF) Schema Specification 1.0 / World Wide Web Consortium, 2000. Forschungsbericht
- [Brill 1994] BRILL, D.: *Loom Reference Manual, Version 2.0*. Marina del Rey, 1994
- [de Bruijn u. a. 2005a] DE BRUIJN, J. ; FRANCONI, E. ; TESSARIS, S.: Logical Reconstruction of Normative RDF. In: *OWL: Experiences and Directions Workshop (OWLED 2005)*, 2005
- [de Bruijn u. a. 2005b] DE BRUIJN, J. ; FRANCONI, E. ; TESSARIS, S.: Logical Reconstruction of RDF and Ontology Languages. In: *Third Workshop on Principles and Practice of Semantic Web Reasoning*, 2005
- [Buchheit u. a. 1993a] BUCHHEIT, M. ; DONINI, F. ; SCHAEFER, A.: Decidable Reasoning in Terminological Knowledge Representation Systems. In: *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI 1993)*, 1993
- [Buchheit u. a. 1993b] BUCHHEIT, M. ; JEUSFELD, M. ; NUTT, W. ; STAUDT, M.: Subsumption between Queries to Object-Oriented Databases / Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI). Saarbrücken, 1993 (RR-93-44). Forschungsbericht
- [Buchheit u. a. 1994a] BUCHHEIT, M. ; JEUSFELD, M. ; NUTT, W. ; STAUDT, M.: Subsumption between Queries to Object-oriented Databases. In: *Information Systems* 19 (1994), Nr. 1
- [Buchheit u. a. 1994b] BUCHHEIT, M. ; KLEIN, R. ; NUTT, W.: Configuration as Model Construction: the Constructive Problem Solving Approach. In: *Proceedings of the 4th International Conference on Artificial Intelligence in Design*, 1994
- [Buneman 1997] BUNEMAN, P.: Semistructured Data. In: *Proceedings of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 1997)*, 1997
- [Calvanese u. a. 1997] CALVANESE, D. ; DE GIACOMO, G. ; LENZERINI, M.: Conjunctive Query Containment in Description Logics with n -ary Relations. In: *Proceedings of the International Workshop on Description Logics (DL 1997)*, 1997

- [Calvanese u. a. 1998a] CALVANESE, D. ; DE GIACOMO, G. ; LENZERINI, M. ; NARDI, D. ; ROSATI, R.: Knowledge Representation Approach to Information Integration. In: *Proceedings of the AAAI Workshop on AI and Information Integration*, 1998
- [Calvanese u. a. 1998b] CALVANESE, D. ; GIACOMO, G. ; LENZERINI, M. ; NARDI, D. ; ROSATI, R.: Information Integration: Conceptual Modeling and Reasoning Support. In: *Conference on Cooperative Information Systems*, 1998
- [CLSQL 2006] CLSQL: CLSQL. <http://clsql.b9.com>, 2006
- [Cohn 1993] COHN, A. G.: Modal and Non Modal Qualitative Spatial Logics. In: *Proceedings of the Workshop on Spatial and Temporal Reasoning (IJCAI 1993)*, 1993
- [Cohn u. a. 1997] COHN, A. G. ; BENNETT, B. ; GOODAY, J. M. ; GOTTS, N.: RCC: A Calculus for Region based Qualitative Spatial Reasoning. In: *GeoInformatica* 1 (1997)
- [Cordes u. a. 1992] CORDES, R. ; KRUSE, R. ; LANGENDÖRFER, H. ; RUST, H.: *Prolog – Eine methodisch Einführung*. 3. Auflage, Vieweg & Sohn Verlagsgesellschaft, 1992
- [Cunis u. a. 1991] CUNIS, R. (Hrsg.) ; GÜNTER, A. (Hrsg.) ; STRECKER, Helmut (Hrsg.): *Das PLAKON-Buch, Ein Expertensystemkern für Planungs- und Konfigurationsaufgaben in technischen Domänen*. Springer-Verlag, 1991
- [Ding u. Haarslev 2005] DING, Y. ; HAARSLEV, V.: Towards Efficient Reasoning for Description Logics with Inverse Roles. In: *Proceedings of the International Workshop on Description Logics (DL 2005)*, 2005
- [Donini u. a. 1992] DONINI, F. ; LENZERINI, M. ; NARDI, D. ; NUTT, W. ; SCHAEFER, A.: Adding Epistemic Operators to Concept Languages. In: *Proceedings of the 3rd International Conference on the Principles of Knowledge Representation and Reasoning (KR 1992)*, 1992
- [Dudenverlag 1993] DUDENVERLAG (Hrsg.): *Duden: Informatik*. 2. Auflage, Dudenverlag, 1993
- [Dudenverlag 1994] DUDENVERLAG (Hrsg.): *Duden: Mathematik*. 5. Auflage, Dudenverlag, 1994
- [Ebbinghaus 1994] EBBINGHAUS, H.-D.: *Einführung in die Mengenlehre*. 3. Auflage, BI-Wissenschaftsverlag, 1994
- [Ebbinghaus u. Flum 1999] EBBINGHAUS, H.-D. ; FLUM, J.: *Finite Model Theory*. 2. Auflage, Springer-Verlag, 1999

- [Ebbinghaus u. a. 1996] EBBINGHAUS, H.-D. ; FLUM, J. ; THOMAS, W.: *Einführung in die mathematische Logik*. 4. Auflage, Spektrum Akademischer Verlag, 1996
- [Ferber 2003] FERBER, R.: *Information Retrieval*. dpunkt.verlag, 2003
- [Ferber 2006] FERBER, R.: *Information Retrieval*. <http://information-retrieval.de/irb/ir.html>, 2006
- [Fikes u. a. 2003] FIKES, R. ; HAYES, P. ; HORROCKS, I.: OWL-QL - A Language for Deductive Query Answering on the Semantic Web / Knowledge Systems Lab, Stanford University, CA, USA. 2003 (KSL-03-14). Forschungsbericht
- [Fikes u. a. 2005] FIKES, R. ; HAYES, P. ; HORROCKS, I.: OWL-QL: A Language for Deductive Query Answering on the Semantic Web. In: *Journal of Web Semantics* 2 (2005), Nr. 1
- [Fikes u. McGuinness 2001] FIKES, R. ; MCGUINNESS, D.: An Axiomatic Semantics for RDF, RDF-S, and DAML+OIL / World Wide Web Consortium, 2001. Forschungsbericht
- [Fowler 2004] FOWLER, M.: *UML Distilled – A Brief Guide to the Standard Object Modeling Language*. 3. Auflage, Addison-Wesley, 2004
- [Franconi u. Ng 2000] FRANCONI, E. ; NG, G.: The i.com tool for Intelligent Conceptual Modeling. In: *Proceedings of the 7th International Workshop on Knowledge Representation meets Databases (KRDB 2000)*, 2000
- [Franz Inc. 1994] FRANZ INC.: *Common Lisp Interface Manager (CLIM) 2.0: User Guide*, 1994
- [Freie und Hansestadt Hamburg 1998] FREIE UND HANSESTADT HAMBURG: Promotionsordnung der Technischen Universität Hamburg-Harburg. In: *Amtlicher Anzeiger der Freien und Hansestadt Hamburg* 85 (1998), Juli
- [Galinski u. a. 2005] GALINSKI, J. ; KAYA, A. ; MÖLLER, R.: Development of a Server to Support the Formal Semantic Web Query Language OWL-QL. In: *Proceedings of the International Workshop on Description Logics (DL 2005)*, 2005
- [Gamma u. a. 1996] GAMMA, E. ; HELM, R. ; JOHNSON, R. ; VLISSIDES, J.: *Entwurfsmuster*. Addison-Wesley, 1996
- [Genesereth u. Nilsson 1987] GENESERETH, M. R. ; NILSSON, N. J.: *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers, Inc., 1987

- [Genesereth u. Fikes 1992] GENESERETH, M. ; FIKES, R.: Knowledge Interchange Format, Version 3.0 Reference Manual / Stanford University, 1992. Forschungsbericht.
- [Gennari u. a. 2003] GENNARI, J. ; MUSEN, M. ; FERGERTON, R. ; GROSSO, W. ; CRUBZY, M. ; ERIKSSON, H. ; NOY, N. ; TU, S.: The Evolution of Protégé: An Environment for Knowledge-Based Systems Development. In: *International Journal of Human-Computer Studies* 58 (2003), Januar, Nr. 1
- [Goos 1997] GOOS, G.: *Vorlesungen über Informatik – Band 1: Grundlagen und funktionales Programmieren*. 2. Auflage, Springer-Verlag, 1997
- [Görz u. a. 2003] GÖRZ, G. (Hrsg.) ; ROLLINGER, C.-R. (Hrsg.) ; SCHNEEBERGER, J. (Hrsg.): *Handbuch der Künstlichen Intelligenz*. 3. Auflage, Oldenburg Wissenschaftsverlag, 2003
- [Grädel 1999] GRÄDEL, E.: Why are Modal Logics so Robustly Decidable? In: *Bulletin of the European Association for Theoretical Computer Science* 68 (1999)
- [Graham 1994] GRAHAM, P.: *On Lisp*. Prentice-Hall, 1994
- [Graham 1996] GRAHAM, P.: *ANSI Common Lisp*. Prentice-Hall, 1996
- [Grigni u. a. 1995] GRIGNI, M. ; PAPADIAS, D. ; PAPADIMITRIOU, C.: Topological Inference. In: *14th International Joint Conference on Artificial Intelligence*, 1995
- [Gruber 1993a] GRUBER, T. R.: A Translation Approach to Portable Ontologies. In: *Knowledge Acquisition* 5 (1993), Nr. 2
- [Gruber 1993b] GRUBER, T. R.: Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In: *Formal Ontology in Conceptual Analysis and Knowledge Representation*. Kluwer Academic Publishers, 1993
- [Guarion 1998] GUARION, N.: Formal Ontology and Information Systems. In: *Proceedings Formal Ontology in Information Systems (FOIS 1998)*. ISO Press, 1998
- [Guo u. a. 2003] GUO, Y. ; HEFLIN, J. ; PAN, Z.: Benchmarking DAML+OIL Repositories. In: *Proceedings of the Second International Semantic Web Conference (ISWC 2003)*, 2003
- [Guo u. a. 2004a] GUO, Y. ; PAN, Z. ; HEFLIN, J.: An Evaluation of Knowledge Base Systems for Large OWL Datasets. In: *Proceedings of the Third International Semantic Web Conference (ISWC 2004)*, 2004

- [Guo u. a. 2004b] GUO, Y. ; PAN, Z. ; HEFLIN, J.: An Evaluation of Knowledge Base Systems for Large OWL Datasets / CSE Department, Lehigh University. 2004 (LU-CSE-04-012). Forschungsbericht
- [Haarslev u. Möller 2000] HAARSLEV, V. ; MÖLLER, R.: Consistency Testing: The RACE Experience. In: *Proceedings of the International Conference, Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2000)*, 2000
- [Haarslev u. Möller 2003] HAARSLEV, V. ; MÖLLER, R.: Racer: A Core Inference Engine for the Semantic Web. In: *Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON 2003)*, 2003
- [Haarslev u. Möller 2004] HAARSLEV, V. ; MÖLLER, R.: Optimization Techniques for Retrieving Resources Described in OWL/RDF Documents: First Results. In: *Ninth International Conference on the Principles of Knowledge Representation and Reasoning (KR 2004)*, 2004
- [Haarslev u. a. 2004a] HAARSLEV, V. ; MÖLLER, R. ; WESSEL, M.: Querying the Semantic Web with Racer + nRQL. In: *Proceedings of the KI 2004 Workshop on Applications of Description Logics (ADL 2004)*, 2004
- [Haarslev u. a. 2005] HAARSLEV, V. ; MÖLLER, R. ; WESSEL, M.: Description Logic Inference Technology: Lessons Learned in the Trenches. In: *Proceedings of the International Workshop on Description Logics (DL 2005)*, 2005
- [Haarslev u. a. 1998] HAARSLEV, V. ; LUTZ, C. ; MÖLLER, R.: Foundations of Spatioterminological Reasoning with Description Logics. In: *Proceedings of the 6th International Conference on the Principles of Knowledge Representation and Reasoning (KR 1998)*, 1998
- [Haarslev u. a. 1999] HAARSLEV, V. ; LUTZ, C. ; MÖLLER, R.: A Description Logic with Concrete Domains and Role-Forming Predicates. In: *Journal of Logic and Computation* 9 (1999), Nr. 3
- [Haarslev u. Möller 2001a] HAARSLEV, V. ; MÖLLER, R.: High Performance Reasoning with Very Large Knowledge Bases: A Practical Case Study. In: *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*, 2001
- [Haarslev u. Möller 2001b] HAARSLEV, V. ; MÖLLER, R.: Optimizing Reasoning in Description Logics with Qualified Number Restrictions. In: *Proceedings of the International Workshop on Description Logic (DL 2001)*, 2001
- [Haarslev u. Möller 2001c] HAARSLEV, V. ; MÖLLER, R.: RACER System Description. In: *Automated Reasoning: First International Joint Conference (IJCAR 2001)*, 2001

- [Haarslev u. Möller 2002] HAARSLEV, V. ; MÖLLER, R.: *Optimization Strategies for Instance Retrieval*. In: *Proceedings of the International Workshop on Description Logics (DL 2002)*, 2002
- [Haarslev u. a. 2004b] HAARSLEV, V. ; MÖLLER, R. ; VAN DER STRAETEN, R. ; WESSEL, M.: *Extended Query Facilities for Racer and an Application to Software-Engineering Problems*. In: *Proceedings of the International Workshop on Description Logics 2004 (DL 2004)*, 2004
- [Haarslev u. a. 2001a] HAARSLEV, V. ; MÖLLER, R. ; TURHAN, A.-Y.: *Exploiting Pseudo Models for TBox and ABox Reasoning in Expressive Description Logics*. In: *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR 2001)*, 2001
- [Haarslev u. a. 2001b] HAARSLEV, V. ; MÖLLER, R. ; WESSEL, M.: *The Description Logic $\mathcal{ALCNH}_{R+}$ Extended with Concrete Domains: A Practically Motivated Approach*. In: *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR 2001)*, 2001
- [Freie und Hansestadt Hamburg 1992] HANSESTADT HAMBURG, Landesbetrieb Geoinformation und Vermessung: *Beschreibung der Datenschnittstelle für Datenerlieferungen der Digitalen Stadtkarte (DISK)*, 1992
- [Freie und Hansestadt Hamburg 1996] HANSESTADT HAMBURG, Landesbetrieb Geoinformation und Vermessung: *Digitale Stadtkarte von Hamburg (DISK) und zugeordnete Daten: Verzeichnis der Ebenen und Schlüssel*, 1996
- [Harel 1984] HAREL, David: *Dynamic Logic*. In: GABBAY, D. M. (Hrsg.) ; GUENTHNER, F. (Hrsg.): *Handbook of Philosophical Logic* Bd. II. D. Reidel Publishing Company, 1984
- [Harel u. a. 2000] HAREL, D. ; KOZEN, D. ; TIURYN, J.: *Dynamic Logic*. The MIT Press, 2000
- [Hayes 1979] HAYES, P. J.: *The Logic of Frames*. In: METZING, D. (Hrsg.): *Frame Conceptions and Text Understanding*. Walter de Gruyter and Co., 1979
- [Hayes 2004] HAYES, P.: *RDF Semantics / World Wide Web Consortium*, 2004. Forschungsbericht
- [Henriksen u. a. 1995] HENRIKSEN, J.G. ; JENSEN, J. ; JØRGENSEN, M. ; KLARLUND, N. ; PAIGE, B. ; RAUHE, T. ; SANDHOLM, A.: *Mona: Monadic Second-order Logic in Practice*. In: *Tools and Algorithms for the Construction and Analysis of Systems, First International Workshop (TACAS 1995)*, 1995
- [Herman 2004] HERMAN, I.: *Questions (and Answers) on the Semantic Web (Presentation)*. <http://www.w3.org/2005/Talks/1007-Munich-IH/>, 2004

- [Hollunder u. Nutt 1990] HOLLUNDER, B. ; NUTT, W.: Subsumption Algorithms for Concept Languages / Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI). Kaiserslautern, 1990 (RR-90-04). Forschungsbericht
- [Horrocks u. a. 2004a] HORROCKS, I. ; PATEL-SCHNEIDER, P. F. ; BOLEY, H. ; TABET, S. ; GROSOFF, B. ; DEAN, M.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML / World Wide Web Consortium, 2004. Forschungsbericht
- [Horrocks 1997a] HORROCKS, I.: Optimisation Techniques for Expressive Description Logics / University of Manchester, Department of Computer Science. 1997 (UMCS-97-2-1). Forschungsbericht
- [Horrocks 1997b] HORROCKS, I.: *Optimising Tableaux Decision Procedures for Description Logics* / University of Manchester, Department of Computer Science, Dissertation, 1997
- [Horrocks 2003b] HORROCKS, I.: Implementation and Optimization Techniques. In: **[Baader u. a. 2003a]**, Kapitel 6
- [Horrocks u. a. 2004b] HORROCKS, I. ; LI, L. ; TURI, D. ; BECHHOFFER, S.: The Instance Store: DL Reasoning with Large Numbers of Individuals. In: *Proceedings of the International Workshop on Description Logics (DL 2004)*, 2004
- [Horrocks u. Patel-Schneider 1999] HORROCKS, I. ; PATEL-SCHNEIDER, P.: Optimizing Description Logic Subsumption. In: *Journal of Logic and Computation* 9 (1999), Nr. 3
- [Horrocks u. Patel-Schneider 2003] HORROCKS, I. ; PATEL-SCHNEIDER, P.: Reducing OWL Entailment to Description Logic Satisfiability. In: *Proceedings of the 2003 International Semantic Web Conference (ISWC 2003)*, 2003
- [Horrocks u. a. 2003] HORROCKS, I. ; PATEL-SCHNEIDER, P. ; VAN HARMELEN, F. : From SHIQ and RDF to OWL: The Making of a Web Ontology Language. In: *Journal of Web Semantics* 1 (2003), Nr. 1
- [Horrocks u. a. 2000a] HORROCKS, Ian ; SATTLER, U. ; TESSARIS, S. ; TOBIES, S.: How to decide Query Containment under Constraints using a Description Logic. In: *Proceedings of the 7th International Conference on Logic for Programming and Automated Reasoning (LPAR 2000)*, 2000
- [Horrocks u. a. 1999a] HORROCKS, I. ; SATTLER, U. ; TESSARIS, S. ; TOBIES, S.: Query Containment Using a DLR ABox / RWTH Aachen, 1999 (LTCS-Report 99-15). Forschungsbericht

- [Horrocks u. a. 1999b] HORROCKS, I. ; SATTLER, U. ; TOBIES, S.: Practical Reasoning for Expressive Description Logics. In: *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR 1999)*, 1999
- [Horrocks u. a. 2000b] HORROCKS, I. ; SATTLER, U. ; TOBIES, S.: Practical Reasoning for Very Expressive Description Logics. In: *Journal of the Interest Group in Pure and Applied Logic* 8 (2000), Nr. 3
- [Horrocks u. Tessaris 2002] HORROCKS, I. ; TESSARIS, S.: Querying the Semantic Web: a Formal Approach. In: *Proceedings of the 2002 International Semantic Web Conference (ISWC 2002)*, 2002
- [Hughes u. Cresswell 1968] HUGHES, G. ; CRESSWELL, M.: *An Introduction to Modal Logic*. Methuen, 1968
- [Hustadt u. a. 2004] HUSTADT, U. ; MOTIK, B. ; SATTLER, U.: Reducing *SHIQ*⁻ Description Logic to Disjunctive Datalog Programs. In: *Proceedings of the 9th International Conference on Principles of Knowledge Representation and Reasoning (KR 2004)*, 2004
- [KAON 2007] KAON: *KAON - The Karlsruhe Ontology and Semantic Web Tool Suite*. <http://kaon.semanticweb.org>, 2007
- [Kaplunova u. a. 2006] KAPLUNOVA, A. ; KAYA, A. ; MÖLLER, R.: Experiences with Load Balancing and Caching for Semantic Web Applications. In: *Proceedings of the International Workshop on Description Logics (DL 2006)*, 2006
- [Karabaev u. Lutz 2004] KARABAEV, E. ; LUTZ, C.: Mona as a DL Reasoner. In: *Proceedings of the 2004 International Workshop on Description Logics (DL 2004)*, 2004
- [Karvounarakis u. a. 2002] KARVOUNARAKIS, G. ; ALEXAKI, S. ; CHRISTOPHIDES, V. ; PLEXOUSAKIS, D. ; SCHOLL, M.: RQL: A Declarative Query Language for RDF. In: *The Eleventh International World Wide Web Conference (WWW 2002)*, 2002
- [Keene 1989] KEENE, S. E.: *Object-Oriented Programming in Common Lisp*. Addison-Wesley, 1989
- [Kemper u. Eickler 2004] KEMPER, A. ; EICKLER, A.: *Datenbanksysteme – Eine Einführung*. 5. Auflage, Oldenbourg Wissenschaftsverlag, 2004
- [Kiczales u. a. 1991] KICZALES, G. ; DES RIVIÈRES, J. ; BOBROW, D.G.: *The Art of the Metaobject Protocol*. MIT Press, 1991

- [Kiczales 1993] KICZALES, G.: Traces (A Cut at the Make Isn't Generic Problem). In: *Proceedings of the 2nd International Symposium on Object Technologies for Advanced Software (ISOTAS 1993)*, 1993
- [Kokkelink u. Schwänzl] KOKKELINK, S. ; SCHWÄNZL, R.: *Expressing Qualified Dublin Core in RDF / XML*. <http://dublincore.org/documents/dcq-rdf-xml/>,
- [Kumar 1992] KUMAR, V.: Algorithms for Constraint-Satisfaction Problems: A Survey. In: *AI Magazine* 13 (1992), Nr. 1
- [Kutz u. a. 2004] KUTZ, O. ; LUTZ, C. ; WOLTER, F. ; ZAKHARYASCHEV, M.: $\mathcal{E}$ -connections of Abstract Description Systems. In: *Artificial Intelligence* 156 (2004), Nr. 1
- [Lassila u. McGuinness 2001] LASSILA, O. ; MCGUINNESS, D.: The Role of Frame-Based Representation on the Semantic Web. In: *Electronic Transactions on Artificial Intelligence (ETAI)* 6 (2001), Nr. 5
- [Lenat 1995] LENAT, D. B.: CYC: A Large-Scale Investment in Knowledge Infrastructure. In: *Communications of the ACM* 38 (1995), Nr. 11
- [Lenat u. Guha 1990] LENAT, D. B. ; GUHA, R. V.: *Building Large Knowledge-Based Systems*. Addison-Wesley, 1990
- [Lockemann u. Schmidt 1987] LOCKEMANN, P. C. (Hrsg.) ; SCHMIDT, J. W. (Hrsg.): *Datenbankhandbuch*. Springer-Verlag, 1987
- [Lutz 1998] LUTZ, C.: *Repräsentation topologischer Information in Beschreibungslogiken* / Universität Hamburg, Fachbereich Informatik, Diplomarbeit, 1998
- [Lutz u. Wolter 2004] LUTZ, C. ; WOLTER, F.: Modal Logics of Topological Relations. In: *Proceedings of Advances in Modal Logics 2004*, 2004
- [Lutz u. Wolter 2006] LUTZ, C. ; WOLTER, F.: Modal Logics of Topological Relations. In: *Logical Methods in Computer Science* 2 (2006), Nr. 2
- [Lutz 1999] LUTZ, C.: Complexity of Terminological Reasoning Revisited. In: *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR 1999)*, 1999
- [Lutz 2001] LUTZ, C.: NEXPTIME-complete Description Logics with Concrete Domains. In: *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR 2001)*, 2001

- [MacGregor 1991] MACGREGOR, R.M.: The Evolving Technology of Classification-based Knowledge Representation Systems, 1991
- [Manola u. Miller 2004] MANOLA, F. ; MILLER, E.: RDF Primer / World Wide Web Consortium, 2004. Forschungsbericht
- [McBride 2002] MCBRIDE, B.: Jena: A Semantic Web Toolkit. In: *IEEE Internet Computing* 6 (2002), Nr. 6
- [McGuinness u. v. Harmelen 2004] MCGUINNESS, D. ; VAN HARMELEN, F. : OWL Web Ontology Language Overview / World Wide Web Consortium, 2004. Forschungsbericht
- [McGuinness 1996] MCGUINNESS, D.: *Explaining Reasoning in Description Logics*, Department of Computer Science, Rutgers University, Dissertation, 1996
- [McGuinness 2003] MCGUINNESS, D.: Configuration. In: **[Baader u. a. 2003a]**, Kapitel 12
- [McKay 1991] MCKAY, S.: CLIM: The Common Lisp Interface Manager. In: *Communications of the ACM* 34 (1991), Nr. 9
- [Meyer 1990] MEYER, B.: *Objektorientierte Softwareentwicklung*. Carl Hanser-Verlag, 1990
- [Meyer 1994a] MEYER, B.: Pictorial Deduction in Spatial Information Systems. In: *1994 IEEE Symposium on Visual Languages (VL 1994)*, 1994
- [Meyer 1994b] MEYER, B.: *Visuelle Logiksprachen zur Behandlung räumlicher Information* / Fernuniversität Hagen, Dissertation, 1994.
- [Miles u. Brickley 2005] MILES, A. ; BRICKLEY, D.: SKOS Core Guide / World Wide Web Consortium, 2005. Forschungsbericht
- [Ming u. a. 2005] MING, C. ; HAARSLEV, V. ; YUE, J.: LAS: Extending Racer by a Large ABox Store. In: *Proceedings of the International Workshop on Description Logics (DL 2005)*, 2005
- [Minsky 1974] MINSKY, M.: A Framework for Representing Knowledge / Massachusetts Institute of Technology. Massachusetts Institute of Technology, 1974. Forschungsbericht
- [Möller u. Haarslev 2003] MÖLLER, R. ; HAARSLEV, V.: Description Logic Systems. In: **[Baader u. a. 2003a]**, Kapitel 8

- [Möller u. a. 2006a] MÖLLER, R. ; HAARSLEV, V. ; WESSEL, M.: On the Scalability of Description Logic Instance Retrieval. In: *Proceedings of the International Workshop on Description Logics (DL 2006)*, 2006
- [Möller u. a. 2006b] MÖLLER, R. ; HAARSLEV, V. ; WESSEL, M.: On the Scalability of Description Logic Instance Retrieval. In: *29. Deutsche Jahrestagung für Künstliche Intelligenz (KI 2006)*, 2006
- [Musen 1998] MUSEN, M.: Modern Architectures for Intelligent Systems: Reusable Ontologies and Problem-Solving Methods. In: *Journal of the American Medical Informatics Association, Annual Symposium Special Issue*, 1998
- [MySQL 2006] MySQL: *MySQL*. <http://www.mysql.de>, 2006
- [Nebel 1991] NEBEL, B.: Terminological Cycles: Semantics and Computational Properties. In: SOWA, J. F. (Hrsg.): *Principles of Semantic Networks*. Morgan Kaufmann, 1991
- [Nerode u. Shore 1993] NERODE, A. ; SHORE, R. A.: *Logic for Applications*. Springer-Verlag, 1993
- [Newell 1982] NEWELL, A.: The Knowledge Level. In: *Artificial Intelligence 1* (1982), Nr. 18
- [Newell u. Simon 1976] NEWELL, A. ; SIMON, H. A.: Computer Science as Empirically Inquiry: Symbols and Search. In: *Communications of the ACM 3* (1976), Nr. 19
- [Norvig 1992] NORVIG, P.: *Artificial Intelligence Programming: Case Studies in Common Lisp*. Morgan Kaufmann, 1992
- [Oestereich 2005] OESTEREICH, B.: *Analyse und Design mit UML 2 - Objektorientierte Softwareentwicklung*. 7. Auflage, Oldenbourg Wissenschaftsverlag, 2005
- [OMG 2004] OMG: UML 2.0 Superstructure / Object Management Group, Framingham, MA, USA, 2004. Forschungsbericht
- [Page 1991] PAGE, B.: *Diskrete Simulation: Eine Einführung mit Modula-2*. Springer-Verlag, 1991
- [Palmer 1978] PALMER, S.: Fundamental Aspects of Cognitive Representations. In: ROSCH, E. (Hrsg.) ; LLOYD, B. (Hrsg.): *Cognition and Categorization*. Lawrence Erlbaum Associates, Publishers, 1978
- [Pan u. Horrocks 2003a] PAN, J. ; HORROCKS, I.: RDFS(FA): A DL-ised Sublanguage of RDFS. In: *Proceedings of the International Workshop on Description Logic (DL 2003)*, 2003

- [Pan u. Horrocks 2003b] PAN, J. ; HORROCKS, I.: RDFS(FA) and RDF MT: Two Semantics for RDFS. In: *Proceedings of the 2nd International Semantic Web Conference (ISWC 2003)*, 2003
- [Pan u. Heflin 2003] PAN, Z. ; HEFLIN, J.: DLDB: Extending Relational Databases to Support Semantic Web Queries. In: *Proceedings of the Workshop on Practical and Scaleable Semantic Web Systems (ISWC 2003)*, 2003
- [Patel-Schneider 1998] PATEL-SCHNEIDER, P.: DLP System Description. In: *Proceedings of the International Workshop on Description Logics (DL 1998)*, 1998
- [Patel-Schneider u. a. 2004] PATEL-SCHNEIDER, P. ; HAYES, P. ; HORROCKS, I.: OWL Web Ontology Language Semantics and Abstract Syntax / World Wide Web Consortium, 2004. Forschungsbericht
- [Patel-Schneider u. Swartout 1993] PATEL-SCHNEIDER, P. ; SWARTOUT, B.: Description-Logic Knowledge Representation System Specification from the KRSS Group of the ARPA Knowledge Sharing Effort / AI Principles Research Department, AT&T Bell Laboratories, 1993. Forschungsbericht.
- [Powers 2003] POWERS, S.: *Practical RDF*. O'Reilly & Associates, Inc., 2003
- [Prud'hommeaux u. Seaborne 2006] PRUD'HOMMEAUX, E. ; SEABORNE, A.: SPARQL Query Language for RDF / World Wide Web Consortium, 2006. Forschungsbericht
- [Quillian 1968] QUILLIAN, M. R.: Semantic Memory. In: MINSKY, M. (Hrsg.): *Semantic Information Processing*. The MIT Press, 1968,
- [Racer Systems 2006] RACER SYSTEMS GMBH & CO. KG: RacerPro User's Guide 1.9.0 / Racer Systems GmbH & Co. KG, <http://www.racer-systems.com/products/racerpro/users-guide-1-9.pdf>, 2006. Systemhandbuch
- [Randell u. a. 1992] RANDELL, D. A. ; CUI, Z. ; COHN, A. G.: A Spatial Logic based on Regions and Connections. In: *Principles of Knowledge Representation and Reasoning (KR 1992)*, 1992
- [Rechenberg u. Pomberger 2006] RECHENBERG, P. (Hrsg.) ; POMBERGER, G. (Hrsg.): *Informatik-Handbuch*. Carl Hanser-Verlag, 2006
- [Reiter 1984] REITER, R.: Towards a Logical Reconstruction of Relational Database Theory. In: *On Conceptual Modeling: Perspectives from Artificial Intelligence Databases and Programming Languages*, 1984

- [Riazanov u. Voronkov 2002] RIAZANOV, A. ; VORONKOV, A.: The Design and Implementation of VAMPIRE. In: *AI Communications* 15 (2002), Nr. 2
- [Rich u. Knight 1991] RICH, E. ; KNIGHT, K.: *Artificial Intelligence*. 2. Auflage, McGraw-Hill, Inc., 1991
- [Rizk 2005] RIZK, R.: *A Study on Query Completeness and Answering Time In Knowledgebases and Databases* / TU Hamburg-Harburg, Studienarbeit, 2005
- [Russel u. Norvig 2003] RUSSEL, S. ; NORVIG, P.: *Artificial Intelligence – A Modern Approach*. Prentice Hall, 2003
- [Sattler 1996b] SATTLER, U.: A Concept Language Extended with Different Kinds of Transitive Roles. In: *Proceedings of the 20th German Annual Conference on Artificial Intelligence (KI 1996)*, 1996
- [Scheffe 1991] SCHEFFE, P.: *Künstliche Intelligenz – Überblick und Grundlagen*. 2. Auflage, BI-Wissenschaftsverlag, 1991
- [Schild 1991] SCHILD, K.: A Correspondence Theory for Terminological Logics: Preliminary Report. In: *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI 1991)*, 1991
- [Schmidt-Schauß 1989] SCHMIDT-SCHAUSS, M.: Subsumption in KL-ONE is Undecidable. In: *Proceedings of the 1st International Conference on the Principles of Knowledge Representation and Reasoning (KR 1989)*, 1989
- [Schmidt-Schauß u. Smolka 1991] SCHMIDT-SCHAUSS, M. ; SMOLKA, G.: Attributive Concept Descriptions with Complements. In: *Artificial Intelligence* 48 (1991), Nr. 1
- [Schöning 1992] SCHÖNING, U.: *Logik für Informatiker*. 3. Auflage, BI-Wissenschaftsverlag, 1992
- [Sedgewick 1988] SEDGEWICK, R.: *Algorithms*. 2. Auflage, Addison-Wesley, 1988
- [Shalloway u. Trott 2006] SHALLOWAY, A. ; TROTT, J.: *Design Patterns Explained: A New Perspective on Object-Oriented Design*. Addison-Wesley, 2006
- [Siedersleben 2004] SIEDERSLEBEN, J.: *Moderne Softwarearchitektur*. dpunkt.verlag, 2004
- [Singh u. Huhns 2006] SINGH, M. P. ; HUHS, M. N.: *Service-Oriented Computing – Semantics, Processes, Agents*. John Wiley & Sons Ltd., 2006

- [Sirin u. Parsia 2006] SIRIN, E. ; PARSIA, B.: Pellet System Description. In: *Proceedings of the International Workshop on Description Logics (DL 2006)*, 2006
- [Sirin u. a. 2006] SIRIN, E. ; PARSIA, B. ; GRAU, B. ; KALYANPUR, A. ; KATZ, Y.: Pellet: A Practical OWL-DL Reasoner. In: *Journal of Web Semantics* (2006)
- [Smith 1985] SMITH, B. C.: Prologue to Reflection and Semantics in Procedural Language. In: BRACHMAN, R. J. (Hrsg.) ; LEVESQUE, H. J. (Hrsg.): *Readings in Knowledge Representation*, Morgan Kaufmann Publishers, Inc., 1985
- [Sowa 1984] SOWA, J. F.: *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, 1984
- [Star u. Estes 1990] STAR, J. ; ESTES, J.: *Geographic Information Systems – An Introduction*. Prentice Hall, 1990
- [Steele 1990] STEELE, Guy L. Jr.: *Common Lisp – The Language, Second Edition*. Digital Press, 1990
- [Tobies 2000] TOBIES, S.: The Complexity of Reasoning with Cardinality Restrictions and Nominals in Expressive Description Logics. In: *Journal of Artificial Intelligence Research* 12 (2000)
- [Tobies 2001] TOBIES, S.: *Complexity Results and Practical Algorithms for Logics in Knowledge Representation* / RWTH Aachen, Dissertation, 2001
- [Tsarkov u. a. 2004] TSARKOV, D. ; RIAZANOV, A. ; BECHHOFFER, S. ; HORROCKS, I.: Using Vampire to Reason with OWL. In: *Proceedings of the 2004 International Semantic Web Conference (ISWC 2004)*, 2004
- [Turhan u. a. 2006] TURHAN, A.-Y. ; BECHHOFFER, S. ; KAPLUNOVA, A. ; LIEBIG, T. ; LUTHER, M. ; MÖLLER, R. ; NOPPENS, O. ; PATEL-SCHNEIDER, P. ; SUNTISRIVARAPORN, B. ; WEITHÖNER, T.: DIG 2.0 Towards a Flexible Interface for Description Logic Reasoners. In: *Proceedings of the Second International Workshop on OWL: Experiences and Directions (OWLED 2006)*, 2006.
- [Tuschik u. Wolter 1993] TUSCHIK, H.-P. ; WOLTER, H.: *Mathematische Logik - kurzgefasst*. BI-Wissenschaftsverlag, 1993
- [Van Der Straeten u. a. 2003] VAN DER STRAETEN, R. ; SIMMONDS, J. ; MENS, T.: Detecting Inconsistencies between UML Models Using Description Logic. In: *Proceedings of the International Workshop on Description Logics (DL 2003)*, 2003

- [Vardi 1982] VARDI, M. Y.: The Complexity of Relational Query Languages. In: *Proceedings of the 14th ACM SIGACT Symp. on Theory of Computing (STOC 1982)*, 1982
- [Vardi u. Wolper 1986] VARDI, M. Y. ; WOLPER, P.: Automata-Theoretic Techniques for Modal Logics of Programs. In: *Journal of Computer and System Science* 32 (1986), April, Nr. 2
- [Vogel u. a. 2005] VOGEL, O. ; ARNOLD, I. ; CHUGHTAI, A. ; IHLER, E. ; MEHLIG, U. ; NEUMANN, Th. ; VÖLTER, M. ; ZDUN, U.: *Software-Architektur – Grundlagen – Konzepte – Praxis*. Elsevier Spektrum Akademischer Verlag, 2005
- [Wessel 1996] WESSEL, M.: *Entwicklung eines konzeptorientierten generischen Grafikeditors in Common Lisp* / Universität Hamburg, Fachbereich Informatik, Studienarbeit, 1996
- [Wessel 1998] WESSEL, M.: *Eine visuelle Sprache zur Definition räumlicher (ebener) Konstellationen*, / Universität Hamburg, Fachbereich Informatik, Studienarbeit, 1998
- [Wessel 2000a] WESSEL, M.: *Decidable and Undecidable Extensions of $\mathcal{ALC}$ with Composition-based Role Inclusion Axioms* / Universität Hamburg, Fachbereich Informatik, 2000 (FBI-HH-M-301/01). Forschungsbericht. URL: [http://kogs-www.informatik.uni-hamburg.de/~mwessel/report5.{ps.gz | pdf}](http://kogs-www.informatik.uni-hamburg.de/~mwessel/report5.{ps.gz|pdf})
- [Wessel 2000b] WESSEL, M.: *Obstacles on the Way to Spatial Reasoning with Description Logics – Undecidability of $\mathcal{ALC}_{\mathcal{RA}^\oplus}$* / Universität Hamburg, Fachbereich Informatik, 2000 (FBI-HH-M-297/00). Forschungsbericht. URL: [http://kogs-www.informatik.uni-hamburg.de/~mwessel/report4.{ps.gz | pdf}](http://kogs-www.informatik.uni-hamburg.de/~mwessel/report4.{ps.gz|pdf})
- [Wessel 2001a] WESSEL, M.: *Obstacles on the Way to Spatial Reasoning with Description Logics – Some Undecidability Results*. In: *Proceedings of the International Workshop on Description Logics (DL 2001)*, 2001
- [Wessel 2001b] WESSEL, M.: *Undecidability of $\mathcal{ALC}_{\mathcal{RA}}$* / Universität Hamburg, Fachbereich Informatik, 2001 (FBI-HH-M-302/01). Forschungsbericht. URL: [http://kogs-www.informatik.uni-hamburg.de/~mwessel/report6.{ps.gz | pdf}](http://kogs-www.informatik.uni-hamburg.de/~mwessel/report6.{ps.gz|pdf})
- [Wessel 2002] WESSEL, M.: *On Spatial Reasoning with Description Logics - Position Paper*. In: *Proceedings of the International Workshop on Description Logics (DL 2002)*, 2002
- [Wessel 2003a] WESSEL, M.: *Qualitative Spatial Reasoning with the $\mathcal{ALCI}_{RCC}$ -family – First Results and Unanswered Questions* / Universität Hamburg, Fachbereich Informatik, 2003 (FBI-HH-M-324/03). Forschungsbericht.

URL: [http://kogs-www.informatik.uni-hamburg.de/~mwessel/report7.{ps.gz | pdf}](http://kogs-www.informatik.uni-hamburg.de/~mwessel/report7.{ps.gz|pdf})

- [Wessel 2003b] WESSEL, M.: Some Practical Issues in Building a Hybrid Deductive Geographic Information System with a DL Component. In: *Proceedings of the 10th International Workshop on Knowledge Representation meets Databases (KRDB 2003)*, 2003
- [Wessel u. a. 2000] WESSEL, M. ; HAARSLEV, V. ; MÖLLER, R.: $\mathcal{ALC}_{\mathcal{RA}}$ – $\mathcal{ALC}$ with Role Axioms. In: *Proceedings of the International Workshop on Description Logics (DL 2000)*, 2000
- [Wessel u. Möller 2005] WESSEL, M. ; MÖLLER, R.: A High Performance Semantic Web Query Answering Engine. In: *Proceedings of the International Workshop on Description Logics (DL 2005)*, 2005
- [Wessel u. Möller 2006] WESSEL, M. ; MÖLLER, R.: A Flexible DL-based Architecture for Deductive Information Systems. In: *Proceedings of the IJCAR 2006 Workshop on Empirically Successful Computerized Reasoning (ESCoR 2006)*, 2006
- [Wessel u. Möller 2007a] WESSEL, M. ; MÖLLER, R.: Design Principles and Realization Techniques for User Friendly, Interactive, and Scalable Ontology Browsing and Inspection Tools. In: *Proceedings of the International Workshop on OWL: Experiences and Directions (OWLED 2007)*, 2007
- [Wessel u. Möller 2007b] WESSEL, M. ; MÖLLER, R.: Flexible Software Architectures for Ontology-Based Information Systems. In: *Journal of Applied Logic, Special Issue: Empirically Successful Computerized Reasoning 7* (2009), März, Nr. 1
- [Wikipedia 2006a] WIKIPEDIA: *Information Retrieval*. <http://de.wikipedia.org/wiki/Informationretrieval>, 2006
- [Wikipedia 2006b] WIKIPEDIA: *Wissensorganisation*. <http://de.wikipedia.org/wiki/Wissensorganisation>, 2006
- [Winston 1993] WINSTON, P. H.: *Artificial Intelligence*. 3. Auflage, Addison-Wesley, 1993
- [Woods u. Schmolze 1992] WOODS, W. ; SCHMOLZE, J.: The KL-One Family. In: LEHMAN, F. (Hrsg.): *Semantic Networks in Artificial Intelligence*. Pergamon Press, 1992
- [Woods 1975] WOODS, W. A.: What's in a Link: Foundations for Semantic Networks. In: BOBROW, D. G. (Hrsg.) ; COLLINS, A. M. (Hrsg.): *Representation and Understanding: Studies in Cognitive Computer Science*. Academic Press, 1975

A. Anhang

A.1. LUBM-TBox

(IN-TBOX LUBM)

```
(DEFINE-PRIMITIVE-ROLE |advisor|  
  :DOMAIN |Person| :RANGE |Professor|)  
(DEFINE-PRIMITIVE-ROLE |affiliatedOrganizationOf|  
  :DOMAIN |Organization| :RANGE |Organization|)  
(DEFINE-PRIMITIVE-ROLE |affiliateOf|  
  :DOMAIN |Organization| :RANGE |Person|)  
(DEFINE-PRIMITIVE-ROLE |degreeFrom| :INVERSE |hasAlumnus|  
  :DOMAIN |Person| :RANGE |University|)  
(DEFINE-PRIMITIVE-ROLE |hasAlumnus| :INVERSE |degreeFrom|  
  :DOMAIN |University| :RANGE |Person|)  
(DEFINE-PRIMITIVE-ROLE |doctoralDegreeFrom| :PARENTS |degreeFrom|  
  :DOMAIN |Person| :RANGE |University|)  
(DEFINE-PRIMITIVE-ROLE |headOf| :PARENTS |worksFor|  
  :DOMAIN (AND (OR (ALL |headOf| (NOT |College|)) |Dean|)))  
(DEFINE-PRIMITIVE-ROLE |worksFor| :PARENTS |memberOf|)  
(DEFINE-PRIMITIVE-ROLE |listedCourse|  
  :DOMAIN |Schedule| :RANGE |Course|)  
(DEFINE-PRIMITIVE-ROLE |mastersDegreeFrom| :PARENTS |degreeFrom|  
  :DOMAIN |Person| :RANGE |University|)  
(DEFINE-PRIMITIVE-ROLE |member| :INVERSE |memberOf|  
  :DOMAIN |Organization| :RANGE |Person|)  
(DEFINE-PRIMITIVE-ROLE |memberOf| :INVERSE |member|  
  :DOMAIN |Person| :RANGE |Organization|)  
(DEFINE-PRIMITIVE-ROLE |orgPublication|  
  :DOMAIN |Organization| :RANGE |Publication|)  
(DEFINE-PRIMITIVE-ROLE |publicationAuthor|  
  :DOMAIN |Publication| :RANGE |Person|)  
(DEFINE-PRIMITIVE-ROLE |publicationDate|  
  :DOMAIN |Publication|)  
(DEFINE-PRIMITIVE-ROLE |publicationResearch|  
  :DOMAIN |Publication| :RANGE |Research|)
```

A.1 LUBM-TBox

```
(DEFINE-PRIMITIVE-ROLE |researchProject|
  :DOMAIN |ResearchGroup| :RANGE |Research|)
(DEFINE-PRIMITIVE-ROLE |softwareDocumentation|
  :DOMAIN |Software| :RANGE |Publication|)
(DEFINE-PRIMITIVE-ROLE |softwareVersion|
  :DOMAIN |Software|)
(DEFINE-PRIMITIVE-ROLE |subOrganizationOf| :TRANSITIVE T
  :DOMAIN |Organization| :RANGE |Organization|)
(DEFINE-PRIMITIVE-ROLE |takesCourse|)
(DEFINE-PRIMITIVE-ROLE |teacherOf|
  :DOMAIN |Faculty| :RANGE |Course|)
(DEFINE-PRIMITIVE-ROLE |teachingAssistantOf|
  :DOMAIN |TeachingAssistant| :RANGE |Course|)
(DEFINE-PRIMITIVE-ROLE |tenured|
  :DOMAIN |Professor|)
(DEFINE-PRIMITIVE-ROLE |undergraduateDegreeFrom| :PARENTS |degreeFrom|
  :DOMAIN |Person| :RANGE |University|)

(DEFINE-DATATYPE-PROPERTY |age|)
(DEFINE-DATATYPE-PROPERTY |name|)
(DEFINE-DATATYPE-PROPERTY |telephone|)
(DEFINE-DATATYPE-PROPERTY |emailAddress|)
(DEFINE-DATATYPE-PROPERTY |officeNumber|)
(DEFINE-DATATYPE-PROPERTY |researchInterest|)
(DEFINE-DATATYPE-PROPERTY |title|)

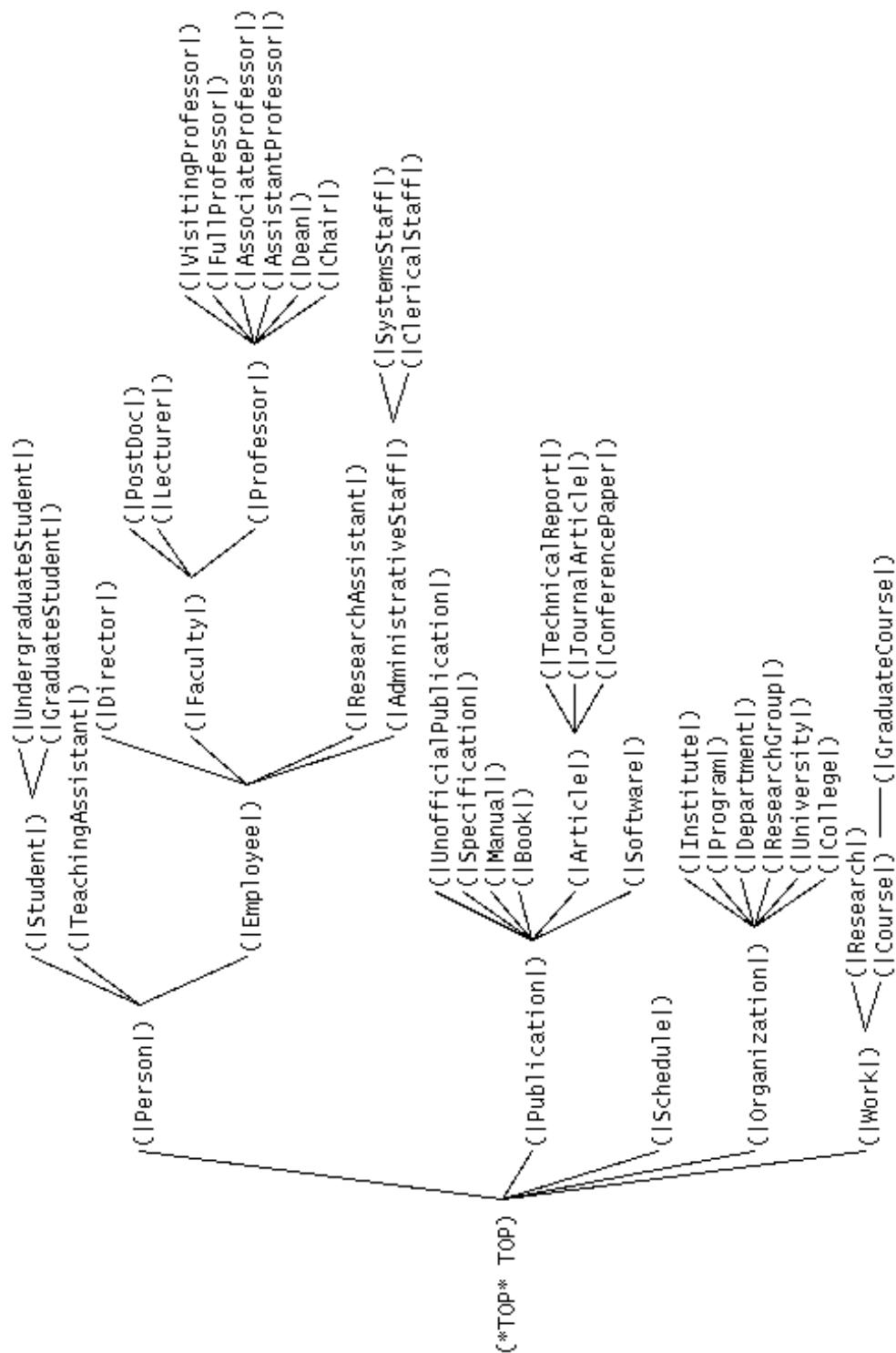
(IMPLIES |Schedule| *TOP*)
(IMPLIES |Research| |Work|)
(IMPLIES |ResearchGroup| |Organization|)
(IMPLIES |Publication| *TOP*)
(IMPLIES |Software| |Publication|)
(IMPLIES |Organization| *TOP*)
(IMPLIES |Faculty| |Employee|)
(IMPLIES |Course| |Work|)
(IMPLIES |Professor| |Faculty|)
(IMPLIES |University| |Organization|)
(IMPLIES |Person| (OR (ALL |headOf| (NOT |Department|)) |Chair|))
(IMPLIES |AdministrativeStaff| |Employee|)
(IMPLIES |Article| |Publication|)
(IMPLIES |AssistantProfessor| |Professor|)
(IMPLIES |AssociateProfessor| |Professor|)
(IMPLIES |Book| |Publication|)
(IMPLIES |Chair| (AND (SOME |headOf| |Department|) |Professor| |Person|))
(IMPLIES |Department| |Organization|)
(IMPLIES |ClericalStaff| |AdministrativeStaff|)
```



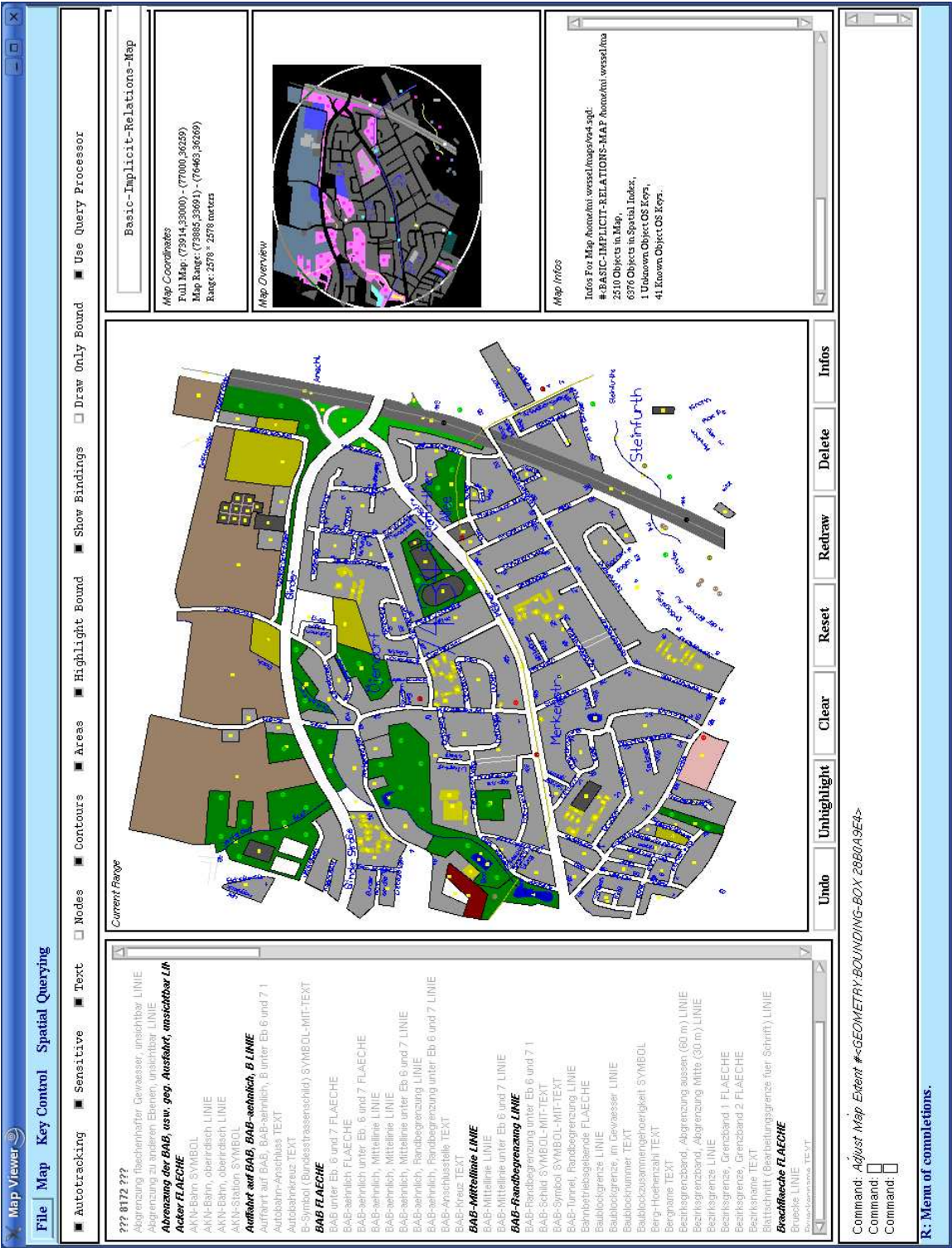
```
(IMPLIES |College| |Organization|)
(IMPLIES |ConferencePaper| |Article|)
(IMPLIES |Dean| (AND (SOME |headOf| |College|) |Professor|))
(IMPLIES |Program| |Organization|)
(IMPLIES |FullProfessor| |Professor|)
(IMPLIES |GraduateCourse| |Course|)
(IMPLIES |GraduateStudent|
  (AND (SOME |takesCourse| |GraduateCourse|) |Person|))
(IMPLIES |Institute| |Organization|)
(IMPLIES |JournalArticle| |Article|)
(IMPLIES |Lecturer| |Faculty|)
(IMPLIES |Manual| |Publication|)
(IMPLIES |PostDoc| |Faculty|)
(IMPLIES |ResearchAssistant|
  (AND (SOME |worksFor| |ResearchGroup|) |Person|))
(IMPLIES |Specification| |Publication|)
(IMPLIES |SystemsStaff| |AdministrativeStaff|)
(IMPLIES |TechnicalReport| |Article|)
(IMPLIES |UndergraduateStudent| |Student|)
(IMPLIES |UnofficialPublication| |Publication|)
(IMPLIES |VisitingProfessor| |Professor|)
(IMPLIES |Work| *TOP*)

(DEFINE-CONCEPT |TeachingAssistant|
  (AND (SOME |teachingAssistantOf| |Course|) |Person|))
(DEFINE-CONCEPT |Director|
  (AND (SOME |headOf| |Program|) |Person|))
(DEFINE-CONCEPT |Employee|
  (AND (SOME |worksFor| |Organization|) |Person|))
(DEFINE-CONCEPT |Student|
  (AND (SOME |takesCourse| |Course|) |Person|))
```

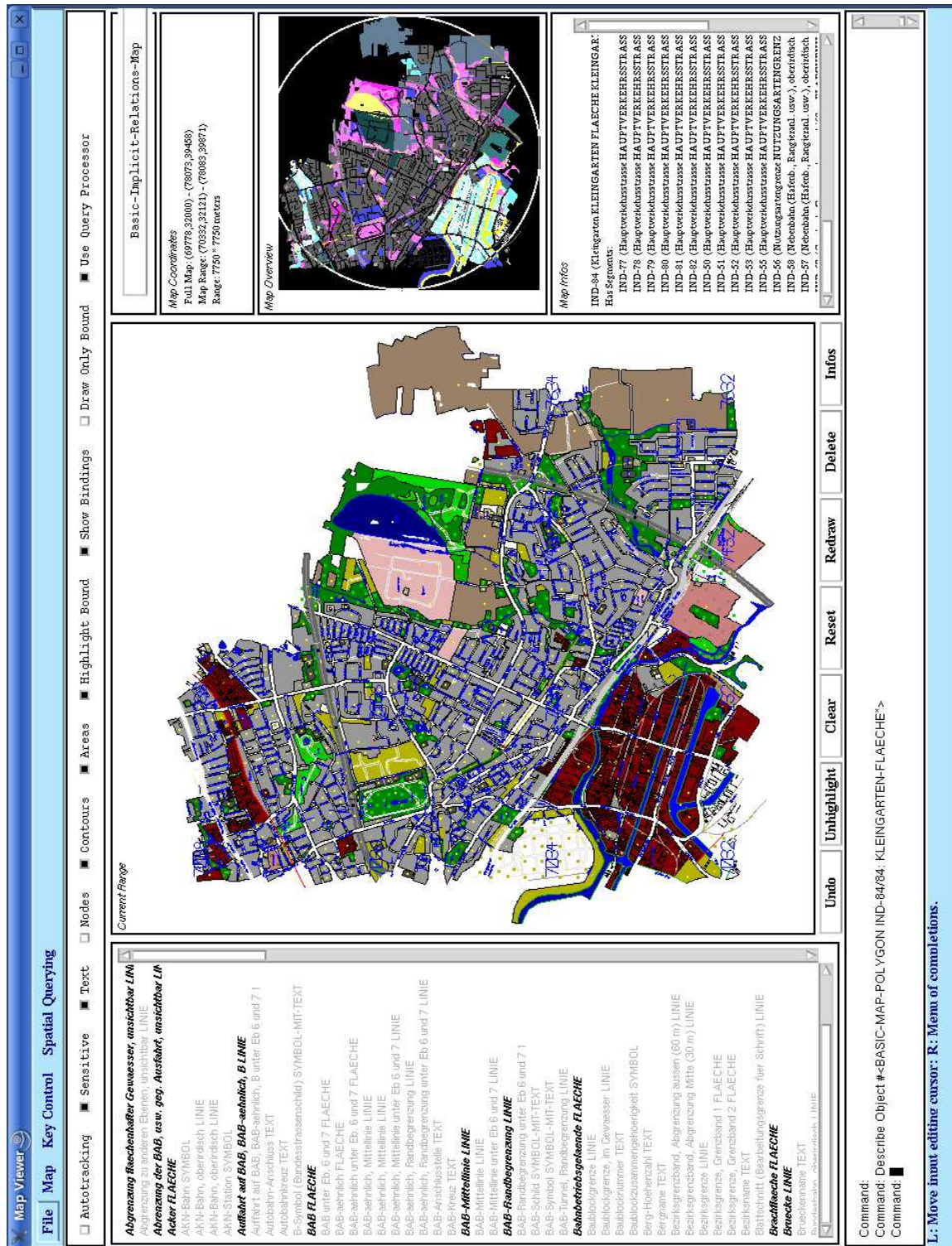
A.2. LUBM-Taxonomie



A.3. DISK 1



A.4. DISK 2



A.5. Verzeichnis wichtiger Akronyme, Abkürzungen und Namen

ABF Anfrage-Bearbeitungs-Funktion	DSA Daten-Substrat-Anfragesprache
ABox Assertionale Box	EC RCC-Relation: Externally Connected
ABP Anfrage-Bearbeitungs-Plan	EQ RCC-Relation: Equal
ADS Active Domain Semantic	ER Entity Relationship
ADT Abstrakter Datentyp	Fact Fast Classification of Concept Terms (DL-System)
API Application Program Interface	GAG Gerichteter Azyklischer Graph
ASG Abstract Syntax Graph	GCI General Concept Inclusion
AST Abstract Syntax Tree	GIS Geographisches Informationssystem
BAA Basis-ABox-Anfragesprache	GUI Graphical User Interface
CAD Computer Aided Design	HTTP Hypertext Transport Protocol
CDA Closed Domain Assumption	IR Information Retrieval
CTDS Clash IDs	IS Informationssystem
CLASSIC Classification of Individuals and Concepts (DL-System)	ISO International Standards Organization
CLIM COMMON LISP Interface Manager	JENA SEMANTIC WEB Toolkit
CLOS COMMON LISP Object System	JEPD Jointly Exhaustive, Pairwise Disjoint
CLSQL COMMON LISP ODBC-Treiber	KAON2 The Karlsruhe Ontology and Semantic Web Tool Suite (DL-System)
CWA Closed World Assumption	KD Konkrete Domäne
DAG Directed Acyclic Graph	KI Künstliche Intelligenz
DB Datenbank	KIF Knowledge Interchange Format
DBMS Data Base Management System	KL-ONE Knowledge Representation System One (DL-System)
DBS Datenbank-Verwaltungssystem	Km Total vermaschter Graph
DC RCC-Relation: Discrete	Kn Eine einfache Multi-Modallogik
DIG Description Logic Implementors Group	KNF Konjunktive Normalform
DISK Digitale Stadtkarte	KPO Kopf-Projektions-Operator
DISKQL DISK Query Language	KRIS Knowledge Representation and Inference System (DL-System)
DL Description Logic (Beschreibungslogik)	KRSS Knowledge Representation System Standard
DLMAPS Description Logic Maps	LUBM Lehigh University Benchmark
DNF Disjunktive Normalform	
DR RCC-Relation: Disconnected	

A.5 Verzeichnis wichtiger Akronyme, Abkürzungen und Namen

MIDeLoRA Michael's Description Logic Reasoner (DL-System)	RDBS Relationales Datenbankverwaltungssystem
NAF Negation As Failure	RDF Resource Description Framework
NNF Negationsnormalform	RDFS Resource Description Framework Schema
NRQL New RACERPRO Query Language	RDF(S) RDF + RDFS
NTPP RCC-Relation: Non-Tangentially Proper Part	RVM Role Value Map
NTPPI RCC-Relation: Non-Tangentially Proper Part Inverse	SDT Spatial Data Type
O RCC-Relation: Overlaps	SPARQL Eine RDF-Anfragesprache
ODA Open Domain Assumption	SQD Siemens CAD (SICAD) Dateiformat
ODBC Open Database Connectivity	SQL Structured Query Language
ONE RCC-Relation: Overlapping, but Not Equal	SR RCC-Relation: Spatially Related
OO Objektorientiert	STS TUHH-Institut „Software, Technologie und Systeme“
OOP Objektorientierte Programmierung	SWIR SEMANTIC WEB Information Retrieval
OWA Open World Assumption	SWRL SEMANTIC WEB Rule Language
OWL Web Ontology Language	TBox Terminologische Box
OWL-QL OWL Query Language	TCP/IP Netzwerkprotokolle
PDA Personal Digital Assistant	TPP RCC-Relation: Tangentially Proper Part
PELLET DL-System	TPPI RCC-Relation: Tangentially Proper Part Inverse
PO RCC-Relation: Partial Overlap	TUHH Technische Universität Hamburg-Harburg
RACERPORTER GUI für RACERPRO	TV Told Value
PP RCC-Relation: Proper Part	UML Unified Modeling Language
PPI RCC-Relation: Proper Part Inverse	UNA Unique Name Assumption
QBox Query Box	URI Uniform Resource Identifier
RACE Reasoner for ABox and Concept Expressions (DL-System)	URIRef Uniform Resource Identifier Reference
RACER Renamed ABox and Concept Expression Reasoner (DL-System)	URL Uniform Resource Locator
RACERPRO Kommerzielle Variante / Nachfolger von RACER (DL-System)	URN Uniform Resource Name
RBox Relations- oder Rollen-Box	W3C World Wide Web Consortium
RCC Region Connection Calculus bzw. entsp. Relationsmenge	WBS Wissensbasiertes System

WRH Wissensrepräsentations-Hypothese

WWW World Wide Web

XML Extensible Markup Language

XSD XML-Schema

B. Lebenslauf

Persönliche Daten

Michael Achim Wessel
Willy-Nissen-Ring, 15a
22179 Hamburg
Tel.: (040) 63 97 31 90
E-Mail: michael.wessel@gmx.de
Geb. am 08.08. 1970 in Hamburg
Ledig, deutsch, keine Kinder

Schule

1977–1981	Grundschule Burgunderweg in Hamburg-Niendorf.
1981–1990	Gymnasium Ohmoor in Hamburg-Niendorf (Leistungskurse Mathematik und Chemie).

Zivildient

08/1990–10/1991	Zivildienst im allg. Krankenhaus Ochsenzoll in Hamburg-Langenhorn.

Studium

10/1991–04/1998	Studium der Informatik an der Universität Hamburg mit Ergänzungsfach Physik (Vordiplom und Diplom mit Gesamtnote „Sehr Gut“).

Beruf

06/1998–06/1999	Wissenschaftlicher Mitarbeiter im Arbeitsbereich „Kognitive Systeme“ am Fachbereich Informatik der Universität Hamburg.
07/1999–06/2001	Stipendiat an der Universität Hamburg (Grundstipendium); weiterhin assoziiert mit dem Arbeitsbereich „Kognitive Systeme“.
07/2001–08/2004	Wissenschaftlicher Mitarbeiter im DFG-Projekt „DLS“ am Arbeitsbereich „Kognitive Systeme“.
09/2004–06/2006	Wissenschaftlicher Mitarbeiter im „Institut für Softwaresysteme (STS)“ der Technischen Universität Hamburg-Harburg.

07/2006–heute Vollzeitbeschäftigter Unternehmer bzw. Kommanditist der Firma Racer Systems GmbH & Co. KG, Blumenau 50, 22089 Hamburg (Durchführung von Beratungen und Projekten zum Thema SEMANTIC WEB, Softwareentwicklung, etc.).

Hamburg, 18. Juli 2007