Applying Markov Logics for Controlling Abox Abduction

Vom Promotionsausschuss der
Technischen Universität Hamburg-Harburg
zur Erlangung des akademischen Grades
Doktor Ingenieurin
genehmigte Dissertation

von
Dipl.-Ing. Anahita Nafissi

aus Shiraz, Iran

2013

Reviewers:
Prof. Dr. Ralf Möller
Prof. Dr. Bernd Neumann

Day of the defense:
10.10.2013

Abstract
Manually annotating the multimedia documents is a time-consuming and cost-intensive task. In this work, we define a media interpretation agent for automatically generating annotations for multimedia documents. Observations of the agent are given as surface-level information extracted by state-of-the-art media analysis tools. Based on background knowledge the agent interprets observations by computing high-level explanations. Observations and their explanations constitute the annotations of a media document. For this purpose, we investigate an abduction algorithm which computes explanations using a logic-based knowledge representation formalism. Multiple explanations might be possible for certain media content. Since the agent's resources for computing explanations are limited, we need to control the abduction procedure in terms of branching of the computation process and "depth" of computed results, while still producing acceptable annotations. To control the abduction procedure, we employ a first-order probabilistic formalism.

Kurzfassung
Die manuelle Erstellung von Anmerkungen zu Multimedia-Dokumenten ist eine zeit- und kostenintensive Aufgabe. In dieser Arbeit definieren wir einen Agenten zur Medieninterpretation, der automatisch Anmerkungen zu Multimedia-Dokumenten generiert. Die Beobachtungen des Agenten werden durch oberflächliche Informationen gegeben, die mit Hilfe von modernen Medien-Analyse-Werkzeugen extrahiert wurden. Auf der Basis von Hintergrundwissen interpretiert der Agent diese Beobachtungen durch die Herleitung von Erklärungen auf einer höheren Ebene. Beobachtungen und ihre Erklärungen bilden die Anmerkungen zu einem Multimedia-Dokument. Zu diesem Zweck untersuchen wir einen Abduktionsalgorithmus, der Erklärungen durch die Verwendung eines logikbasierten Wissensrepräsentationsformalismus bestimmt. Für bestimmte Medieninhalte können auch mehrere Erklärungen möglich sein. Da dem Agenten zur Bestimmung der Erklärungen nur begrenzte Ressourcen zur Verfügung stehen, müssen wir die Abduktionsprozedur in Bezug auf die Verzweigung des Rechenverfahrens und Tiefe der berechneten Ergebnisse beschränken, wobei immer noch akzeptable Anmerkungen generiert werden sollen. Um die Abduktionsprozedur zu beschränken, benutzen wir einen probabilistischen Formalismus erster Ordnung.

To my dear parents

Acknowledgements

It is a pleasure to pay tribute to those who made the completion of this thesis possible:

I would like to express my deep and sincere gratitude to my supervisor Prof. Dr. Ralf Möller for giving me the opportunity to do research in this exciting field and supporting throughout this work. I would also like to thank Prof. Dr. Bernd Neumann for reviewing my thesis.

Special thanks go to Maurice Rosenfeld and Björn Hass for evaluating the results. I am also very grateful to my colleagues at the Institute for Software Systems: Dr. Michael Wessel, Dr. Atila Kaya, Oliver Gries, Kamil Sokolski, Dr. Sofia Espinosa, Dr. Özgür Özçep, Tobias Näth, and Karsten Martiny. I would also like to thank all colleagues in the CASAM and PRESINT projects.

Last but not least, sincere thanks to my family whose support, patience and encouragement enabled me to complete this thesis.

Contents

1 Introduction **1**
- 1.1 Motivation and Objectives . . . 1
- 1.2 Contributions . . . 2
- 1.3 Outline of the Dissertation . . . 4

2 Preliminaries **5**
- 2.1 Description Logics, Queries, and Rules . . . 5
  - 2.1.1 Syntax, Semantics, and Decision Problems . . . 5
  - 2.1.2 Extensions: Queries, and Rules . . . 9
- 2.2 Probabilistic Representation Formalisms . . . 12
  - 2.2.1 Basic Notions of Graph and Probability Theory . . . 12
  - 2.2.2 Bayesian Networks . . . 16
  - 2.2.3 Markov Logic Networks . . . 21

3 Abduction for Data Interpretation: Related Work **31**
- 3.1 Perception as Abduction: The General Picture . . . 31
- 3.2 Probabilistic Horn Abduction and Independent Choice Logic . . . 33
- 3.3 Computing Explanations as Markov Logic Inference . . . 40
- 3.4 Preference Models Using Bayesian Compositional Hierarchies . . . 43
- 3.5 DLLP-Abduction Based Interpretation . . . 44
  - 3.5.1 DLLP Abduction . . . 45
  - 3.5.2 Ranking Simple Explanations . . . 48
- 3.6 The Need for Well-Founded Control for Sequential Abduction Steps . . . 49

4 Probabilistic Control for DLLP-Abduction Based Interpretation **51**
- 4.1 Abduction-based Interpretation Inside an Agent . . . 51
- 4.2 Controlling the Interpretation Process . . . 60
  - 4.2.1 Controlling Branching . . . 60
  - 4.2.2 Controlling Abduction Depth . . . 61
  - 4.2.3 Controlling Reactivity . . . 65
- 4.3 Comparison with Independent Choice Logic . . . 66
- 4.4 Conversion of the Knowledge Base into ML Notation . . . 67

5 Evaluation **79**
5.1 Optimization Techniques 79
5.2 Case Study: CASAM 84
5.3 Hypotheses and Results 85
5.4 Quality of the Interpretation Results 93

6 Conclusions **97**

A Alchemy Knowledge Representation System and Language **103**
A.1 Alchemy Knowledge Representation Language 103
A.2 Interfaces to Alchemy 105
A.3 Inference Services 106
A.4 Peculiarities of the Inference Engine Alchemy 108

B RacerPro and its Module for DLLP Abduction **115**

C The DLLP-Abduction Based Interpretation System **117**

Bibliography **118**

Chapter 1

Introduction

1.1 Motivation and Objectives

The world wide web contains a huge amount of documents, and their number grows rapidly over time. To simplify retrieval processes for human users, in recent years several content-based search engines have been developed. Despite the fact that many irrelevant documents are shown, these engines are successful because humans can quite easily eliminate irrelevant entries in the retrieval result set. Nevertheless, content-based retrieval is pushed to its limits. To further improve retrieval results, in particular also for automatic processes accessing multimedia documents, the semantics of the multimedia documents must be considered by retrieval processes. This is the issue discussed in the field of the semantic web. According to the idea of the semantic web, multimedia documents are analyzed by state-of-the-art analysis tools and subsequently are automatically interpreted in order to produce annotations to be attached to media documents. Semantics-based retrieval of multimedia documents is then based on these annotations.

In previous work, interpretation of multimedia documents has been formalized by defining an interpretation agent which, in order to compute interpretations, uses an abduction algorithm for computing explanations for "observations" [MN08, EKM09]. Observations are seen as surface-level information extracted by state-of-the-art analysis tools. Using the abduction algorithm, the agent interprets observations by computing high-level explanations for (lower-level) observations. Observations and their explanations constitute the annotations of a media document. Usually, multiple explanations might be possible for certain media content. As a knowledge representation language a combination of description logic [BCM$^+$03] and Horn logic [Llo87] has been used for defining the space of possible interpretations of media content in an operational way as in [MN08, Kay11, Esp11]. Although declarative abduction approaches have been proposed [DK02, LY02], fixing the number of ground literals that can be abduced, which is common to these approaches, is seen as a limitation because guessing ground literals in beforehand is considered to be hardly possible in our media interpretation context.

Since the agent's resources for computing explanations are limited, we need to control the abduction procedure in terms of branching limitations as well as depth restrictions, while still producing acceptable annotations. The general procedure for information processing is an implementation of the architecture described in Robert Kowalski's influential book on Computational Logic and Human Thinking [Kow11]. The focus of the work discussed in this doctoral thesis is on how to control abduction as part of Kowalski's framework. In particular, besides computing new assertions about the world that explain the observations (or help realizing achievement goals in Kowalski's framework), the main idea about why an agent should actually apply abduction is to reduce uncertainty about incoming observations (or about the realizability of achievement goals), and newly incoming assertions representing new knowledge available to the agent should increase uncertainty again, which, in turn, is then reduced by abductive reasoning [FK00, Pau93]. In this spirit, data interpretation, is seen as reduction of uncertainty.

In this work, we investigate a first-order probabilistic formalism, Markov logic [DR07], for controlling abduction as an inference task used to formalize data interpretation. Note that Kowalski [Kow11] does not consider uncertainty in this way. We investigate how the interpretation process can deal with input data that are uncertain and may be even inconsistent, and explore how ranking of possible interpretations can be accomplished in terms of a probabilistic scoring function. Extending the approach described in [Kay11], in this work we would like to increase the expressivity of the knowledge representation language by supporting recursive Horn rules. Furthermore, we investigate ways to incrementally process input data, thereby coming up with interpretation results as early as possible. The main requirement of this work is that by explaining the observations successively, the ranks of the interpretation alternatives increase monotonically. Consequently, the idea is that interpretation increases the belief of an agent in its observations, and thus, interpretation provides a better basis for computing appropriate actions.

1.2 Contributions

The scientific contributions of the thesis are summarized as follows: the thesis provides a formalization of first-order abduction-based media interpretation in terms of a probabilistic ranking principle for interpretation alternatives. In this context, interpretation is formalized using a set of weighted Horn rules in combination with a description logic knowledge base. The semantics for weights is defined in terms of Markov logic [DR07]. Interpretations are defined as explanations of ground formulas modeling observations of an agent. Explanations (and observations) are stored as annotations for media documents in order to better support semantics-based retrieval. The task of the interpretation engine is to generate deep-level information of multimedia documents according to the given surface-level information. The surface-level information extracted by state-of-

the-art analysis tools is supplied as incremental input to the probabilistic interpretation engine. We present solutions for branching and depth control problems in abduction-based interpretation engines, and we evaluate our probabilistic interpretation engine in a practical scenario.

Besides media interpretation, the thesis also contributes to the understanding of software agents which incrementally process multimodal data in an incremental way. We propose the use of Markov logic [DR07] to define the motivation for the agent to generate interpretations, namely to reduce uncertainty of its observations. This principle can be seen as an implicit utility measure built into the agent's architecture. Given a set of observations about which the agent might be uncertain or which might be even inconsistent given the agent's knowledge base, we investigate how Markov logic can help to interpret the observations based on a maximal subset of observations that the agent assumes most-probably to be true. Interpretation of the agent's observations is formalized as a first-order abduction process for computing explanations based on a set of interpretation rules (weighted Horn formulas) and an ontology (description logic axioms). Using a Markov logic based ranking principle, the agent architecture supports incremental processing of new observations such that by generating interpretations (explanations) over time the agent reduces its uncertainty, until then new observations are provided as input and the process starts over again. As new data becomes available, running interpretation processes are possibly be interrupted such that the agent can focus on most recent observations.

This work has been supported by the European Commission as part of the CASAM project (Contract FP7-217061) and by the German Science Foundation as part of the PRESINT project (DFG MO 801/1-1). The following papers have been published up to now as part of this thesis:

- O. Gries, R. Möller, A. Nafissi, M. Rosenfeld, K. Sokolski, and M. Wessel. A Probabilistic Abduction Engine for Media Interpretation based on Ontologies. In Pascal Hitzler and Thomas Lukasiewicz, editors, Proceedings of 4th International Conference on Web Reasoning and Rule Systems (RR-2010), September 2010.

- O. Gries, R. Möller, A. Nafissi, M. Rosenfeld, K. Sokolski, and M. Wessel. A Probabilistic Abduction Engine for Media Interpretation based on Ontologies. In Thomas Lukasiewicz, Rafael Penaloza, and Anni-Yasmin Turhan, editors, Proceedings of International Workshop on Uncertainty in Description Logics (UnIDL-2010), 2010.

Project deliverable to which thesis has contributed have been submitted to the European Commission:

- O. Gries, R. Möller, A. Nafissi, M. Rosenfeld, and K. Sokolski. Formalisms Supporting First-order Probabilistic Structures. CASAM project deliverable D3.1. Institute for Software Systems (STS), Hamburg University of Technology, 2008.

- O. Gries, R. Möller, A. Nafissi, M. Rosenfeld, and K. Sokolski. CASAM Domain Ontology. CASAM project deliverable D6.2. Institute for Software Systems (STS), Hamburg University of Technology, 2009.
- O. Gries, R. Möller, A. Nafissi, M. Rosenfeld, K. Sokolski, and M. Wessel. Basic Reasoning Engine: Report on Optimization Techniques for First-Order Probabilistic Reasoning. CASAM project deliverable D3.2. Institute for Software Systems (STS), Hamburg University of Technology, 2009.
- O. Gries, R. Möller, A. Nafissi, M. Rosenfeld, K. Sokolski, and M. Wessel. Probabilistic Abduction Engine: Report on Algorithms and the Optimization Techniques used in the Implementation. CASAM project deliverable D3.3. Institute for Software Systems (STS), Hamburg University of Technology, 2010.
- O. Gries, R. Möller, A. Nafissi, M. Rosenfeld, K. Sokolski, and M. Wessel. Meta-Level Reasoning Engine, Report on Meta-Level Reasoning for Disambiguation and Preference Elicitation. CASAM project deliverable D3.4. Institute for Software Systems (STS), Hamburg University of Technology, 2010.

1.3 Outline of the Dissertation

The remaining parts of the thesis are structured as follows. Chapter 2 explains the syntax and semantics of the Description Logic language $\mathcal{ALH}_f{}^-$ [BCM+03] as well as the signature of the knowledge base used in this work. Basic definitions of the probability theory used in this thesis are introduced. Moreover, probabilistic formalisms applied in this work are described and examples are given. In Chapter 3, we discuss related work on (probabilistic) abduction.

Chapter 4 presents the probabilistic interpretation engine. We discuss how the interpretation procedure is performed, possibly based on inconsistency and uncertainty in the input data. We also introduce an approach for probabilistically ranking interpretation alternatives. For this purpose, we define a probabilistic scoring function according to the Markov logic formalism [DR07]. Furthermore, we present a media interpretation agent which generates annotations for multimedia documents. We define an approach for improving abduction in terms of branching and depth control.

Chapter 5 evaluates the proposed approach in a practical scenario, showing that abduction control based on probabilistic ranking interpretations results in plausible agent behavior in general, and provides for sensible media annotations to be generated in the particular application scenario.

The last chapter, Chapter 6, summarizes this thesis and provides an outlook to the future work.

Chapter 2

Preliminaries

In this chapter, we explain the syntax and semantics of the description logic knowledge representation language $\mathcal{ALH}_f{}^-$. We also define basic definitions of probability theory and describe probabilistic formalisms applied in this work.

2.1 Description Logics, Queries, and Rules

Description logics (DLs) [BCM+03] are defined as decidable fragments of first-order logic [RN03] with varying levels of expressivity. DLs provide well understood means to establish ontologies. Therefore they are also used as representation languages for the semantic web [BHS05]. Section 2.1 is mostly taken from [GMN+10b, GMN+10c, EKM09].

For specifying the ontology used to describe surface-level analysis results as well as deep-level interpretation results, in this work a less expressive description logic is applied to facilitate fast computations for typical-case inputs. We decided to represent the domain knowledge with the DL $\mathcal{ALH}_f{}^-$ (a restricted attributive concept language with role hierarchies, functional roles and a string-based concrete domain) [BCM+03]. The motivation for supporting only restricted existential restrictions is to provide a well-founded integration of the description logic part of the knowledge base with the probabilistic part (based on Markov logic networks [DR07], see Section 2.2.3) in the context of abduction using Horn rules (see below for details). In addition, the description logic inference system used in this thesis (RacerPro [HM01]) is known to empirically perform well for inference problems w.r.t. this fragment.

2.1.1 Syntax, Semantics, and Decision Problems

In logic-based approaches, atomic representation units have to be specified. The atomic representation units are fixed using a so-called signature. A DL *signature* is a tuple $\mathcal{S} = (\mathbf{CN}, \mathbf{RN}, \mathbf{IN})$, where $\mathbf{CN} = \{A_1, ..., A_n\}$ is the set of concept names (denoting sets of domain objects) and $\mathbf{RN} = \{R_1, ..., R_m\}$ is the set of role names (denoting relations

between domain objects). The signature also contains a component **IN** indicating a set of individuals (names for domain objects).

At this point, we present the signature $\mathcal{S}$ whose notions will be used later as an ontology in several examples. The set of the concept names **CN** is given as follows:

$$\begin{aligned}\mathbf{CN} = \{&CarEntry, CarExit, EnvConference, EnvProt, Movement, HumanHealth,\\ &Car, DoorSlam, Building, Vehicle, Audio, Applause, Rain, Flood,\\ &AirPurification, WaterPollution, AirPollution, NoisePollution, TrafficJam,\\ &OilPollution, CityWithAirPollution, Env, EnvWorkshop, Energy, Winds,\\ &CityWithIndustry, CityWithTrafficJam, RenewableEnergy\}\end{aligned}$$

where *EnvConference*, *EnvProt*, *Env*, and *EnvWorkshop* indicate environmental conference, environmental protection, environment, and environmental workshop, respectively. Additionally, the set of the role names **RN** and the set of the individual names **IN** (also called individuals for brevity) are represented as follows:

$$\begin{aligned}\mathbf{RN} = \{&causes, hasObject, hasEffect, occursAt, hasTopic, adjacent, hasEvent,\\ &hasTheme, hasSubEvent, hasPart, hasLocation, EnergyToWinds\}\end{aligned}$$

$$\mathbf{IN} = \{c_1, c_2, ds_1, ds_2, ind_{42}, \ldots, ind_{45}, hamburg, berlin\}$$

Just a fragment of the signature can be listed here for reasons of brevity. In order to relate concept names and role names to each other (terminological knowledge) and to talk about specific individuals (assertional knowledge), a knowledge base has to be specified.

An $\mathcal{ALH}_f{}^-$ *knowledge base* $\Sigma_\mathcal{S} = (\mathcal{T}, \mathcal{A})$, defined with respect to a signature $\mathcal{S}$, comprises a terminological component $\mathcal{T}$ (called *Tbox*) and an assertional component $\mathcal{A}$ (called *Abox*). In the following, we just write Σ if the signature is clear from context. A Tbox is a set of so-called *axioms*, which are restricted to the following form in $\mathcal{ALH}_f{}^-$:

1. Subsumption $A_1 \sqsubseteq A_2$, $R_1 \sqsubseteq R_2$
2. Disjointness $A_1 \sqsubseteq \neg A_2$
3. Domain and range restrictions for roles $\exists R.\top \sqsubseteq A$, $\top \sqsubseteq \forall R.A$
4. Functional restriction on roles $\top \sqsubseteq (\leq 1\, R)$
5. Local range restrictions for roles $A_1 \sqsubseteq \forall R.A_2$
6. Definitions with value restrictions $A \equiv A_0 \sqcap \forall R_1.A_1 \sqcap \ldots \sqcap \forall R_n.A_n$

With axioms of the form shown in Item 1, concept (role) names can be declared to be subconcepts (subroles) of each other. An example for a concept subsumption axiom is:

$$carEntry \sqsubseteq Movement$$

With axioms of the form shown in Item 6., so-called definitions (with necessary and sufficient conditions) can be specified for concept names found on the left-hand side of the $\equiv$ sign. In axioms, so-called *complex concepts* are used. Complex concepts are concept names or expressions of the form $\top$ (anything), $\bot$ (nothing), $\neg A$ (atomic negation), $(\leq 1\, R)$ (role functionality), $\exists R.\top$ (limited existential restriction), $\forall R.A$ (value restriction) and $(C_1 \sqcap ... \sqcap C_n)$ (concept conjunction) with C_i being complex concepts. Concept names and complex concepts are also called concepts, and role names are also called roles for brevity.

At this point, a Tbox $\mathcal{T}$ is given which, among subsumption axioms, contains disjointness axioms as well as domain and range restrictions. This Tbox will be applied later in this work:

$$\begin{array}{l} Car \sqsubseteq Vehicle \\ DoorSlam \sqsubseteq Audio \\ CarEntry \sqsubseteq Movement \\ CarExit \sqsubseteq Movement \\ \\ Car \sqsubseteq \neg DoorSlam \\ Car \sqsubseteq \neg Audio \\ Vehicle \sqsubseteq \neg DoorSlam \\ Vehicle \sqsubseteq \neg Audio \\ CarEntry \sqsubseteq \neg CarExit \\ \\ \exists causes.\top \sqsubseteq Car \\ \top \sqsubseteq \forall causes.DoorSlam \end{array}$$

Table 2.1: An example for a Tbox $\mathcal{T}$

Knowledge about individuals is represented in the Abox part of Σ. An Abox $\mathcal{A}$ is a set of expressions of the form $A(a)$ or $R(a, b)$ (concept assertions and role assertions, respectively) where A stands for a concept name, R stands for a role name, and a, b stand for individuals. Aboxes can also contain equality ($a = b$) and inequality assertions ($a \neq b$). We say that the unique name assumption (UNA) is applied, if $a \neq b$ is assumed for all pairs of individuals a and b. In the following, an example for an Abox is given:

$Car(c_1)$
$DoorSlam(ds_1)$
$causes(c_1, ds_1)$
$CarEntry(ind_{42})$
$hasObject(ind_{42}, c_1)$
$hasEffect(ind_{42}, ds_1)$
$Car(c_2)$
$DoorSlam(ds_2)$
$causes(c_2, ds_2)$
$CarExit(ind_{43})$
$hasObject(ind_{43}, c_2)$
$hasEffect(ind_{43}, ds_2)$

Table 2.2: An example for an Abox $\mathcal{A}$

In order to understand the notion of logical entailment, we introduce the semantics of $\mathcal{ALH}_f{}^-$. In DLs such as $\mathcal{ALH}_f{}^-$, the semantics is defined with interpretations $\mathcal{I} = (\triangle^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\triangle^{\mathcal{I}}$ is a non-empty countable set of domain objects (called the domain of $\mathcal{I}$) and $\cdot^{\mathcal{I}}$ is an interpretation function which maps individuals to objects of the domain ($a^{\mathcal{I}} \in \triangle^{\mathcal{I}}$), atomic concepts to subsets of the domain ($A^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}}$) and roles to subsets of the cartesian product of the domain ($R^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}}$). The interpretation of arbitrary $\mathcal{ALH}_f{}^-$ concepts is then defined by extending $\cdot^{\mathcal{I}}$ to all $\mathcal{ALH}_f{}^-$ concept constructors:

$$
\begin{array}{lcl}
\top^{\mathcal{I}} & = & \triangle^{\mathcal{I}} \\
\bot^{\mathcal{I}} & = & \emptyset \\
(\neg A)^{\mathcal{I}} & = & \triangle^{\mathcal{I}} \setminus A^{\mathcal{I}} \\
(\leq 1\, R)^{\mathcal{I}} & = & \{u \in \triangle^{\mathcal{I}} \mid (\forall v_1, v_2)\, [((u, v_1) \in R^{\mathcal{I}} \wedge (u, v_2) \in R^{\mathcal{I}}) \rightarrow v_1 = v_2] \\
(\exists R.\top)^{\mathcal{I}} & = & \{u \in \triangle^{\mathcal{I}} \mid (\exists v)\, [(u, v) \in R^{\mathcal{I}}]\} \\
(\forall R.C)^{\mathcal{I}} & = & \{u \in \triangle^{\mathcal{I}} \mid (\forall v)\, [(u, v) \in R^{\mathcal{I}} \rightarrow v \in C^{\mathcal{I}}]\} \\
(C_1 \sqcap \ldots \sqcap C_n)^{\mathcal{I}} & = & C_1^{\mathcal{I}} \cap \ldots \cap C_n^{\mathcal{I}}
\end{array}
$$

A concept C is *satisfied* by an interpretation $\mathcal{I}$ if $C^{\mathcal{I}} \neq \emptyset$, analogously for roles. $\mathcal{I}$ is called a model for C in this case. A concept inclusion $C \sqsubseteq D$ (concept definition $C \equiv D$) is satisfied in $\mathcal{I}$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ (resp. $C^{\mathcal{I}} = D^{\mathcal{I}}$) and a role inclusion $R \sqsubseteq S$ (role definition $R \equiv S$), if $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ (resp. $R^{\mathcal{I}} = S^{\mathcal{I}}$). Similarly, assertions $C(a)$ and $R(a, b)$ are satisfied in $\mathcal{I}$, if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ resp. $(a, b)^{\mathcal{I}} \in R^{\mathcal{I}}$. If an interpretation $\mathcal{I}$ satisfies all axioms of $\mathcal{T}$ resp. assertions in $\mathcal{A}$, it is called a *model* of $\mathcal{T}$ resp. $\mathcal{A}$. If it satisfies both $\mathcal{T}$ and $\mathcal{A}$, it is called a model of Σ. Finally, if there is a model of Σ (i.e., a model for $\mathcal{T}$ and $\mathcal{A}$), then Σ is called satisfiable.

We are now able to define the entailment relation $\models$. A DL knowledge base Σ *logically entails* an assertion α (symbolically $\Sigma \models \alpha$) if α is satisfied by all models of

Σ. For an Abox $\mathcal{A}$, we say $\Sigma \models \mathcal{A}$ if $\Sigma \models \alpha$ for all $\alpha \in \mathcal{A}$. If α is of the form $C(i)$ then i is said to be an instance of C.

Closed-World Assumption (CWA): Databases employ the closed-world assumption where only the positive facts are given and consequently, negative facts are implicit. In addition, based on the closed-world-assumption, a database is assumed to be complete with respect to the constants of the domain [Rei87]. The same idea can also be applied to knowledge bases. Let us consider a knowledge base Σ. An assertion $\neg C(i)$ is considered as satisfied in all models if $C(i)$ is not entailed by Σ.

$$\Sigma \models_{CWA} \neg C(i) \text{if } \Sigma \not\models_{CWA} C(i) \tag{2.1}$$

Thus, i is found to be an instance of $\neg C$ under the closed-world assumption. This is relevant for query answering, which is discussed in the next subsection. Before discussing this, we mention the open-world assumption.

Open-World Assumption (OWA): Description logics employ the open-world assumption [BCM$^+$03]: Given a knowledge base Σ, answering queries in description logics only involves axioms/assertions which are entailed by Σ, and the number of domain objects is unbounded. The open-world assumption is useful in order to avoid problems of non-monotonicity in description logics.

2.1.2 Extensions: Queries, and Rules

In order to define queries and rules, a few definitions are required. Section 2.1.2 is mostly taken from [EKM09].

A *variable* is a string of characters of the form {A...Z}{a...z}*. In the following definitions, we denote variables or places where variables can appear with uppercase letters. Let V be a set of variables, and let $\underline{X}, \underline{Y_1}, \ldots, \underline{Y_n}$ be sequences $\langle \ldots \rangle$ of variables from V. The notation $\underline{z}$ denotes a sequence of individuals. We consider sequences of length 1 or 2 only if not indicated otherwise, and assume that $(\langle X \rangle)$ is to be read as (X) and $(\langle X, Y \rangle)$ is to be read as (X, Y) etc. Furthermore, we assume that sequences are automatically flattened. A function *as_set* turns a sequence into a set in the obvious way.

A *variable substitution* $\sigma = [X \leftarrow i, Y \leftarrow j, \ldots]$ w.r.t. a specified Abox is a mapping from variables to individuals mentioned in the Abox. The application of a variable substitution σ to a sequence of variables $\langle X \rangle$ or $\langle X, Y \rangle$ is defined as $\langle \sigma(X) \rangle$ or $\langle \sigma(X), \sigma(Y) \rangle$, respectively, with $\sigma(X) = i$ and $\sigma(Y) = j$. In this case, a sequence of individuals is derived. If a substitution is applied to a variable X for which there is no mapping $X \leftarrow k$ in σ then the result is undefined. A substitution for which all required mappings are defined is called *admissible* (w.r.t. the context).

Grounded Conjunctive Queries

Let $\underline{X}, \underline{Y_1}, \ldots, \underline{Y_n}$ be sequences of variables, and let $Q_1, \ldots, Q_n$ denote concept or role names. A query is defined by the following syntax: $\{(\underline{X}) \mid Q_1(\underline{Y_1}), \ldots, Q_n(\underline{Y_n})\}$. The sequence $\underline{X}$ may be of arbitrary length, but all variables mentioned in $\underline{X}$ must also appear in at least one of the $\underline{Y_1}, \cdots, \underline{Y_n}$: $as\_set(\underline{X}) \subseteq as\_set(\underline{Y_1}) \cup \cdots \cup as\_set(\underline{Y_n})$. Informally speaking, $Q_1(\underline{Y_1}), \ldots, Q_n(\underline{Y_n})$ defines a conjunction of so-called *query atoms* $Q_i(\underline{Y_i})$.

The list of variables to the left of the sign | is called the *head* and the atoms to the right are called the query *body*. Answering a query with respect to a knowledge base Σ means finding admissible variable substitutions σ such that

$$\Sigma \models \{\sigma(Q_1(\underline{Y_1})), \ldots, \sigma(Q_n(\underline{Y_n}))\}$$

We say that a variable substitution $\sigma = [X \leftarrow i, Y \leftarrow j, \ldots]$ introduces *bindings* $i, j, \ldots$ for variables $X, Y, \ldots$. Substitutions are defined w.r.t. a specified Abox. Given all possible variable substitutions σ, the *result* of a query is defined as $\{(\underline{z}) \mid \underline{z} = \sigma(\underline{X})\}$. Note that variables range over named domain objects only, and thus the queries used here are called grounded conjunctive queries.

Let us consider Abox $\mathcal{A}$ given in Table 2.2. Additionally, let us assume the next conjunctive query which is used to retrieve the *causes* relation among Car and $DoorSlam$ instances:

$$q_1 = \{(X, Y) \mid Car(X), causes(X, Y), DoorSlam(Y)\}$$

The answer to the query q_1 is:

$$\{(c_1, ds_1), (c_2, ds_2)\}$$

For answering the query q_1, we have the following substitutions which define the bindings for X, and Y:

$$X \leftarrow c_1, Y \leftarrow ds_1$$
$$X \leftarrow c_2, Y \leftarrow ds_2$$

A *Boolean* query is a query with $\underline{X}$ being of length zero. If for a Boolean query there is a variable substitution σ such that

$$\Sigma \models \{\sigma(Q_1(\underline{Y_1})), \ldots, \sigma(Q_n(\underline{Y_n}))\}$$

holds, we say that the query is answered with *true*, otherwise the answer is *false*.

Let us consider again Abox $\mathcal{A}$ given in Table 2.2. An example for a Boolean query is:

$$q_2 = \{() \mid CarEntry(X), hasEffect(X, Y), DoorSlam(Y)\}$$

For answering the query q_2, we have the following substitutions which define the bindings for X, and Y:

$$X \leftarrow ind_{42}, Y \leftarrow ds_1$$

Since for the query q_2 the variable substitution σ exists, the query is answered with true.

Later on, we will have to convert query atoms into Abox assertions. This is done with the function *transform*. The function *transform* applied to a set of query atoms $\{\gamma_1, \ldots \gamma_n\}$ is defined as:

$$\{transform(\gamma_1, \sigma), \ldots, transform(\gamma_n, \sigma)\}$$

where $transform(P(\underline{X}), \sigma) := P(\sigma(\underline{X}))$.

Answering grounded conjunctive queries is supported by DL reasoners, e.g., RacerPro [HM01], KAON2 [HMS04], and Pellet [SP06].

Rules

A rule r has the following form

$$P(\underline{X}) \leftarrow Q_1(\underline{Y_1}), \ldots, Q_n(\underline{Y_n}) \tag{2.2}$$

where P, $Q_1, \ldots, Q_n$ denote concept or role names with the additional restriction (safety condition) that $as\_set(\underline{X}) \subseteq as\_set(\underline{Y_1}) \cup \cdots \cup as\_set(\underline{Y_n})$. Rules are used to derive new Abox assertions, and we say that a rule r is *applied* to an Abox $\mathcal{A}$.

Let S be the set of substitutions σ such that the answer to the query

$$\{() \mid Q_1(\sigma(\underline{Y_1})), \ldots, Q_n(\sigma(\underline{Y_n}))\}$$

is *true* with respect to $\Sigma \cup \mathcal{A}$.[1] The function call $apply(\Sigma, P(\underline{X}) \leftarrow Q_1(\underline{Y_1}), \ldots, Q_n(\underline{Y_n}), \mathcal{A})$ returns a set of Abox assertions

$$\bigcup_{\sigma \in S} \{\sigma(P(\underline{X}))\}$$

for all $\sigma \in S$. The application of a set of rules $\mathcal{R} = \{r_1, \ldots r_n\}$ to an Abox is defined as follows:

$$apply(\Sigma, \mathcal{R}, \mathcal{A}) = \bigcup_{r \in \mathcal{R}} apply(\Sigma, r, \mathcal{A})$$

The result of $forward\_chain(\Sigma, \mathcal{R}, \mathcal{A})$ is defined to be $\emptyset$ if $apply(\Sigma, \mathcal{R}, \mathcal{A}) \cup \mathcal{A} = \mathcal{A}$ holds. Otherwise the result of *forward_chain* is determined by the recursive call

$$apply(\Sigma, \mathcal{R}, \mathcal{A}) \cup forward\_chain(\Sigma, \mathcal{R}, \mathcal{A} \cup apply(\Sigma, \mathcal{R}, \mathcal{A})).$$

For some set of rules $\mathcal{R}$ we extend the entailment relation by specifying:

$$(\mathcal{T}, \mathcal{A}) \models_{\mathcal{R}} \mathcal{A}_0 \quad \textit{iff} \quad (\mathcal{T}, \mathcal{A} \cup forward\_chain((\mathcal{T}, \emptyset), \mathcal{R}, \mathcal{A})) \models \mathcal{A}_0 \tag{2.3}$$

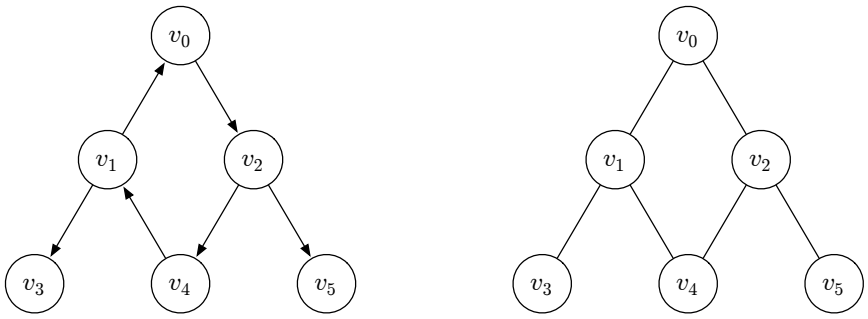
[1] We slightly misuse notation in assuming $(\mathcal{T}, \mathcal{A}) \cup \Delta = (\mathcal{T}, \mathcal{A} \cup \Delta)$. If $\Sigma \cup \mathcal{A}$ is inconsistent the result is well-defined but useless.

2.2 Probabilistic Representation Formalisms

2.2.1 Basic Notions of Graph and Probability Theory

In this section, we present basic notions of graph and probability theory in order to fix the nomenclature required to explain probabilistic knowledge representation formalisms based on graphs. The following definitions of graphs are based on the notations used in [Kna11]. Section 2.2.1 is taken from [GMN+08, GMN+09a, GMN+10c].

Graph: A graph $G = (V, E)$ is composed of a finite set of vertices V and a finite set of edges $E \subseteq V \times V$. Thus, $E = \{(v_i, v_j) \mid \text{for some } v_i, v_j \in V\}$, representing the fact that there are edges from v_i to v_j, respectively. G is said to be **directed** (Figure 2.1a), if for each v_i, v_j it holds that $(v_i, v_j) \in E$ does not imply $(v_j, v_i) \in E$ and, analogously, G is said to be **undirected** (Figure 2.1b), if for each v_i, v_j it holds that $(v_i, v_j) \in E$ implies $(v_j, v_i) \in E$. Directed edges are depicted with arrows and undirected edges with simple lines. Figure 2.1a shows an example of a directed graph and its undirected counterpart. $V = \{v_0, \ldots, v_5\}$, $E = \{(v_0, v_2), (v_1, v_0), (v_1, v_3), (v_2, v_4), (v_2, v_5), (v_4, v_1)\}$ for the directed graph, and the set of edges for the corresponding undirected graph results by adding all tuples to E needed in order to ensure symmetry).



(a) Directed graph (b) Undirected graph

Figure 2.1: Example of a directed and an undirected graph

A path P in a directed graph $G = (V, E)$ is a sequence $v_0, \rightsquigarrow v_1 \rightsquigarrow \ldots \rightsquigarrow v_n$ of vertices from V where $(v_i, v_{i+1}) \in E$ for all $0 \leq i < n$. An example for a path in Figure 2.1a is $v_0 \rightsquigarrow v_2 \rightsquigarrow v_4 \rightsquigarrow v_1 \rightsquigarrow v_3$.

A cycle C is a path $P = v_i \rightsquigarrow v_j \rightsquigarrow \ldots \rightsquigarrow v_i$ which begins and ends with the same vertex. In Figure 2.1 in both graphs there is the cycle $C_1 = v_0 \rightsquigarrow v_2 \rightsquigarrow v_4 \rightsquigarrow v_1 \rightsquigarrow v_0$. In the undirected graph there is the additional cycle $C_2 = v_0 \rightsquigarrow v_1 \rightsquigarrow v_4 \rightsquigarrow v_2 \rightsquigarrow v_0$. A graph G is called cyclic if it contains at least one cycle, otherwise it is called acyclic.

A function $Parents : V \to V$ maps a vertex $v \in V$ to a set $W = \{w \mid (w, v) \in E\}$. Clearly $W \subseteq V$. In Figure 2.1a, $Parents(v_2) = \{v_0\}$.

The basic notion of probabilistic knowledge representation formalisms is the so-called *random experiment*. In the following, we define the basic probability notations based on the notation used in [RN03].

Random Variable: A *random variable* X is a function assigning a value to the result of a random experiment. A random experiment is represented by a so-called sample space. Random variables are functions without arguments, and they return different values at different points of time. There are two types of random variables: discrete and continuous ones. Unlike continuous random variables, which provide a map to real numbers, discrete random variables provide a map to a finite number of distinct values. In this work, we consider only discrete random variables.

Possible values of a random variable comprise the so-called *domain* of the random variable. A random variable X is called binary if $dom(X)$ contains 2 values. It is called Boolean if $dom(X) = \{true, false\}$. Otherwise, it is called a multi-valued random variable. For instance, the domain of a discrete random variable $WaterPollution$ has the following values which indicate the water pollution level: $dom(WaterPollution) = \{low, medium, high\}$.

Event: An event is a set of possible outcomes of a random experiment, i.e., a subset of the sample space. Let $\vec{X} = \{X_1, ..., X_n\}$ be the ordered set of all random variables of a random experiment. An atomic event, denoted by $\vec{X} = \vec{x}$, is an assignment, i.e., a mapping definition, $X_1 = x_1, ..., X_n = x_n$ for all random variables. Atomic events represent single outcomes for each random variable $\{X_1, ..., X_n\}$, and are also called (possible) worlds. For example, an atomic event from the above example using only the random variable $WaterPollution$ is $WaterPollution = low$.

With the obvious meaning (complex) events are denoted using propositional logic formulae involving multiple atomic events, possibly for different random variables. In case of an event with a Boolean random variable X, we write x as an abbreviation for $X = true$ and $\neg x$ as an abbreviation for $X = false$.

Probability: A possible world can be associated with a *probability value* or *probability* for short. A (prior) probability or unconditional probability of an event $X = x$ is the chance that random variable X takes value x in a random experiment. The probability value of an event is denoted as

$$P(event)$$

Mappings of events to probabilities (or assignment of probabilities to events) are specified with so-called *probability assertions* using the syntax

$$P(event) = p$$

where p is a real value between 0 and 1. Probability values must be assigned such that the Kolmogorov axioms [Kol50] hold. For instance, the probability of the event

$X = x$ is denoted as $P(X = x)$. Analogously for complex events. We use the term prior probability if no other information about random variable X is given.

A set of probabilistic assertions is called a *probabilistic knowledge base* (with signature $\vec{X}$ if $\vec{X}$ contains all random variables mentioned in the assertions). For example, $P(OilPollution = true) = 0.1$ indicates that the probability of the event $OilPollution = true$ is 0.1.

Probability Distribution: A total mapping from the domain of a random variable X to probability values $[0, 1]$ is called a *distribution*. For distributions we use the notation

$$\mathbf{P}(X) \text{ or } \mathbf{P}(X_1, \ldots, X_n)$$

if distributions are to be denoted for (ordered) sets of random variables.

For specifying a distribution, probability assertions for *all* domain values must be specified, and the values p_i $(1 \leq i \leq m)$ with $m = |dom(X)|$ must sum up to 1.

$$\mathbf{P}(X) = \langle p_1, \ldots, p_m \rangle^T$$

For joint distributions, we have the following

$$\mathbf{P}(X_1, \ldots, X_n) = \langle p_1, \ldots, p_m \rangle^T$$

with $m = |dom(X_1) \times \ldots \times dom(X_n)|$ in this case.

Let us consider an example. Assume the probability distribution:

$$\mathbf{P}(WaterPollution) = \langle 0.5, 0.3, 0.2 \rangle$$

This means, the associated probabilities to the events are, respectively:

$$P(WaterPollution = low) = 0.5$$
$$P(WaterPollution = medium) = 0.3$$
$$P(WaterPollution = high) = 0.2$$

In other words, the latter three probability assertions state the same knowledge.

Full Joint Probability Distribution: In case all random variables of a random experiment are involved in a distribution, we speak of a *full joint probability distribution* (JPD), otherwise the expression is said to denote a *joint distribution* or a *marginal distribution* (projection of the n-dimensional space of probability values to a lower-dimensional space with m dimensions). Let us assume n random variables $X_1, \ldots, X_n$ are specified in the signature of a probabilistic knowledge base. Consequently $\mathbf{P}(X_1, \ldots, X_n)$ is a full joint probability distribution. The expression

$$\mathbf{P}(X_1, \ldots, X_m, X_{m+1} = x_{m+1}, \ldots, X_l = x_l)$$

denotes an m-dimensional distribution with known values $x_{m+1}, \ldots, x_l$ for random variables $X_{m+1} \ldots X_l$. In slight misuse of notation, we sometimes write $\vec{E} = \vec{e}$ or even just $\vec{e}$ for these events (e stands for evidence). The fragment $\vec{e}$ need not necessarily be written at the end in the parameter list of $\mathbf{P}$.

Conditional Probability: A conditional probability or posterior probability $P(X = x|Y = y)$ is the probability of event $X = x$ under the condition that event $Y = y$ might take place $P(Y = y) > 0$. This is defined as follows:

$$P(X = x|Y = y) = \frac{P(X = x \wedge Y = y)}{P(Y = y)} \tag{2.4}$$

In distribution form we have

$$\mathbf{P}(X|Y) = \frac{\mathbf{P}(X, Y)}{\mathbf{P}(Y)}$$

where a faction of distributions is to be read as a vector of fractions for all values from $dom(X) \times dom(X)$ computed as indicated above. The distribution is called conditional probability distribution.

Continuing the environmental example, we consider the probability of $Flood = true$ given $Rain = true$, which is indicated by:

$$P(Flood = true|Rain = true)$$

where $Flood$ and $Rain$ are binary random variables. This probability is determined by

$$P(Flood = true|Rain = true) = \frac{P(Flood = true \wedge Rain = true)}{P(Rain = true)}$$

Combined forms are also possible for conditional probabilities: $\mathbf{P}(X_1, ..., X_m \mid \vec{e})$ is defined as:

$$\frac{\mathbf{P}(\vec{X}, \vec{e})}{P(\vec{e})} \tag{2.5}$$

The semantics and algebra of these expressions should be obvious given what is explained above.

Probabilistic Inference Problems and their Algorithms: For a probabilistic knowledge base, formal inference problems are defined. We restrict our attention to the conditional probability query. A *conditional probability query* is a denotation for the joint distribution of a set of m random variables out of a set X of n random variables conditioned on $\vec{e}$ and is denoted with $P_X(x_1 \wedge \ldots \wedge x_m \mid \vec{e})$ where $vars(x_1, \ldots, x_m) \cap vars(\vec{e}) = \emptyset$ and $vars(x_1, \ldots, x_m) \cup vars(\vec{e}) \subseteq X$ with $vars$ specified in the obvious way. In this context x_i indicates $X_i = x_i$. We also have the distribution form of the above query: $\mathbf{P}_X(X_1, \ldots, X_m \mid \vec{e})$. If the set of random variables X is known from the context, the subscript $_X$ is often omitted. For solving problems of this kind, mostly two approaches are discussed in the literature (e.g. [RN03]), exact inference and approximate inference.

Exact algorithms for solving this problem work by summing out all hidden random variables [RN03]. Therefore, exact inference is known to be highly intractable even in

the case of large numbers of independence assumptions (see the next section). Since probability distributions applied to problems in the real world can be very complex, with probabilities varying greatly over a high-dimensional space, there may be no way to sensibly characterize such distributions analytically [Nea93]. Thus, the combinatorial combination of probability values provides for long runtimes in practice.

With *sampling algorithms* it is possible to mitigate these kinds of problems. Instead of summing out all hidden random variables, the primitive element in any sampling algorithm is the generation of samples from a known probability distribution [RN03]. For example, instead of computing all possible outcomes of a complex experiment with coin throws, the idea is to "flip" the coin itself a number of times.

Even with sampling algorithms specifying a full joint distribution requires an exponential number of probabilistic assertions. In the next section, this problem is solved by exploiting another form of knowledge, namely probabilistic independence assertions, which are specified as

$$\mathbf{P}(X_i|X_{i_1},\ldots,X_{i_k}) = \mathbf{P}(X_i|X_{j_1},\ldots,X_{j_m})$$

with $\{i_1,\ldots,i_k\} \subseteq \{j_1,\ldots,j_m\} \subseteq N \setminus \{i\}$

Even exact inference can be dramatically improved if independence assumptions are considered for solving inference problems. While this can be exploited for summing-out techniques as well, it can also be exploited for reducing the number of probability values to be specified in a knowledge base. This is discussed in the next section in the context of so-called probabilistic knowledge representation formalisms. We will also discuss sampling techniques in this context.

In following, we introduce two probabilistic knowledge representation formalisms, namely Bayesian networks [Pea88] and Markov logic [DR07]. Using several examples from the environmental domain, advantages and disadvantages of each formalism are discussed.

2.2.2 Bayesian Networks

Bayesian networks [Pea88] are one of the frameworks for effectively answering queries with respect to probabilistic knowledge bases. They are used in many real-world applications including diagnosis, forecasting, automated vision, sensor fusion and manufacturing control. In the next sections, syntax and semantics of Bayesian networks are discussed. Section 2.2.2 is taken from [GMN+08]. The following discussion and notations are according to [RN03].

Syntax: A Bayesian network $BN = (G, \gamma)$ is defined by a directed acyclic graph $G = (V, E)$ and a function $\gamma : V \to T$ which maps a vertex $v \in V$ to a conditional probability distribution $T_i \in T$. Note that a vertex v indicates a random variable X_i and an edge $e_{ij} = (v_i, v_j)$ represents a direct influence of a parent vertex on a child vertex v_i. We use the function $Parents_E(\cdot)$ to denote the set $\{Y \mid (Y, X) \in E\}$,

and omit the subscript if the graph is clear from context. A conditional probability distribution T_i is given as $\mathbf{P}(X_i|Parents(X_i))$. If $Parents(X_i) = \emptyset$, then T_i specifies a prior probability. $\mathbf{P}(X_i|Parents(X_i))$ is usually specified as a table $\gamma(X_i)$ (see below for examples). $\mathbf{P}(X_i|Parents(X_i))$ is also called conditional probability table (CPT).

The structure of a Bayesian network is determined by the insertion order of the vertices [RN03]. During the network construction, vertices are added to the network individually. After adding a vertex, conditional dependencies of the new vertex to the vertices of the current network are to be determined. If there are dependencies, incoming edges to the new vertex are added accordingly. Each incoming edge comes from a previously added vertex which has conditional dependency to the new vertex. On the other hand, for instance, two sibling nodes are conditionally independent given the value of the father node is known (for other examples, see below).

Semantics: The semantics of a Bayesian network $BN = ((\{X_1, \ldots, X_n\}, E), \gamma)$ can be seen in two different ways [RN03]:

1. The first semantics of a Bayesian network indicates that the structure of a Bayesian network shows the conditional independence relationships which hold among the variables in the domain.

2. Bayesian networks are representations of full joint probability distributions of domain variables $\{X_1, \ldots, X_n\}$ (Equation 2.6).

$$\mathbf{P}_{BN}(X_1, \ldots, X_n) = \prod_{i=1}^{n} \mathbf{P}(X_i|Parents_E(X_i)) \tag{2.6}$$

where $\mathbf{P}(X_i|Parents_E(X_i)) = \gamma(X_i)$. The joint probability distribution of a set of variables is the product of their conditional probability distributions. But, in contrast to what the chain rule defines, the result simplifies dramatically in the presence of conditional independence assumptions.

Example 1 In this example we consider two events which influence the human health, namely air pollution and noise pollution. One of the factors which affects air- and noise pollution is traffic jam. These relationships are modelled by the Bayesian network graph in Figure 2.2:

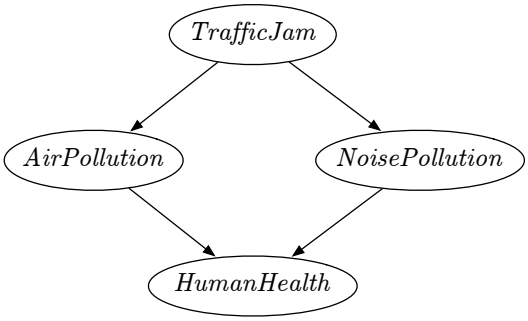
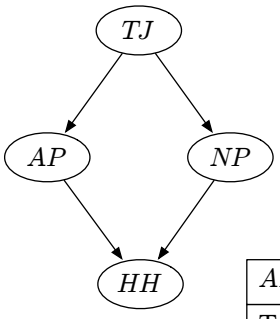


Figure 2.2: Example of a Bayesian network.

TrafficJam and *HumanHealth* are conditionally independent given *AirPollution* and *NoisePollution* therefore there is no link between them. Similarly *AirPollution* and *NoisePollution* are conditionally independent given *TrafficJam*. Figure 2.3 depicts the above Bayesian network with conditional probability distributions. The variables *TJ*, *AP*, *NP* and *HH* stand for *TrafficJam*, *AirPollution*, *NoisePollution* and *HumanHealth*, respectively. All random variables are binary.

$P(TJ = true)$
0.3



TJ	$P(NP = true \mid TJ)$
T	0.7
F	0.1

TJ	$P(AP = true \mid TJ)$
T	0.6
F	0.1

AP	NP	$P(HH = true \mid AP,NP)$
T	T	0.1
T	F	0.3
F	T	0.2
F	F	0.6

Figure 2.3: Example of a Bayesian network with CPTs $\gamma(X)$ associated.

Since *TJ* has no parents, only prior probabilities are assigned to them. The probability of *TJ=true* is 0.3. Consequently, the probability of *TJ=false* is 0.7. Since *AP* has only one parent, its conditional probability distribution has two rows. For example, the first row means that the probability of *AP=true* is 0.6 if *TJ* is true, and analogously $P(AP = true)$ is 0.1 in case *TJ=false*. *HH* has two parents, consequently its

conditional probability table has four rows. The first row means that the probability of *HH=true* is 0.1 if AP and NP are both *true* and so on.

An example for an entry in the full joint distribution is $P(\neg hh \wedge ap \wedge np \wedge tj)$ which is computed based on Equation 2.6 as follows:

$$\begin{aligned} P(\neg hh \wedge ap \wedge np \wedge tj) &= P(tj)P(ap|tj)P(np|tj)P(\neg hh|ap \wedge np) \\ &= 0.3 \times 0.6 \times 0.7 \times 0.9 \\ &= 0.1134 \end{aligned}$$

This example shows how a Bayesian network is constructed. Additionally, it demonstrates how a full joint probability is determined according to the conditional probability tables.

Inference in Bayesian networks

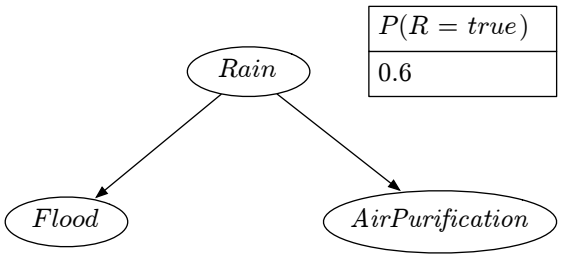
In this section, we discuss the problem of computing the (posterior) probability distribution $\mathbf{P}(X|\vec{E} = \vec{e})$ for a so-called query variable X given a set of evidence variables and their values, i.e., $\vec{E} = \{E_1, E_2, ...\}$ where $\vec{e}$ is a tuple of particular observed values. In addition to X and $\vec{E}$, there is a set of non-evidence variables $\vec{Y} = \{Y_1, Y_2, ...\}$ which are considered in the solution of the inference problem. There are two main solutions for the inference problem, namely exact inference and approximate inference. Our discussion is taken from [RN03].

Exact inference: The exact inference solves the inference problem by the full joint distribution:

$$\mathbf{P}(X|\vec{E} = \vec{e}) = \alpha\, \mathbf{P}(X, \vec{E} = \vec{e}) = \alpha \sum_{\vec{y} \in dom(\vec{Y})} \mathbf{P}(X, \vec{E} = \vec{e}, \vec{Y} = \vec{y}) \tag{2.7}$$

where the summation is over all possible values of non-evidence (hidden) variables $\vec{Y}$. In the above equation, α denotes a normalization constant. With this notation, the above equation can be written as a full joint distribution. This simple algorithm for exact query answering for a Bayesian network with n Boolean variables is exponential in the number of (hidden) variables. In general, probabilistic inference in Bayesian networks is NP-hard [Coo90], and this result shows that, in general, exact inference for large networks is intractable. In the following, an example for exact inference with the simple algorithm specified in Formula 2.7 is given for illustration purposes.

Figure 2.4 depicts a Bayesian network where the relationships between rain, flood and air purification are given. In the conditional probability distributions R, F and AP stand for the corresponding random variables *Rain*, *Flood* and *AirPurification*, respectively, which are all binary variables. Since *Flood* and *AirPurification* are conditionally independent, there is no edge between them:



$P(R = true)$
0.6

R	$P(F = true \mid R)$
T	0.6
F	0.1

R	$P(AP = true \mid R)$
T	0.8
F	0.4

Figure 2.4: Example of a Bayesian network

The next table depicts the full joint distribution for the three Boolean random variables where *airPurif* indicates *airPurification*:

	$flood$		$\neg flood$	
	airPurif	$\neg$*airPurif*	*airPurif*	$\neg$*airPurif*
$rain$	0.288	0.072	0.192	0.048
$\neg rain$	0.016	0.024	0.144	0.216

Table 2.3: A full joint distribution for *Rain*, *Flood* and *AirPurification* world

In Equation 2.8, we compute the probability of $Rain = true$ given $Flood = true$, where *Rain* is a query variable and $Flood$ is an evidence variable:

$$P(rain|flood) = \frac{P(rain \wedge flood)}{P(flood)} = \frac{0.288 + 0.072}{0.288 + 0.072 + 0.016 + 0.024} = 0.9 \qquad (2.8)$$

Similarly, we compute the probability of *Rain=false* given $Flood = true$:

$$P(\neg rain|flood) = \frac{P(\neg rain \wedge flood)}{P(flood)} = \frac{0.016 + 0.024}{0.288 + 0.072 + 0.016 + 0.024} = 0.1 \qquad (2.9)$$

The term $1/P(Flood = true)$ in Equations 2.8 and 2.9 is a normalization constant, which causes the sum of the above probabilities to be set to one. The non-evidence variable in the above inference is *AirPurification*. If we use probability distributions,

the above equations can be written in a single equation:

$$\begin{aligned}\mathbf{P}(Rain|flood) &= \alpha\,\mathbf{P}(Rain, flood)\\ &= \alpha[\mathbf{P}(Rain, flood, airPurification) +\\ &\quad\ \mathbf{P}(Rain, flood, \neg airPurification)]\\ &= \alpha\,[\langle 0.288, 0.016\rangle + \langle 0.072, 0.024\rangle]\\ &= \alpha\,[\langle 0.36, 0.04\rangle] = \langle 0.9, 0.1\rangle\end{aligned}$$

This example shows how the conditional probabilities in a Bayesian network are determined according to exact inference.

Approximate Inference: Since the complexity of exact inference for large networks is very high, approximate inference methods have been developed. These methods utilize the generation of samples for the considered random variables. The accuracy of sampling methods depends on the number of samples. It means, generating more samples leads to higher accuracy and consequently the result converges to the result of exact inference. In the literature (e.g. [RN03]), many approximate inference methods for Bayesian networks are defined. Due to space constraints, we do not describe them in this work.

Advantages and Disadvantages of Bayesian Networks

Bayesian networks [Pea88] are one of the best-understood models for effectively representing the joint probability distribution of a domain. Using acyclic graphs, Bayesian networks provide a compact representation for conditional independence assumptions. Despite these interesting properties, Bayesian networks have the following disadvantages. If Bayesian networks are used for modeling causality, humans can understand local conditional probability distributions quite well. However, due to the fact that Bayesian networks are acyclic, modeling arbitrary joint probability distributions also poses quite substantial challenges for providing sensible values for (non-causal) dependencies and resulting CPTs.

In Bayesian networks, it is not easily possible to refer to objects and their relations. For example, we cannot refer to the *TrafficJam* in a particular city like Hamburg or a *Flood* in Berlin. Bayesian networks of the kind discussed so far are inherently propositional. Many extensions of Bayesian networks have been proposed in the literature (see also [RN03] for entry points). Rather than introducing first-order Bayesian network modeling approaches (e.g. [KP97, Las08]), we now directly introduce an approach that avoids acyclicity restrictions and provides means for first-order modeling.

2.2.3 Markov Logic Networks

The formalism of Markov logic [DR07] provides a means to combine the expressivity of first-order logic [RN03] augmented with the formalism of Markov networks [Pea88].

The Markov logic formalism uses first-order logic to define "templates" for constructing Markov networks. Section 2.2.3 is mostly taken from [GMN+08, GMN+09a, GMN+10c].

A Markov logic network $MLN = (\mathcal{F}_{MLN}, \mathcal{W}_{MLN})$ [DR07] consists of a sequence of first-order formulas $\mathcal{F}_{MLN} = \langle F_1, ..., F_m \rangle$ and a sequence of real number weights $\mathcal{W}_{MLN} = \langle w_1, ..., w_m \rangle$. The association of a formula to its weight is by position in the sequence. For a formula $F \in \mathcal{F}_{MLN}$ with associated weight $w \in \mathcal{W}_{MLN}$ we also write $w\,F$ (weighted formula). Thus, a Markov logic network can also be defined as a set of weighted formulas. Both views can be used interchangeably. As a notational convenience, for ordered sets we nevertheless sometimes write $\vec{X}, \vec{Y}$ instead of $\vec{X} \cap \vec{Y}$.

In contrast to standard first-order logics such as predicate logic, relational structures not satisfying a formula F_i are not ruled out as models. If a relational structure does not satisfy a formula associated with a large weight it is just considered to be quite unlikely the intended one.

Let us consider the universally quantified formula:

$$\forall x\ CityWithTrafficJam(x) \rightarrow CityWithAirPollution(x)$$

The above formula might be true in some relational structures, but might be false in others. By assigning a reasonable weight to a formula, a formula can become a "soft constraint" allowing some relational structures not satisfying this formula to be still considered as possible models (possible worlds). In other words, the relational structure in question corresponds to a world associated with a non-zero probability.

Let $C = \{c_1, ..., c_m\}$ be the set of all constants mentioned in $\mathcal{F}_{MLN}$. A *grounding* of a formula $F_i \in \mathcal{F}_{MLN}$ is a substitution of all variables in the matrix of F_i with constants from C. From all groundings, the (finite) set of grounded atomic formulas (also referred to as *ground atoms*) can be obtained. Grounding corresponds to a domain closure assumption. The motivation is to get rid of the quantifiers and reduce inference problems to the propositional case. The Markov logic network [DR07] is composed of a set of nodes and edges which are defined based on the formulas F_i and the constants in the knowledge base. In the following, it is explained how the nodes and edges of a Markov logic network are defined [DR07]:

- One node for each possible grounding of each predicate appearing in MLN
- One edge between two nodes if and only if the corresponding ground predicates appear together in a grounding of a formula F_i in MLN

Since a ground atom can either be true or false in an interpretation (or world), it can be considered as a Boolean random variable X. Consequently, for each MLN with associated random variables $\vec{X}$, there is a set of possible worlds $\vec{x}$. In this view, sets of ground atoms are sometimes used to denote worlds. In this context, negated ground atoms correspond to *false* and non-negated ones to *true*. We denote worlds using a sequence of (possibly negated) atoms. Let us assume the constant *hamburg*.

An example world specification using this convention is:

$$\langle cityWithTrafficJam(hamburg), \neg cityWithAirPollution(hamburg)\rangle$$

So, there is traffic jam in Hamburg, but no air pollution. When a world $\vec{x}$ violates a weighted formula (does not satisfy the formula) the idea is to ensure that this world is less probable rather than impossible as in predicate logic. Note that weights do not directly correspond to probabilities (see [DR07] for details). For each possible world of a Markov logic network $MLN = (\mathcal{F}_{MLN}, \mathcal{W}_{MLN})$ there is a probability for its occurrence. The weights associated with the formulae define probabilistic knowledge, i.e., the weights associated with the formulas induce a probability distribution over the derived ground atoms. In the formalism of Markov networks the full joint probability distribution of a Markov logic network MLN is specified in symbolic form as:

$$\mathbf{P}_{MLN}(\vec{X}) = (P(\vec{X} = \vec{x}_1), \ldots, P(\vec{X} = \vec{x}_n)) \tag{2.10}$$

for every possible $\vec{x}_i \in \{true, false\}^n$, $n = |\vec{X}|$ and

$$P(\vec{X} = \vec{x}) \ := log\_lin_{MLN}(\vec{x}) \tag{2.11}$$

For a motivation of the log-linear form, see, e.g., [DR07], where log_lin being defined as

$$log\_lin_{MLN}(\vec{x}) \ = \ \frac{1}{Z}\, exp\,(\sum_{i=1}^{|\mathcal{F}_{MLN}|} w_i n_i(\vec{x})) \tag{2.12}$$

According to this definition, the probability of a possible world $\vec{x}$ is determined by the exponential of the sum of the number of true groundings (computed with the function n_i) of formulas $F_i \in \mathcal{F}_{MLN}$ in $\vec{x}$, multiplied with their corresponding weights $w_i \in \mathcal{W}_{MLN}$, and finally normalized with

$$Z = \sum_{\vec{x} \in \vec{X}} \exp\,(\sum_{i=1}^{|\mathcal{F}_{MLN}|} w_i n_i(\vec{x})), \tag{2.13}$$

the sum of the probabilities of all possible worlds. Thus, rather than specifying the full joint distribution directly in symbolic form as we have discussed before, in the Markov logic formalism, the probabilistic knowledge is specified implicitly by the weights associated with formulas. Determining these formulas and their weights in a practical context is all but obvious, such that machine learning techniques [LD07, DLK+08] are usually employed for knowledge acquisition.

A *conditional probability query for a Markov logic network* MLN is the computation of the joint distribution of a set of m events involving random variables conditioned on $\vec{e}$ and is denoted by:

$$P_{MLN}(x_1 \wedge \ldots \wedge x_m \mid \vec{e}) \tag{2.14}$$

where $\vec{e}$ indicates the evidence vector which contains a set of weighted and/or strict ground atoms of the predicates in the knowledge base. Note that the absence of a ground atom in the evidence vector means that the weight of this ground atom is zero. Consequently, the knowledge base in the Markov logic network [DR07] is not based on the closed world assumption. The semantics of this query is given as:

$$P_{rand\_vars(MLN)}(x_1 \wedge \ldots \wedge x_m \mid \vec{e}) \; w.r.t. \; \mathbf{P}_{MLN}(rand\_vars(MLN))$$

where

$$vars(x_1, \ldots, x_m) \cap vars(\vec{e}) = \emptyset$$

and

$$vars(x_1, \ldots, x_m) \subseteq rand\_vars(MLN)$$

The function $rand\_vars$ is defined as follows:

$rand\_vars((\mathcal{F}, \mathcal{W})) := \{A(\underline{C}) \mid A(\underline{C})$ is mentioned in some grounded formula $F \in \mathcal{F}\}$

Grounding is accomplished w.r.t. all constants that appear in $\mathcal{F}$ where A denotes atomic concept or atomic role. An algorithm for answering queries of the above form is investigated in [GM10].

Example 2 Let us consider the next weighted formulas and a constant *hamburg*:

$$0.5 \; \forall x \; CityWithTrafficJam(x) \Rightarrow CityWithAirPollution(x)$$
$$0.5 \; \forall x \; CityWithIndustry(x) \Rightarrow CityWithAirPollution(x)$$

Thus, $rand\_vars((\mathcal{F}, \mathcal{W}))$ is:

$$\begin{aligned} rand\_vars((\mathcal{F}, \mathcal{W})) \; = \; & \{CityWithTrafficJam(hamburg), \\ & CityWithAirPollution(hamburg), \\ & CityWithIndustry(hamburg)\} \end{aligned}$$

Furthermore, let us assume that the evidence vector is:

$$\vec{e} = \{CityWithIndustry(hamburg) = true\}$$

Thus, $vars(\vec{e})$ is:

$$vars(\vec{e}) = \{CityWithIndustry(hamburg)\}$$

Let us assume that the ground atoms denote Boolean random variables. The value of $CityWithIndustry(hamburg)$ is fixed by evidence. Thus, there are four possible worlds $W_1, \ldots, W_4$ left. In the next table, the probability of each world is determined according to the Markov logic formalism where $CWTJ$, $CWAP$, CWI, and h indicate

CityWithTrafficJam, $CityWithAirPollution$, *CityWithIndustry*, and *hamburg*, respectively:

Worlds W_i	$CWTJ(h)$	$CWAP(h)$	$CWI(h)$	$P(W_i)$
W_1	0	0	1	$\exp(0.5)/Z = 0.20$
W_2	0	1	1	$\exp(1)/Z = 0.34$
W_3	1	0	1	$1/Z = 0.12$
W_4	1	1	1	$\exp(1)/Z = 0.34$

Table 2.4: The possible worlds and their probabilities according to the Markov logic formalism.

The normalization constant Z is determined according to Equation 2.13:

$$Z = 1 + 2 \times \exp(1) + \exp(0.5)$$

As you can see in the above table, the sum of the worlds probabilities is equal to 1:

$$P(W_1) + \ldots + P(W_4) = 1$$

In the following, we extend the above example with the next formula:

$$1 \quad \forall x \, [CityWithAirPollution(x) \Rightarrow \exists y[Adjacent(x, y) \wedge CityWithAirPollution(y)]] \quad (2.15)$$

Additionally, we consider the new constant *berlin*. Thus, the possible ground atoms which are the nodes of the Markov network are defined as follows:

$$\begin{aligned} rand\_vars((\mathcal{F}, \mathcal{W})) \quad = \quad & \{CityWithTrafficJam(h), CityWithAirPollution(h), \\ & CityWithTrafficJam(b), CityWithAirPollution(b), \\ & CityWithIndustry(h), CityWithIndustry(b) \\ & Adjacent(h, h), Adjacent(h, b), \\ & Adjacent(b, h), Adjacent(b, b)\} \end{aligned}$$

where h and b denote *hamburg* and *berlin*, respectively. The possible groundings of Formula 2.15 are as follows:

$$\begin{aligned} CityWithAirPollution(h) \quad &\Rightarrow \quad [Adjacent(h, h) \wedge CityWithAirPollution(h)] \\ CityWithAirPollution(b) \quad &\Rightarrow \quad [Adjacent(b, h) \wedge CityWithAirPollution(h)] \\ CityWithAirPollution(h) \quad &\Rightarrow \quad [Adjacent(h, b) \wedge CityWithAirPollution(b)] \\ CityWithAirPollution(b) \quad &\Rightarrow \quad [Adjacent(b, b) \wedge CityWithAirPollution(b)] \end{aligned}$$

which leads to the following Markov network subgraph:

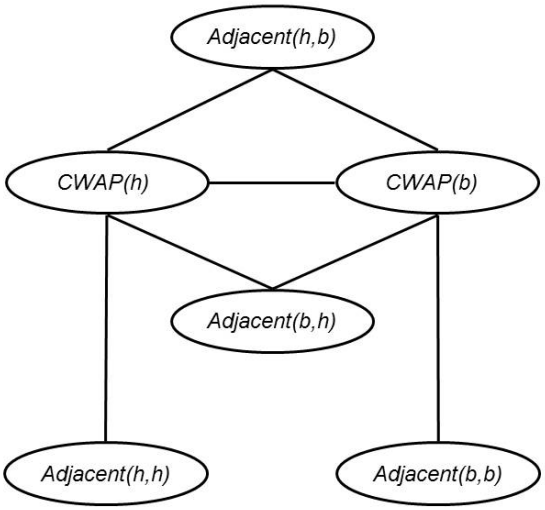


Figure 2.5: The Markov network subgraph of Example 2.

The next figure depicts the complete Markov network graph of this example:

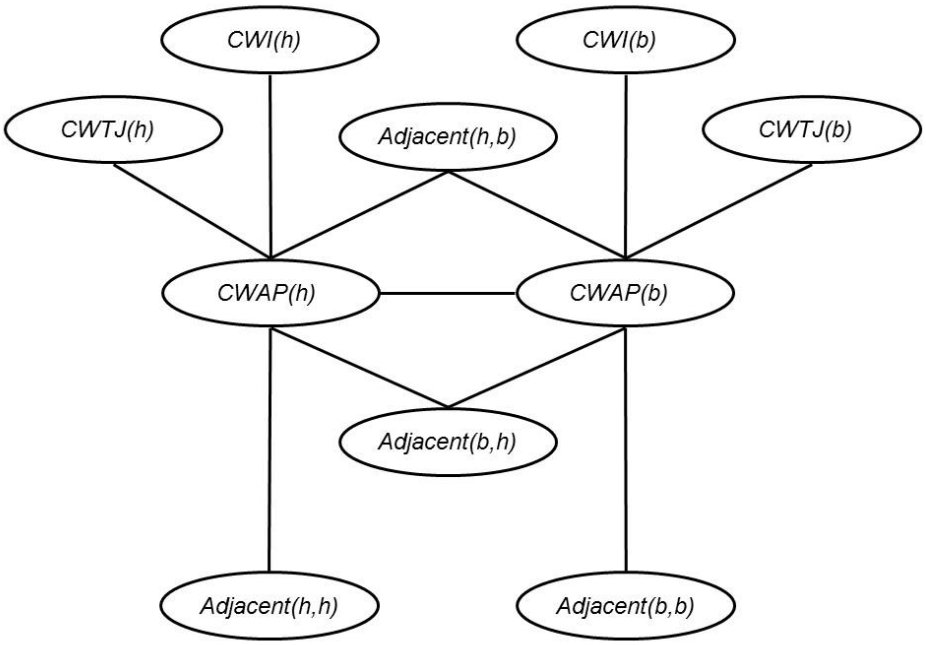


Figure 2.6: The Markov network graph of Example 2.

The above Markov network contains 10 binary ground atoms, and there are 2^{10} worlds. Since the truth value of the ground atom $CityWithIndustry(h)$ is known, the number

of possible worlds is reduced to 2^9. A possible world for this example is:

$$\begin{aligned}\vec{x} \;=\; \langle & cityWithTrafficJam(h), \neg cityWithAirPollution(h),\\ & \neg cityWithTrafficJam(b), cityWithAirPollution(b),\\ & cityWithIndustry(h), cityWithIndustry(b),\\ & \neg adjacent(h,h), adjacent(h,b),\\ & adjacent(b,h), adjacent(b,b)\rangle\end{aligned}$$

This example shows how a Markov logic network is constructed according to a set of formulas and a set of constants. Additionally, it demonstrates how the probability of a world is determined.

MAP Inference Problem in MLN

The Maximum A Posteriori (MAP) inference [SD05] returns the most-likely state of query atoms given the evidence. Based on the MAP inference the "most-probable world" given the evidence is determined as a set of events. The MAP inference problem given a distribution MLN for a set of random variables $X = rand\_vars(MLN)$ is formalized as follows:

$$MAP_X(\vec{e}) := \vec{e} \cup argmax_{\vec{y}} P_{MLN}(\vec{y} \mid \vec{e}) \tag{2.16}$$

where:

$$\begin{aligned} vars(\vec{y}) \cap vars(\vec{e}) &= \emptyset \\ vars(\vec{y}) \cup vars(\vec{e}) &= X \end{aligned}$$

In the MLN, the definition of the MAP problem [DLK+08] is rewritten as follows. The conditional probability term $P(\vec{x}|\vec{e})$ is replaced with the Markovian formula:

$$MAP_{MLN}(\vec{e}) := \vec{e} \cup argmax_{\vec{y}} \frac{1}{Z_e} \exp\left(\sum_i w_i n_i\left(\vec{y}, \vec{e}\right)\right) \tag{2.17}$$

Thus, for describing the most-probable world, MAP returns a set of events, one for each random variable used in the Markov network derived from MLN. In the above equation, $\vec{x}$ denotes the hidden variables, and Z_e denotes the normalization constant which indicates that the normalization process is performed over possible worlds consistent with the evidence $\vec{e}$. In the next equation, Z_e is removed since it is constant and it does not affect the *argmax* operation. Similarly, in order to optimize the MAP computation the exp function is left out since it is a monotonic function and only its argument has to be maximized:

$$MAP_{MLN}(\vec{e}) := \vec{e} \cup argmax_{\vec{y}} \sum_i w_i n_i\left(\vec{y}, \vec{e}\right) \tag{2.18}$$

The above equation shows that the MAP problem in Markov logic formalism is reduced to a new problem which maximizes the sum of weights of satisfied clauses [DLK+08]. Since the MAP determination in Markov networks is an **NP**-hard problem [Rot96], it is usually to be performed by approximate solvers in realistic domains. The most commonly used approximate solver is the MaxWalkSAT algorithm [KSJ97], a weighted variant of the WalkSAT [SKC96] local-search satisfiability solver. The MaxWalkSAT algorithm attempts to satisfy clauses with positive weights and keeps clauses with negative weights unsatisfied [KSR+10].

In this work, we apply the MAP operation [DLK+08] in order to remove inconsistencies in the observation Abox. By applying MAP to the weighted observation Abox, we map the probabilistic logic to the classical logic, i.e., the weighted Abox is changed to a non-weighted consistent Abox which describes several models. The next example shows the effect of applying MAP to an inconsistent Abox.

Example 3 Let us consider the following Tbox axioms:

$$\begin{aligned} DoorSlam &\sqsubseteq Audio \\ Applause &\sqsubseteq Audio \\ Car &\sqsubseteq \neg Audio \\ Car &\sqsubseteq \neg DoorSlam \\ Car &\sqsubseteq \neg Applause \\ DoorSlam &\sqsubseteq \neg Applause \end{aligned}$$

Furthermore, let us assume the following inconsistent Abox where ds_1 is an instance of the disjoint concepts $DoorSlam$ and $Applause$:

$$\mathcal{A} = \{1.3\ Car(c_1), 1.2\ DoorSlam(ds_1), causes(c_1, ds_1), 0.3\ Applause(ds_1)\}$$

In order to determine the most probable world according to the Tbox, the MAP operation is applied to the Abox $\mathcal{A}$. The result is the bit vector W:

$$W = \langle 1, 1, 1, 0, 1 \rangle$$

where each component of W corresponds to a ground atom of a vector G with

$$G = \langle Car(c_1), DoorSlam(ds_1), Causes(c_1, ds_1), Applause(ds_1), Audio(ds_1) \rangle$$

W indicates that there are four positive ground atoms and one negative ground atom denoted by one and zero, respectively. The selected world with the highest probability is the following world:

$$\langle car(c_1), doorSlam(ds_1), causes(c_1, ds_1), \neg applause(ds_1), audio(ds_1) \rangle$$

By considering the positive and negative ground atoms of the most probable world in terms of $\mathcal{A}$, the following assertions remain:

$$\{car(c_1), doorSlam(ds_1), causes(c_1, ds_1), \neg applause(ds_1)\}$$

The ground atom $audio(ds_1)$ is not in the above vector since it is not in $\mathcal{A}$. Since an Abox contains only positive ground atoms, we remove $\neg applause(ds_1)$:

$$\{car(c_1), doorSlam(ds_1), causes(c_1, ds_1)\}$$

This example shows how the inconsistency and the uncertainty in the input data is eliminated by the MAP operation [DLK$^+$08]. Note that an Abox is not a model since an Abox indicates many worlds and only some worlds are models.

The above Tbox does not contain any existential restrictions. In $\mathcal{ALH}_f{}^-$, we have only existentials for the domain restriction of a role which are located on the left side of $\sqsubseteq$ operator. Thus, the determination of the most probable world is not affected.

Relation Between a Weight and a Probability in MLNs:

Based on the Markov logic network the probability of arbitrary weighted atomic concept assertions $w\ A(ind)$ is determined as follows:

$$p = \frac{e^w}{e^w + e^0} \tag{2.19}$$

In other words, $P(A(ind)) = p$ is entailed. The above equation holds for weighted role assertions $w\ R(ind_1, ind_2)$ as well. For $w = -\infty$, the corresponding probability is $p = 0$, and similarly for $w = \infty$, the corresponding probability is $p = 1$. In order to determine the weight for probabilistic assertions given the probability of the associated event, Equation 2.19 has to be resolved to w and the result is:

$$w = \ln(\frac{p}{1-p}) \tag{2.20}$$

Note that the above equations hold if we do not consider the Tbox and any weighted formulas.

Chapter 3

Abduction for Data Interpretation: Related Work

A general introduction on abduction for multimedia data interpretation has been given in [EKM11, Kay11, Esp11]. The insights are not to be repeated here. However, we discuss related work in the context of probabilistic logic-based abduction in general and introduce description logic Abox abduction in particular.

3.1 Perception as Abduction: The General Picture

In [Sha05], abduction is applied in the context of robotics where several sensors are used to discover the physical environment of a robot. Let us assume that Γ indicates the surface-level sensor data. For example, Γ might contain a set of low level observations generated by an edge detector during the image processing. Furthermore, Σ denotes the background knowledge which is composed of two different sets of formulas. The first set describes the effects of the robot's interaction on the world and the second set describes how the changes in the world operates on the robot's sensors. The objective of the abduction is to find one or more explanations Δ such that

$$\Sigma \cup \Delta \models \Gamma \tag{3.1}$$

In addition to the basic abductive framework defined above, Shanahan describes the following important challenges in the context of robotics:

- **Incompleteness and uncertainty**: We have to consider the fact that sensor data is incomplete and uncertain. Incompleteness is due to sensor limitations since each sensor cannot observe its environment completely. There are limitations in the angle, range, etc. Furthermore, there is uncertainty in the data because of the noise effects.

- **Top-down information flow**: The union of the explanation Δ and the background knowledge Σ entails not only the observations Γ but possibly also a set of new assertions called expectations which might not be explicit in Γ. According to these "expectations", surface-level sensor data could be investigated again, possibly by applying different approaches (e.g., different edge detectors), in order to verify whether expectations are actually explanations for observations which were just not made explicit before.

- **Active perception**: Active perception describes the actions to gain more information. Shanahan distinguishes between surface-level and deep-level actions. Examples for surface-level actions are adjusting the threshold of an edge detection routine or rotating the robot's camera. Examples for deep-level actions change the robot's position to observe its environment.

- **Sensor fusion**: The applied sensors have different accuracy rates. Some sensors are more accurate and consequently, their sensor data is more reliable. However, data generated by different sensors might be conflicting. In robotics, for instance, decisions about actions are made according to sensor data. If then sensor data is conflicting, the decisions might not be reliable. In order to make decisions more accurate, sensor fusion can be applied. Sensor fusion is the process of combining different sensor data to a single model of some aspect of the world [CY90].

Let us assume n hypotheses $\Delta_1, \ldots, \Delta_n$ for the sensor data Γ. We assume that only one of the hypotheses can be true at the same time, indicated by:

$$\Delta_1 \oplus \ldots \oplus \Delta_n = true \tag{3.2}$$

Moreover, none of the hypotheses is entailed from any other. Let us consider a hypothesis Δ_k. In the following, we define R which does not contain Δ_k:

$$R = \Delta_1 \oplus \ldots \oplus \Delta_{k-1} \oplus \Delta_{k+1} \oplus \ldots \oplus \Delta_n \tag{3.3}$$

The explanatory value of Δ_k is the conditional probability Δ_k given $\Delta_k \oplus R$:

$$P(\Delta_k \mid \Delta_k \oplus R) \;=\; \frac{P(\Delta_k)}{P(\Delta_k) + P(R)} \tag{3.4}$$

where the prior probability of R is:

$$P(R) = \left(\sum_{i=1}^{n} P(\Delta_i) \right) - P(\Delta_k) \tag{3.5}$$

Thus, it follows that (cf. [Sha05])

$$P(\Delta_k \mid \Delta_k \oplus R) = \frac{P(\Delta_k)}{\sum_{i=1, i \neq k}^{n} P(\Delta_i)} \tag{3.6}$$

Let us assume the hypothesis $\Delta_i = \{\Psi_1, \ldots, \Psi_m\}$. Assuming the assertions of Δ_i are independent, we have:

$$P(\Delta_i) = \prod_{j=1}^{m} P(\Psi_j) \tag{3.7}$$

and we can determine the explanatory value of each explanation Δ_i.

In the following, we summarize the abductive framework approach defined above for explaining surface-level sensor data Γ:

1. Determine explanations Δ_i for sensor data Γ.
2. Calculate the explanatory value of each explanation Δ_i according to Equation 3.6. Then select the best explanations based on the explanatory values according to the idea that the best explanations have the highest explanatory values.
3. Determine the expectations of the best explanations. By different approaches, analyze the images again and check whether the expectations are fulfilled. Accept the explanation in case of fulfillment. Otherwise, reject the explanation.
4. Compute again the explanatory values of the selected explanations according to the results of step 3. Then, sort the explanations accordingly.

In the above procedure, the explanatory value changes in Step 3. Since in Step 3, the image is reanalyzed and the analysis results are determined again. Thus, the existence of some assertions are confirmed and their certainty values are increased. Similarly, the existence of some assertions are rejected. Thus, the new explanation contains assertions with higher certainty values than the previous explanation. Consequently, the new explanation has a higher explanatory value. Note that this approach can only be applied if the independence assumption holds. Thus, the above procedure cannot be applied for every context, but it paves the general way for dealing with interpreting observations using a probabilistic ranking scheme for alternatives computed by abduction.

3.2 Probabilistic Horn Abduction and Independent Choice Logic

One of the main works in the context of logic-based abduction is the independent choice logic (ICL) [Poo97]. The independent choice logic introduces a choice space C and a set of rules R. The choice space C is a set of random variables:

$$C = \{X_1, \ldots, X_n\} \tag{3.8}$$

A set of events $\vec{e}(X)$ is assigned to each random variable X where it holds:

$$\forall X_1, X_2 \in C, \quad X_1 \neq X_2 : \quad \vec{e}(X_1) \cap \vec{e}(X_2) = \emptyset \tag{3.9}$$

The elements of an event $\vec{e}(X)$ are possible values for the random variable X. Let us assume a Boolean random variable X. The possible values for X are *true* and *false* indicated by $X = true$ and $X = false$, respectively. Thus, the set of events of X is:

$$\vec{e}(X) = \{X = true, X = false\} \tag{3.10}$$

Another notation is:

$$\vec{e}(X) = \{\chi, \neg\chi\} \tag{3.11}$$

For better understanding, we give an example for the choice space C. Let us consider the Boolean random variables Car and $DoorSlam$. Thus, the choice space C contains two random variables:

$$C = \{Car, DoorSlam\} \tag{3.12}$$

The sets of events of the above random variables are:

$$\begin{aligned} \vec{e}(Car) &= \{car, \neg car\}, \\ \vec{e}(DoorSlam) &= \{doorSlam, \neg doorSlam\} \end{aligned}$$

where $\vec{e}(Car)$ and $\vec{e}(DoorSlam)$ are disjoint.

The rules R are determined according to the Bayesian network. An event $\alpha \in \vec{e}(X)$ does not appear in the head of a rule.

For every event $\alpha \in \vec{e}(X)$, a probability function P is defined where:

$$\sum_{\alpha \in \vec{e}(X)} P(\alpha) = 1 \tag{3.13}$$

Independent choice logic can also determine explanations for a set of observations. Let us consider the following k abduction rules used for explaining the predicate Q:

$$\begin{aligned} Q &\leftarrow P_1 \\ &\vdots \\ Q &\leftarrow P_k \end{aligned}$$

where $P_1, \ldots, P_k$ indicate predicates and each pair of P_i, P_j cannot be true simultaneously:

$$\forall i, j, \quad i \neq j : \quad P_i \wedge P_j = false \tag{3.14}$$

As it was mentioned before, the rules are determined according to the Bayesian network. In order to explain Q, we consider only the rules where Q is in the head. The probability of an explanation Δ is defined as follows:

$$P(\Delta) = \prod_{\alpha \in \Delta} P(\alpha) \tag{3.15}$$

The above equation is defined under the assumption that the atoms $\alpha \in \Delta$ are independent from each other. The next equation defines the probability of an observation set Γ:

$$P(\Gamma) = \sum_{i=1}^{|\Delta|} P(\Delta) \tag{3.16}$$

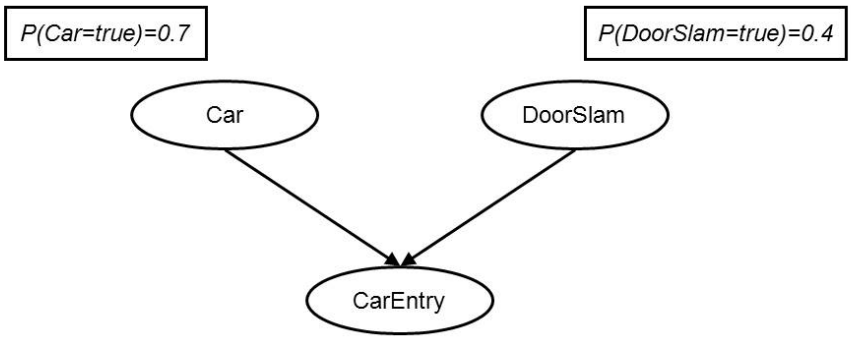
where Δ is a minimal explanation of Γ.

The disadvantage of the independent choice logic is that the computation of an explanation value $P(\Delta)$ is based on the assumption that the atoms α of an explanation Δ are independent from each other (see Equation 3.15). Generally, this assumption does not hold for real applications.

The semantics of the independent choice logic is defined by considering the possible worlds. [Poo08b] introduces a new term called **total choice** to describe the semantics. A total choice for choice space C is defined by selecting exactly one event of each random variable. A possible world corresponds to a total choice. In addition to the events, there are also some other ground atoms in each world which are generated by applying the rules. Note that there is a single model for each possible world since the logic program is acyclic and the events do not appear in the head of the rules. The probability of a possible world is the product of the probabilities of the events which appear in the possible world. Furthermore, the probability of any ground atom is the sum of the probabilities of the possible worlds in which the ground atom is true.

AILOG [Poo08a] is a simple representation and probabilistic reasoning system which is implemented based on the independent choice logic theory [Poo97]. AILog is developed as a simple tell-ask user interface, i.e., the rules are told to the system and queries are asked. Furthermore, AILog acts as a propositional as well as a first-order probabilistic reasoner.

Example 4 In this example, we discuss how a Bayesian network [Pea88] can be represented in independent choice logic [Poo97]. Let us consider the following Bayesian network which consists of three random variables Car, $DoorSlam$ and $CarEntry$ where conditional probability tables are assigned to the random variables. The size of the conditional probability table depends on the number of parents of each node. In the following Bayesian network, the conditional probability tables of Car and $DoorSlam$ have only one single entry since these two nodes have no parents. The conditional probability table of $CarEntry$ has four entries since this node has two parents:



Car	DoorSlam	P(CarEntry=true| Car,DoorSlam)
true	true	0.9
true	false	0.6
false	true	0.3
false	false	0.1

Figure 3.1: An example for Bayesian network

The Boolean random variables Car and $DoorSlam$ are indicated by X_1 and X_2, respectively. The sets of events of these random variables are defined as follows:

$$\begin{aligned} \vec{e}(X_1) &= \{car, \neg car\} \\ \vec{e}(X_2) &= \{doorSlam, \neg doorSlam\} \end{aligned}$$

In the following, the probabilities over the true events ($X = true$) are given. These probabilities are defined according to the conditional probability tables:

$$\begin{aligned} P(car) &= 0.7 \\ P(doorSlam) &= 0.4 \end{aligned}$$

The $CarEntry$ node has two parents. Thus, we introduce four new random variables:

$$\begin{aligned} X_3 &= CarEntryIfCarDoorSlam \\ X_4 &= CarEntryIfCarNoDoorSlam, \\ X_5 &= CarEntryIfNoCarDoorSlam \\ X_6 &= CarEntryIfNoCarNoDoorSlam \end{aligned}$$

The new random variables show how $CarEntry$ depends on Car and $DoorSlam$. The

sets of events of the above random variables are:

$$\begin{aligned}
\vec{e}(X_3) &= \{carEntryIfCarDoorSlam, \neg carEntryIfCarDoorSlam\} \\
\vec{e}(X_4) &= \{carEntryIfCarNoDoorSlam, \neg carEntryIfCarNoDoorSlam\} \\
\vec{e}(X_5) &= \{carEntryIfNoCarDoorSlam, \neg carEntryIfNoCarDoorSlam\} \\
\vec{e}(X_6) &= \{carEntryIfNoCarNoDoorSlam, \neg carEntryIfNoCarNoDoorSlam\}
\end{aligned}$$

According to Figure 3.1, the probabilities are:

$$\begin{aligned}
P(carEntryIfCarDoorSlam) &= 0.9 \\
P(carEntryIfCarNoDoorSlam) &= 0.6 \\
P(carEntryIfNoCarDoorSlam) &= 0.3 \\
P(carEntryIfNoCarNoDoorSlam) &= 0.1
\end{aligned}$$

In order to determine the probabilities $P(X = false)$, we use Equation 3.13. According to Equation 3.13, the sum of the probabilities of the events of a random variable is 1. Thus, for X_1 we have:

$$\begin{aligned}
P(car) + P(\neg car) &= 1 \\
\Rightarrow P(\neg car) &= 0.3
\end{aligned}$$

In the following, the rules R are defined when $CarEntry = true$:

$$\begin{aligned}
R = \{carEntry &\leftarrow car \wedge doorSlam \wedge carEntryIfCarDoorSlam \\
carEntry &\leftarrow car \wedge \neg doorSlam \wedge carEntryIfCarNoDoorSlam \\
carEntry &\leftarrow \neg car \wedge doorSlam \wedge carEntryIfNoCarDoorSlam \\
carEntry &\leftarrow \neg car \wedge \neg doorSlam \wedge carEntryIfNoCarNoDoorSlam\}
\end{aligned}$$

The above rules are defined according to the Bayesian network in Figure 3.1. The choice space C which contains the random variables is defined as follows:

$$C = \{X_1, \ldots, X_6\} \tag{3.17}$$

In the following, the corresponding independent choice logic of the Bayesian network in Figure 3.1 is given (in the notation of AILog [Poo08a]):

$prob\ car : 0.7.$
$prob\ doorSlam : 0.4.$

$carEntry \leftarrow \quad car \wedge doorSlam \wedge carEntryIfCarDoorSlam.$
$carEntry \leftarrow \quad car \wedge \neg doorSlam \wedge carEntryIfCarNoDoorSlam.$
$carEntry \leftarrow \neg car \wedge doorSlam \wedge carEntryIfNoCarDoorSlam.$
$carEntry \leftarrow \neg car \wedge \neg doorSlam \wedge carEntryIfNoCarNoDoorSlam.$

$prob\ carEntryIfCarDoorSlam :\ 0.9.$
$prob\ carEntryIfCarNoDoorSlam :\ 0.6.$
$prob\ carEntryIfNoCarDoorSlam :\ 0.3.$
$prob\ carEntryIfNoCarNoDoorSlam :\ 0.1.$

Table 3.1: An example for independent choice logic

In the ICL specification of a Bayesian network, the probability values of the Bayesian network directly appear [Poo97]. Moreover, an ICL has the same number of probabilities as a Bayesian network.
In this example, there are 2^6 worlds since there are 6 random variables. Each world contains 7 ground atoms where 6 ground atoms are selected from the 6 random variables, respectively. The seventh ground atom, $carEntry$ or $\neg carEntry$, is generated by applying the rules. An example for a possible world is:

$$\begin{aligned} W \quad = \quad & \langle car, doorSlam, carEntryIfCarDoorSlam, \neg carEntryIfCarNoDoorSlam, \\ & \neg carEntryIfNoCarDoorSlam, \neg carEntryIfNoCarNoDoorSlam, carEntry \rangle \end{aligned}$$

In the above world, car, $doorSlam$, and $carEntryIfCarDoorSlam$ are fulfilled. Thus, by applying the first rule of R, we have $CarEntry = true$ which is the last ground atom of W.

In the independent choice logic representation of this example, four rules are defined which show four cases for $CarEntry = true$. Thus, there are four explanations for $CarEntry = true$, namely:

$$\begin{aligned} \Delta_1 \quad &= \quad \{car, doorSlam, carEntryIfCarDoorSlam\} \\ \Delta_2 \quad &= \quad \{car, \neg doorSlam, carEntryIfCarNoDoorSlam\} \\ \Delta_3 \quad &= \quad \{\neg car, doorSlam, carEntryIfNoCarDoorSlam\} \\ \Delta_4 \quad &= \quad \{\neg car, \neg doorSlam, carEntryIfNoCarNoDoorSlam\} \end{aligned}$$

The probability of each explanation Δ_i is calculated based on the Equation 3.15:

$$\begin{aligned} P(\Delta_1) &= 0.7 \times 0.4 \times 0.9 = 0.252 \\ P(\Delta_2) &= 0.7 \times 0.6 \times 0.6 = 0.252 \\ P(\Delta_3) &= 0.3 \times 0.4 \times 0.3 = 0.036 \\ P(\Delta_4) &= 0.3 \times 0.6 \times 0.1 = 0.018 \end{aligned}$$

According to Equation 3.16 the probability $P(CarEntry = true)$ is the sum of the above explanation probabilities:

$$P(CarEntry = true) = P(\Delta_1) + \ldots + P(\Delta_4) = 0.558$$

This example shows that the representation of a Bayesian network [Pea88] in independent choice logic [Poo97] will be complicated if the Bayesian network has nodes with more than two parents, which considerably increases the number of rules.

Probabilistic Horn abduction for diagnostic frameworks is introduced by [Poo91]. Let us consider an explanation D for a set of observations *obs*. Similar to the independent choice logic, the knowledge base is defined according to a Bayesian network.
Let us assume:

$$P(obs \mid D) = 1 \tag{3.18}$$

The above assumption is reasonable if explanation D is a correct explanation for *obs*. In the following, we give an example to show that the conditional probability $P(obs \mid D) = 1$ is reasonable. Let us consider the observation $obs = \{carEntry\}$ and the explanation $D = \{car, doorSlam, carEntryIfCarDoorSlam\}$. Then, the probability $P(obs \mid D)$ is:

$$P(carEntry \mid \{car, doorSlam, carEntryIfCarDoorSlam\}) = 1 \tag{3.19}$$

The above probability is equal to 1 since there is the following rule for the explanation of $carEntry$:

$$carEntry \leftarrow car \wedge doorSlam \wedge carEntryIfCarDoorSlam \tag{3.20}$$

According to the above rule, D is a correct explanation for the observation $carEntry$. Thus, $P(obs \mid D) = 1$ is reasonable.
The scoring value of an explanation D given observations *obs* is determined as follows:

$$P(D \mid obs) = \frac{P(obs \mid D) \times P(D)}{P(obs)} \tag{3.21}$$

$$= \frac{P(D)}{P(obs)} \tag{3.22}$$

In the following, we determine the prior probability of an explanation $D = \{h_1, \ldots, h_n\}$:

$$\begin{aligned} P(D) &= P(h_1 \wedge \ldots \wedge h_{n-1} \wedge h_n) \\ &= P(h_n \mid h_1 \wedge \ldots \wedge h_{n-1}) \times P(h_1 \wedge \ldots \wedge h_{n-1}) \end{aligned}$$

The term $P(h_1 \wedge \ldots \wedge h_{n-1})$ is determined recursively where the final recursive call is $P(true) = 1$. In the following, we compute the term $P(h_n \mid h_1 \wedge \ldots \wedge h_{n-1})$. Since D is a minimal explanation and we also assume that the logically independent instances of D are probabilistically independent, it follows that

$$P(h_n \mid h_1 \wedge \ldots \wedge h_{n-1}) = P(h_n) \tag{3.23}$$

Thus, the prior probability of an explanation D is determined as follows:

$$P(D) = P(h_1 \wedge \ldots \wedge h_n) = \prod_{i=1}^{n} P(h_i) \tag{3.24}$$

Similar to the independent choice logic [Poo97], the probability of an explanation D is the product of the probabilities of the assertions which make up the explanation (Equation 3.15). Thus,

$$P(D \mid obs) = \frac{\prod_{i=1}^{n} P(h_i)}{P(obs)} \tag{3.25}$$

$P(obs)$ is the prior probability of the observations and the value is constant for all explanations. In the context of diagnostic frameworks, we make an assumption that the diagnoses are covering, i.e., considering all existing diagnoses. Moreover, the diagnoses are disjoint (mutually exclusive). Let us assume that $\{e_1, \ldots, e_n\}$ is the set of all explanations of *obs*. Thus, the prior probability of observations is determined as follows:

$$\begin{aligned} P(obs) &= P(e_1 \vee e_2 \vee \ldots \vee e_n) \\ &= P(e_1) + P(e_2) + \ldots + P(e_n) \end{aligned}$$

Similar to the independent choice logic, the prior probability of an observation set *obs* is the sum of the explanation probabilities (Equation 3.16). In the following, we determine the probability of explanation Δ_1 given observation set $obs = \{carEntry\}$:

$$\begin{aligned} P(\Delta_1 \mid obs) &= \frac{P(\Delta_1)}{P(obs)} \\ &= \frac{0.252}{0.558} \\ &= 0.45 \end{aligned}$$

3.3 Computing Explanations as Markov Logic Inference

Probabilistic abduction using Markov Logic networks is discussed in [KM09]. This approach describes how the Abox abduction is performed when the input is probabilistic.

According to this approach, the abduction rules are considered as formulas for Markov logic formalism [DR07]. Since the inference mechanism based on Markov logic is deductive, the notation of the abduction rules has to be changed appropriately. In the following, it is explained how the abduction rules have to be changed to first-order formulas in MLN. The idea of [KM09] is known as Clarke's completion [Lac98].

Let us assume the following n abduction rules in the background knowledge which explain the observation Q:

$$\begin{array}{rcl} Q & \leftarrow & P_1 \\ Q & \leftarrow & P_2 \\ & \vdots & \\ Q & \leftarrow & P_n \end{array}$$

where Q and P_i for $1 \leq i \leq n$ indicate concept or role predicates. The above rules are transformed to the next first-order logic formula which is called the reverse implication:

$$Q \rightarrow P_1 \vee P_2 \vee \ldots \vee P_n$$

The reverse implication formula means that at least one of the possible explanations should be true. Additionally, we have to consider the mutual exclusivity formulas for every pair of explanations as follows:

$$\begin{array}{rcl} Q & \rightarrow & \neg P_1 \vee \neg P_2 \\ & \vdots & \\ Q & \rightarrow & \neg P_1 \vee \neg P_n \\ Q & \rightarrow & \neg P_2 \vee \neg P_3 \\ & \vdots & \\ Q & \rightarrow & \neg P_2 \vee \neg P_n \\ & \vdots & \\ Q & \rightarrow & \neg P_{n-1} \vee \neg P_n \end{array}$$

The mutual exclusivity formulas indicate that multiple explanations cannot be true, simultaneously.

In the Abox abduction, we apply abduction rules which contain existential operators. Thus, new individuals are generated. The problem of this approach [KM09] is that during the Abox abduction new individuals cannot be generated and assigned to the types. In order to deal with the existentials, new individuals are in advance assigned to the types, so that they could be used later during abduction. This approach is not completely correct because the number of required individuals during abduction is not known. Since we do not know exactly how many individuals are required during the Abox abduction, some individuals are randomly assigned to the types. Consequently,

it is possible that the individuals assigned to the types are either not enough or too many. The number of individuals affects the inference process and the precision of the results. If the default individuals are not enough, the result of the Abox abduction is not precise. Similarly, many unrequired individuals slows down the inference process.

By changing the abduction rules according to [KM09] to the notation of formulas in MLN, the Abox abduction is still not defined as desired. The reason is that the explanations are not produced as sets. If there is a small set of abduction rules, the user has a clear overview of them. Thus, the user knows the possible deep level assertions and could ask their probabilities. But if there is a large set of abduction rules, it is not clear any more which assertion should be considered as deep level. Since the user loses the overview, it is not clear which probability should be asked.

The other problem of this approach is that the knowledge base contains only the Horn rules. Thus, the space of hypotheses is not limited.

According to [KM09], we consider non-weighted abduction rules for the Abox abduction. But if there are weighted abduction rules and we want to convert them to the notation of MLN, it is not clear how a weight should be converted to another weight which is assigned to the equivalent formula in MLN.

The other problem is regarding the total number of formulas in MLN. Let us assume n abduction rules with the same predicate in the head. In the following, we determine the total number of equivalent formulas in MLN:

$$\begin{aligned} Sum &= 1 + (n-1) + (n-2) + \ldots + 1 \\ &= 1 + \frac{n(n-1)}{2} \end{aligned}$$

In the above sum, there is only one reverse implication formula and $\frac{n(n-1)}{2}$ mutually exclusive formulas. If we compare, the number of abduction rules n with Sum, we notice that:

$$n \leq Sum \tag{3.26}$$

In the following, we have plotted the Sum function over n:

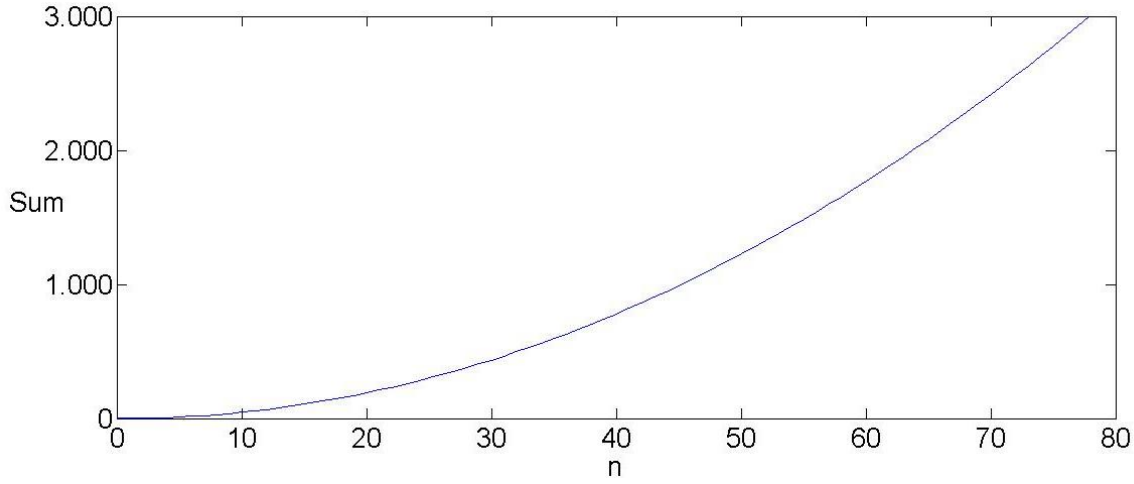


Figure 3.2: The number of abduction rules with the same predicate in the head n and the total number of equivalent formulas in MLN Sum

The above figure shows that by increasing n, the difference among Sum and n becomes greater. This means if we have too many abduction rules with the same predicate in the head, we will have many more equivalent formulas in MLN.

3.4 Preference Models Using Bayesian Compositional Hierarchies

Probabilistic control for interpretation tasks by Bayesian Compositional Hierarchy (BCH) [Neu08] is discussed in [BNHK11, BKN11] (see also earlier work presented in [NM08]). [BKN11] describes a generic framework for ontology-based scene interpretation. The developed framework is applied for real time monitoring of the aircraft service activities. Some examples for aircraft service activities are arrival preparation, unloading, tanking, etc. The input to the interpretation system are analysis results of video streams stemming from various cameras installed at the airport. The output of the interpretation system are high-level activity scene descriptions about events at the airport. Main purposes of real time monitoring of the aircraft service activities are:

- To notify possible delays and to counteract as soon as possible.
- To provide predictions about the completion of an activity.
- To extend monitoring of service activities in order to include unrelated object behaviour (e.g., vehicles which are not allowed to get close to the aircraft.).

The described scene interpretation framework is domain-independent, which means it can be adapted and used for other applications. The input to the interpretation system arrives piecewise since the input is given by the incremental analysis results from the video stream. Thus, early interpretations are with poor context. Since the scene interpretation is determined stepwise, interpretation alternatives need to be explored in parallel. To assign probabilistic ratings to the interpretation alternatives, a probabilistic formalism is used. The BCH formalism [Neu08] defines an aggregate model (compositional hierarchy). An aggregate model is defined as a joint probability distribution in the following form $P(\underline{A}\ \underline{B}_1 \ldots \underline{B}_k\ \underline{C})$ where $\underline{A}$, $\underline{B}_1 \ldots \underline{B}_k$, and $\underline{C}$ indicate the aggregate header, the parts $\underline{B}_i$ of $\underline{A}$, and some conditions $\underline{C}$ on the parts, respectively. An aggregate header represents a part of an aggregate at the next higher level. The work described in [BKN11] employs beam search for exploring the space of possible interpretations given. Interpretations with low ratings are discarded and interpretations with high ratings are kept. For more details see [BKN11].

The approach described in [BKN11] is similar to the approach defined in this work. The difference is in the applied probabilistic formalism and in the defined scoring functions. [BKN11] uses BCH [Neu08] as the probabilistic formalism whereas we use Markov Logic formalism [DR07]. Using the probabilistic formalism of Markov Logic, the construction space for interpretation is defined in terms of abduction in this thesis. Abduction is based on logic programming (LP [Llo87]) rules applied to Aboxes with description logics (DL) [BCM+03] as constraints. The abduction component (called DLLP) is implemented in the RacerPro reasoning system [HM01] and we build on top of it. Basic concepts behind DLLP abduction are described in the next subsection, as part of the abduction-based formalization of interpretation tasks.

3.5 DLLP-Abduction Based Interpretation

Abduction can be applied to concepts and Aboxes. An implementation of concept abduction [CDD+03] is discussed in [RSDD09]. Since interpretations should describe graph rather than tree structures, concept abduction was found not to be expressive enough. [KES11] introduces a formal computational framework for Abox abduction in the DL $\mathcal{ALC}$. The applied reasoning mechanism is based on regular connection tableau [Häh01] and resolution with set-of-support [Lov78]. In the following, we define the term *aggregate instance* which is used in this discussion. An aggregate instance [NM06] is a deep-level object which is built on surface-level objects. We consider the analysis results as surface-level objects. The deep-level objects are the results of the explanation procedure. Thus, the surface-level objects are parts of an aggregate instance. In [KES11], no logic programming rules are used for Abox abduction. We argue that by considering only Tbox axioms as proposed in the elegant approach described in [KES11], we cannot generate appropriate aggregate instances involving graph structures as required for interpretation tasks.

3.5.1 DLLP Abduction

In this section, approaches for computing results for Abox abduction problems are discussed. The main approach is based on description logic [BCM+03] and logic programming [Llo87]. Therefore we use the name DLLP abduction. In the scope of this work, the DLLP Abduction has been implemented with RacerPro [HM01] for interpreting multimedia documents. RacerPro is used as the DL-reasoner in this work. The DLLP Abduction has been well tested through the experimental studies. Section 3.5.1 is taken from [EKM09, GMN+09a, GMN+10a, GMN+10c].

In contrast to the well known deduction process, where the conclusion goes from some causes to an effect, the abduction process goes from an effect to some causes [Pei78]. In general, abduction is formalized as $\Sigma \cup \Delta \models_{\mathcal{R}} \Gamma$ where background knowledge (Σ), rules ($\mathcal{R}$), and observations (Γ) are given, and explanations (Δ) are to be computed. In terms of DLs, Δ and Γ are Aboxes and Σ is a pair of Tbox and Abox. Abox abduction is implemented as a non-standard retrieval inference service in DLs. In contrast to standard retrieval inference services where answers are found by exploiting the ontology, Abox abduction has the task of acquiring what should be added to the knowledge base in order to answer a query. Therefore, the result of Abox abduction is a set of hypothesized Abox assertions. To achieve this, the space of abducibles has to be defined. We do this in terms of rules. We assume that a set of rules $\mathcal{R}$ as defined above (see Section 2.1.2) are specified, and define a non-deterministic function *compute_explanation* as follows [EKM09]:[1]

- $compute\_explanation(\Sigma, \mathcal{R}, \mathcal{A}, P(\underline{z})) = transform(\Phi, \sigma)$ if there is a rule $r \in \mathcal{BR}$ with the following form:

$$P(\underline{X}) \leftarrow Q_1(\underline{Y_1}), \ldots, Q_n(\underline{Y_n})$$

  We apply r to an Abox $\mathcal{A}$ such that a minimal set of atoms Φ and an admissible variable substitution σ with $\sigma(\underline{X}) = \underline{z}$ can be found such that the query

$$Q := \mathbf{\{}() \mid \{Q_1(\sigma'_{fresh\_prefix}(\underline{Y_1})), \ldots, Q_n(\sigma'_{fresh\_prefix}(\underline{Y_n}))\} \setminus \Phi\mathbf{\}}$$

  is answered with *true*. Note that σ might also introduce mappings to individuals not mentioned in $\mathcal{A}$ (new individuals). The number of new individuals is bounded by the number of variables.

  The variable substitution σ' is an extension of σ such that $\sigma'_{prefix}(x) = \sigma(x)$ if $x \in as\_set(\underline{X})$ and, otherwise, $\sigma'_{prefix}(x) = concat(prefix, x)$ where *concat* is a function for concatenating *prefix* and x. The string *fresh_prefix* denotes a fresh name for each application of a rule r (variable renaming).

[1] The funcion *transform* is defined in Section 2.1.2.

- If no such rule r exists in $\mathcal{R}$ it holds that:

$$compute\_explanation(\Sigma, \mathcal{R}, \mathcal{A}, P(\underline{z})) = \emptyset \tag{3.27}$$

The goal of the function *compute_explanation* is to determine what must be added (Φ) such that an entailment $\Sigma \cup \mathcal{A} \cup \Phi \models_{\mathcal{R}} P(\underline{z})$ holds. Hence, for *compute_explanation*, abductive reasoning [FK00, Pau93] is used. The set of assertions Φ represents what needs to be hypothesized in order to answer the query Q with *true*. The definition of *compute_explanation* is non-deterministic due to several possible choices for Φ and r.

In order to locally rank different solutions, we associate a score $n - 2|\Phi|$ with every Φ. The number n denotes the number of query atoms (see the definition of Q above). The score is accessed by referring to a subscript *score* in the expression Φ_{score} (see the next section). This score, a value between 0 and 1 assigns higher values to explanations if more assertions are already entailed and need not be hypothesized (relative to the total number of assertions). We refer to [EKM09] for a detailed evaluation of this score. The score is a "local" score in the sense that only the assertions in the explanations are considered (w.r.t. the background knowledge) but not the whole set of observations is used to score an explanation alternative.

Applying the *compute_explanation* procedure, we say the set of rules is backward-chained, and since there might be multiple rules in $\mathcal{R}$, backward-chaining is non-deterministic. Thus, multiple explanations are potentially generated. The relation *causes* can be explained by the following backward chaining rules:

$$\begin{aligned} \mathcal{BR} = \{\forall x, y\ causes(x, y) &\leftarrow \exists z\ CarEntry(z), hasObject(z, x), \\ &\quad hasEffect(z, y), Car(x), DoorSlam(y) \\ \forall x, y\ causes(x, y) &\leftarrow \exists z\ CarExit(z), hasObject(z, x), \\ &\quad hasEffect(z, y), Car(x), DoorSlam(y)\} \end{aligned}$$

In the following, we devise an abstract computational engine for "explaining" Abox assertions in terms of a given set of rules. Explanation of Abox assertions w.r.t. a set of rules is meant in the sense that using the rules some deep-level explanations are constructed such that the Abox assertions are entailed. The explanation of an Abox is again an Abox. For instance, the output Abox represents results of the content interpretation process. The presentation is slightly extended compared to the one in [CEF+08]. Let the agenda $\mathfrak{A}$ be a set of Aboxes Γ and let Γ be an Abox of observations whose assertions are to be explained. The goal of the explanation process is to use a set of rules $\mathcal{R}$ to derive "explanations" for elements in Γ. The explanation algorithm implemented in the DLLP Abduction Engine (DLLPA Engine) works on a set of Aboxes $\mathfrak{A}$. The complete explanation process is implemented by the DLLPA function [EKM09]:

Algorithm 1: The abduction algorithm

Function DLLPA(Ω, Ξ, Σ, $\mathcal{FR}$, $\mathcal{BR}$, $\mathcal{WR}$, S, $\mathfrak{A}$):
Input: a strategy function Ω, a termination function Ξ, a knowledge base Σ, a set of forward rules $\mathcal{FR}$, a set of backward chaining rules $\mathcal{BR}$, a set of weighted rules $\mathcal{WR}$, a scoring function S, and an agenda $\mathfrak{A}$
Output: a set of interpretation Aboxes $\mathfrak{A}'$
repeat
  $(\mathcal{A}, \alpha, r) := \Omega(\mathfrak{A}, S, \mathcal{WR}, \mathcal{BR})$;
  $\mathfrak{A}' := (\mathfrak{A} \setminus \{\mathcal{A}\}) \cup explanation\_step(\Sigma, r, \mathcal{FR}, \mathcal{A}, \alpha)$;
until $\Xi(\mathfrak{A}')$;
return $\mathfrak{A}'$

In the above algorithm, the strategy function Ω determines the fiat assertion α, Abox $\mathcal{A}$, and the backward chaining rule r where $\alpha \in \mathcal{A}$ and α can be explained by applying r. The strategy function Ω is defined in Algorithm 3. Additionally, DLLPA uses a termination function Ξ in order to check whether to terminate due to resource constraints. The termination condition $\Xi(\mathfrak{A})$ is defined as follows:

$$\neg\exists \text{ a new bunch of observations } \vee \neg\exists \mathcal{A} \in \mathfrak{A} \vee \neg\exists \alpha \in \mathcal{A}$$

where α is an unexplained fiat assertion. Moreover, DLLPA applies a scoring function S to rank explanations. The function *explanation_step* is defined as follows:

$explanation\_step(\Sigma, r, \mathcal{FR}, \mathcal{A}, \alpha)$:

$$\bigcup_{\Delta \in compute\_lo\_explanations(\Sigma, r, \mathcal{FR}, \mathcal{A}, \alpha)} consistent\_completed\_explanations(\Sigma, r, \mathcal{FR}, \mathcal{A}, \Delta)$$

where *compute_lo_explanations* indicates *compute_locally_optimized_explanations*. We need two additional auxiliary functions:

$consistent\_completed\_explanations(\Sigma, r, \mathcal{FR}, \mathcal{A}, \Delta)$:

$$\{\Delta' \mid \Delta' = \Delta \cup \mathcal{A} \cup forward\_chain(\Sigma, \mathcal{FR}, \Delta \cup \mathcal{A}), consistent_{\Sigma}(\Delta')\}$$

The function $consistent_{(\mathcal{T},\mathcal{A})}(\mathcal{A}')$ determines if the Abox $\mathcal{A} \cup \mathcal{A}'$ has a model which is also a model of the Tbox $\mathcal{T}$.

Note the call to the non-deterministic function *compute_explanation*. It may return different values, all of which are collected. In the next chapter, we explain how probabilistic knowledge is used to (i) formalize the effect of the "explanation", and (ii) formalize the scoring function S used in the DLLPA algorithm explained above. In addition, it is shown how the termination condition (represented with the parameter Ξ in the above procedure) can be defined based on the probabilistic conditions.

The Abox abduction algorithm *DLLP* is implemented in the DL reasoner RacerPro [HM01].

3.5.2 Ranking Simple Explanations

A default ranking scheme for explanations (Δs) computed during the abduction process is provided by the implementation is RacerPro [HM01]. A scoring function S evaluates an explanation Δ according to the two criteria proposed by Thagard for selecting explanations [Tha78], namely simplicity and consilience. According to Thagard, the less hypothesized assertions an explanation contains (simplicity) and the more ground assertions (observations) an explanation involves (consilience), the higher its preference score. The following function can be used to compute the preference score for a given explanation[2] [EKM09]:

$$S(\Sigma, \mathcal{R}, \mathcal{A}, \Delta) := S_c(\Sigma, \mathcal{R}, \mathcal{A}, \Delta) - S_h(\Sigma, \mathcal{R}, \mathcal{A}, \Delta) \tag{3.28}$$

The function S_c represents the number of assertions in the analysis Abox $\mathcal{A}$ that follow from $\Sigma \cup \Delta$, and the function S_h indicates the number of assertions in the explanation that do not follow from $\Sigma \cup \mathcal{A}$. Thus, S_c and S_h can be defined as follows:

$$S_c(\Sigma, \mathcal{R}, \mathcal{A}, \Delta) := \sharp\{\alpha \in \mathcal{A} \mid \Sigma \cup \Delta \models_{\mathcal{R}} \alpha\} \tag{3.29}$$

$$S_h(\Sigma, \mathcal{R}, \mathcal{A}, \Delta) := \sharp\{\alpha \in \Delta \mid \Sigma \cup \mathcal{A} \not\models_{\mathcal{R}} \alpha\} \tag{3.30}$$

where $\mathcal{A}$ denotes the analysis Abox. In the context of multimedia interpretation, we prefer an explanation which is more consilient and simpler. Consequently, an explanation with the highest scoring value is preferred. In the following, an example is given which explains how the scoring values of explanations are determined.

Example 5 Let us assume the next Abox:

$$\mathcal{A} = \{Car(c_1), DoorSlam(ds_1), causes(c_1, ds_1)\}$$

Moreover, let us assume the next set of backward chaining rules:

$$\begin{aligned} \mathcal{BR} = \{\forall x, y\ causes(x, y) &\leftarrow \exists z\ CarEntry(z), hasObject(z, x), \\ &\quad hasEffect(z, y), Car(x), DoorSlam(y) \\ \forall x, y\ causes(x, y) &\leftarrow \exists z\ CarExit(z), hasObject(z, x), \\ &\quad hasEffect(z, y), Car(x), DoorSlam(y)\} \end{aligned}$$

By applying the above backward chaining rules on $\mathcal{A}$ the following explanations are generated:

$$\begin{aligned} \Delta_1 &= \{CarEntry(ind_{42}), hasObject(ind_{42}, c_1), hasEffect(ind_{42}, ds_1)\} \\ \Delta_2 &= \{CarExit(ind_{42}), hasObject(ind_{42}, c_1), hasEffect(ind_{42}, ds_1)\} \end{aligned}$$

[2]For the sake of brevity the parameters of S are not shown in the previous functions.

The scoring value of Δ_1 is determined as follows:

$$\begin{aligned} S_{c,1} &= 0 \\ S_{h,1} &= 3 \end{aligned}$$

$$S_1 = S_{c,1} - S_{h,1} = -3$$

Similarly, the scoring value of Δ_2 is determined:

$$\begin{aligned} S_{c,2} &= 0 \\ S_{h,2} &= 3 \end{aligned}$$

$$S_2 = S_{c,2} - S_{h,2} = -3$$

Since the above explanations have the same scoring values, none of them is preferred to the other. This example shows how the explanations are ranked according to the approach defined in [EKM09].

3.6 The Need for Well-Founded Control for Sequential Abduction Steps

The experiments described in [Kay11], [Esp11] indicate that the elementary score described above is required for solving even the simplest one-step abduction problems for interpretation tasks. Only ad hoc control regimes were established for sequential abduction result. In addition, other simplifications were made in [Kay11], [Esp11]. The precondition of the Abox abduction is that the observations are strict. Additionally, in the Abox abduction algorithm no depth and branch control for sequential abduction steps is provided, as the scoring function introduced so far considers only a single abduction step.

As shown in [Kay11], [Esp11], in principle, interpretation of observations can be formalized using DLLP abduction. Although DLLP-abduction based interpretation was shown to be successful from an information retrieval point of view [Kay11] as well as from a content and knowledge management point of view [Esp11], the scores used so far are only a first step, still too many "equal" interpretations can possibly be generated in some contexts.

Therefore, based on the basic score introduced above, in this work, we introduce a new application of DLLP-abduction based interpretation with a second ranking level for which we define a probabilistic scoring function used for interpretation purposes. Although, in principle, the BCH approach [BKN11] described in Section 3.4 could be used as a basis for probabilistic ranking, due to the closed connection to first-order logic we investigated Markov Logic [DR07] for this purpose. Due to the fact that only fixed names can be handled by Markov Logic abduction as proposed by Kate and

Mooney [KM09], we do not follow this approach to generate interpretations but use Markov Logic [DR07] only for ranking.

Unlike abduction as implemented by [Kay11], the probabilistic ranking approach should handle "weighted" observations as well. Using the probabilistic scoring function, approaches for depth and branch control can be implemented as an extension to [Kay11, Esp11]. In addition, a restriction on logic programming rules as used in [Kay11], namely non-recursivity, should be relaxed.

Chapter 4

Probabilistic Control for DLLP-Abduction Based Interpretation

In this chapter, control of multiple applications of the DLLP abduction procedure sketched in the previous chapter is investigated in detail. An interpretation for a set of observations is represented by an Abox, which contains so-called deep-level concept and role assertions. The solution to control interpretation generation is based on a probabilistic ranking principle. The interpretation algorithm includes a scoring function and a termination function. The termination function determines if the interpretation process can be stopped at some point during the interpretation process. The idea for stopping the interpretation process is that if, according to the scoring function, no significant improvements of the interpretation results are achieved there is no benefit for further interpretations, and, consequently, the process is terminated [GMN+10a, GMN+10c] until new input observations arrive. The scoring function defined in this chapter assigns probabilistic scores to interpretations generated by DLLP abduction.

4.1 Abduction-based Interpretation Inside an Agent

Since we adopt the view that intelligent agents are used for solving problems while acquiring information, in the following the interpretation problem is analyzed from the perspective of an agent using an operational semantics. Consider an agent given some observation descriptions in a context where the descriptions are the results of shallow, or surface-level, analysis of multimedia documents.[1] The objective of this agent is to find explanations for input percepts (aka observations). The question is how the interpretation Aboxes for the observations are determined and how long the interpretation procedure should be performed by the agent. The functionality of this

[1] The analysis might also be carried out by the agent.

Media Interpretation Agent is presented in the *MI_Agent* algorithm in Section 4.1. Section 4.1 is mostly taken from [GMN+10a, GMN+10c].

The main idea is summarized as follows. The agent selects observations one after the other for explanation and, basically in a greedy approach, only the best interpretation is used to explain further observations. It might turn out however, that interpretation alternatives currently explored receive lower scores than one of the previous interpretations (that did not explain some observation but may be others very well). Due to space constraints not all interpretations can be kept, however. Thus an ordered agenda of possible interpretations to work on is maintained.

Let us assume that the analysis Abox $\mathcal{A}$ contains n assertions where k assertions are weighted and the remaining assertions are strict:

$$\mathcal{A} = \{w_1\alpha_1, \ldots, w_k\alpha_k, \alpha_{k+1}, \ldots, \alpha_n\}$$

In the above Abox, $\alpha_1, \ldots, \alpha_n$ denote the ground atoms of the MLN and $w_1, \ldots, w_k$ indicate weights.

In the first-order knowledge base, we have weighted rules (or formulas) $\mathcal{WR}$ enforcing that models form a compositional hierarchy with interconnections among the parts [BKN11]:

$$w\ C_3(z) \wedge r_2(z,x) \wedge r_3(z,y) \Rightarrow r_1(x,y) \wedge C_1(x) \wedge C_2(y) \tag{4.1}$$

where w indicates a real number weight from the interval $(-\infty, +\infty)$. In order to explain the terms on the right side of the above weighted rule in the spirit of DLLP abduction, i.e., for technical reasons, we transform the above rule into the following backward chaining rules:

$$r_1(x,y) \leftarrow C_1(x), C_2(y), C_3(z), r_2(z,x), r_3(z,y) \tag{4.2}$$
$$C_1(x) \leftarrow r_1(x,y), C_2(y), C_3(z), r_2(z,x), r_3(z,y) \tag{4.3}$$
$$C_2(y) \leftarrow C_1(x), r_1(x,y), C_3(z), r_2(z,x), r_3(z,y) \tag{4.4}$$

Similarly, we consider the next weighted rule:

$$w\ C_2(z) \wedge r(z,x) \Rightarrow C_1(x) \tag{4.5}$$

The following backward chaining rule is required to explain the term on the right side of the above weighted rule:

$$C_1(x) \leftarrow C_2(z), r(z,x) \tag{4.6}$$

The derivation of backward chaining rules for DLLP abduction is straightforward and can be automatized. Thus, the interpretation knowledge of the agent can be specified as a set of weighted rules of the form explained above. Next, we introduce the term *fiat assertion* which is used frequently in subsequent algorithms. Fiat assertions are the assertions which require explanations. The predicate of a fiat assertion is matched

with the head of a backward chaining rule. Note that each fiat assertion represents an observation.

In the following, we define $Fiats(\mathcal{A})$ which returns a set containing fiat assertions of $\mathcal{A}$ w.r.t. a set of backward chaining rules $\mathcal{BR}$:

$$Fiats(\mathcal{A}) = \{\alpha \in \mathcal{A} \mid \exists r \in \mathcal{BR} : predicateSym(head(r)) = predicateSym(\alpha)\}$$

where $predicateSym(\alpha)$ returns the predicate symbol (role or concept name) mentioned in the assertion α. Moreover, $head(r)$ returns the head of a backward chaining rule r.

The interpretation procedure *interpret* is defined in Algorithm 2 shown below. Before discussing the algorithm in detail, the auxiliary operation $P(\mathcal{A}, \mathcal{WR})$ in the *interpret* algorithm is explained. The $P(\mathcal{A}, \mathcal{WR})$ function determines the scoring value of the interpretation Abox $\mathcal{A}$ with respect to a set of weighted rules $\mathcal{WR}$. In an interpretation Abox, the fiat assertions are divided into two disjoint sets, namely explained and unexplained fiat assertions. The defined scoring value in this work is the average of the fiat assertion probabilities, including explained and unexplained ones. We assume that the probability of an unexplained fiat assertion to be true is 0.5 (the value could be supplied as a parameter or could be made context-specific).

Let us assume that the interpretation Abox $\mathcal{A}$ has n explained and m unexplained fiat assertions. The scoring value of $\mathcal{A}$ is determined based on the Markov logic formalism [DR07] as follows:

$$P(\mathcal{A}, \mathcal{WR}) = \frac{1}{n+m}\left(m \times 0.5 + \sum_{i=1}^{n} P_{MLN(\mathcal{A},\mathcal{WR})}(Q_i(\mathcal{A}) \mid \vec{e}(\mathcal{A}))\right) \tag{4.7}$$

where $Q_i(\mathcal{A})$ and $\vec{e}(\mathcal{A})$ denote the query and the evidence vector, respectively. The query $Q_i(\mathcal{A})$ is defined as follows:

$$Q_i(\mathcal{A}) = \langle \alpha_i = true \rangle$$

where α_i is an explained fiat assertion of $\mathcal{A}$. The query $Q(\mathcal{A})$ and evidence vector $\vec{e}(\mathcal{A})$ are functions of the interpretation Abox $\mathcal{A}$ since each $\mathcal{A}$ contains different fiat assertions. Note that during the interpretation procedure the query might change. This is due to increasing fiat assertions by applying forward and backward chaining rules. Additionally, by explaining fiat assertions, the number of explained fiat assertions in the interpretation Abox increases and the number of unexplained fiat assertions decreases.

The evidence vector $\vec{e}(\mathcal{A})$ contains in addition to the observations, assertions generated by forward, and backward chaining rules except the fiat assertions:

$$\vec{e}(\mathcal{A}) = \{\alpha = true \mid \alpha \in \mathcal{A} \setminus Fiats(A)\}$$

The observations and the hypothesized assertions (except the fiat assertions) are considered as evidences for the determination of the probabilistic scoring values. In order to answer the query $P_{MLN(\mathcal{A},\mathcal{WR})}(Q(\mathcal{A}) \mid \vec{e}(\mathcal{A}))$ the function $MLN(\mathcal{A}, \mathcal{WR})$ is called.

This function returns a Markov logic network $(\mathcal{F}_{MLN}, \mathcal{W}_{MLN})$ where $r \in \mathcal{F}$ and $w \in \mathcal{W}$ iff $w\, r \in \mathcal{WR}$.

Note that for constructing Markov logic network [DR07] only the weighted rules $\mathcal{WR}$ are considered as formulas. Furthermore, the constants of the Markov logic network are the individuals which exist in $\mathcal{A}$. There are also some predicates in the weighted rules with no constants. Thus, constants are defined and assigned to the types of those predicates. As a result, the constant set contains the individuals which exist in $\mathcal{A}$ and the defined constants. Note that the strict assertions of $\mathcal{A}$ are not considered as formulas for constructing the Markov logic network.

In the following, the interpretation algorithm *interpret* is presented:

Algorithm 2: The interpretation algorithm

Function interpret($\mathfrak{A}$, Γ, $(\mathcal{T}, \mathcal{A}_0)$, $\mathcal{FR}$, $\mathcal{BR}$, $\mathcal{WR}$)
Input: an agenda $\mathfrak{A}$, an Abox of observations Γ, a Tbox $\mathcal{T}$, an Abox $\mathcal{A}_0$, a set of forward chaining rules $\mathcal{FR}$, a set of backward chaining rules $\mathcal{BR}$, and a set of weighted rules $\mathcal{WR}$
Output: an agenda $\mathfrak{A}'$, and a new interpretation Abox $NewI$
$\Sigma := (\mathcal{T}, \mathcal{A}_0)$;
$S := \lambda(\mathcal{A}).P(\mathcal{A} \cup \mathcal{A}_0, \mathcal{WR})$;
$\mathfrak{A}' := DLLPA(\Omega, \Xi, \Sigma, \mathcal{FR}, \mathcal{BR}, \mathcal{WR}, S, \mathfrak{A})$;
$NewI := argmax_{\mathcal{A} \in \mathfrak{A}'}(P(\mathcal{A}, \mathcal{WR}))$;
return $(\mathfrak{A}', NewI)$;

In the above algorithm, the $DLLPA$ function is called which returns an agenda $\mathfrak{A}'$. The function $argmax_{\mathcal{A} \in \mathfrak{A}} S(\mathcal{A})$ selects those elements $\mathcal{A} \in \mathfrak{A}$ for which the scoring function S applied to $\mathcal{A}$ returns a maximal value:

$$max\_score(\mathfrak{A}) = argmax_{\mathcal{A} \in \mathfrak{A}} S(\mathcal{A}) \tag{4.8}$$

Thus, the interpretation Abox $NewI$ with the maximum score among the Aboxes $\mathcal{A}$ of $\mathfrak{A}'$ is selected.

The reason for considering all interpretation Aboxes on the agenda and not only the one with the highest scoring value is that the fiat assertions in the interpretation Aboxes $\mathcal{A}$ are not the same. Since the query is a function of the interpretation Abox $Q(\mathcal{A})$, it might be the case that after the explanation procedure we can find an interpretation Abox with a higher scoring value than the Abox which has currently the maximum score.

In the following, we define the strategy function Ω, which is one of the parameters of DLLPA:

Algorithm 3: The strategy algorithm

Function $\Omega(\mathfrak{A}, S, \mathcal{WR}, \mathcal{BR})$
Input: a set of interpretation Aboxes $\mathfrak{A}$, a scoring function S, a set of weighted rules $\mathcal{WR}$, and a set of backward chaining rules $\mathcal{BR}$
Output: an Abox $\mathcal{A}$, a fiat assertion α, and a backward chaining rule r
$\mathfrak{A} := sort(\mathfrak{A}, S)$
$\mathcal{A} := max\_score(\mathfrak{A})$;
$B = \emptyset$;
foreach $\alpha \in UnexplainedFiats(\mathcal{A})$ **do**
  foreach $wr \in \mathcal{WR}$ **do**
    `// Determine the tuples containing a fiat assertion and the supporting weighted rule`
    $B = B \cup \{(\alpha_i, w_i r_i)\}$;
$max\_ws = \{(\alpha_n, w_n r_n) \in B \mid \neg\exists(\alpha_l, w_l r_l) \in B, \alpha_n \neq \alpha_l, r_n \neq r_l, w_l > w_n\}$;
$(\alpha_m, w_m r_m) = random\_select(max\_ws)$;
`Find` $r \in \mathcal{BR}$ so that r is the corresponding backward chaining rule of $w_m r_m$;
return $(\mathcal{A}, \alpha_m, r)$;

In the following, we define the supporting weighted rule $\mathcal{SWR}$ for the fiat assertion α:

$$\mathcal{SWR}(\alpha) = \{wr \in \mathcal{WR} \mid predicateSym(\alpha) = predicateSym(head(wr))\}$$

Let us assume an Abox containing multiple fiat assertions. In this algorithm, we discuss which fiat assertion should be explained first, so that the scoring values assigned to the interpretation Aboxes are plausible. We consider the top k interpretation Aboxes according to the scoring values. The scoring value assigned to an interpretation Abox is the average of the fiat assertion probabilities given the evidences. Furthermore, the knowledge base contains weighted rules which are used for computing the scoring values of the interpretation Aboxes. The weights of the weighted rules are not necessarily equal. This means the fiat assertions are supported by the weighted rules with high and/or small weights. Thus, the order to select the next fiat assertion affects the scoring value assigned to the interpretation Abox. In case of selecting a fiat assertion supported by a weighted rule with a small weight, a small scoring value is assigned to the interpretation Abox which could have a high scoring value. Thus, the generated interpretation Abox is most probably not among the top k interpretation Aboxes and is discarded. In order to avoid such cases, we have to select at each step the fiat assertion which has a supported weighted rule with the highest weight.

The above strategy function Ω contains as argument the agenda $\mathfrak{A}$ which is a set of Aboxes $\mathcal{A}$, a scoring function S, a set of weighted rules $\mathcal{WR}$, and a set of backward chaining rules $\mathcal{BR}$. We determine the set B containing the tuples with a fiat assertion $\alpha \in \mathcal{A}$ and a supporting weighted rule for α. Then, we find the tuples from B containing the maximum weight. If there is more than one tuple with the maximum weight, one

of the tuples is randomly selected. The next fiat assertion to be explained is the first element α_m in the tuple with the maximum weight. According to the weighted rule in the selected tuple, the corresponding backward chaining rule from $\mathcal{BR}$ is found which is used for explaining the fiat assertion α_m. The above algorithm returns the Abox $\mathcal{A}$ with the maximum score, the next fiat assertion $\alpha_m \in \mathcal{A}$ to be explained and a backward chaining rule r.

As introduced at the beginning of this chapter, the media interpretation agent extracts observations from the analysis queue. The analysis results do not arrive at equal time distances.

Thus, we check the analysis queue in a loop if it is not empty. In the following, we present the media interpretation agent algorithm MI_Agent. This algorithm calls the *interpret* function.

Algorithm 4: The multimedia interpretation agent algorithm

Function MI_Agent($\mathcal{Q}_\Gamma$, *die*(), $(\mathcal{T}, \mathcal{A}_0)$, $\mathfrak{A}$, $\mathcal{FR}$, $\mathcal{BR}$,$\mathcal{WR}$)
Input: a queue of observations $\mathcal{Q}_\Gamma$, a termination function *die*(), a background knowledge base $(\mathcal{T}, \mathcal{A}_0)$, an agenda $\mathfrak{A}$, a set of forward chaining rules $\mathcal{FR}$, a set of backward chaining rules $\mathcal{BR}$, and a set of weighted rules $\mathcal{WR}$
Output: –
$currentI = \emptyset$;
$\mathfrak{A}' = \{\emptyset\}$;
$\Sigma := (\mathcal{T}, \mathcal{A}_0)$;
$stopProcessing = false$;
repeat
  $\Gamma := extractObservations(\mathcal{Q}_\Gamma)$;
  $W := MAP(\Gamma, \mathcal{T})$;
  $\Gamma' := select(W, \Gamma)$;
  label :
  $\mathcal{A} = argmax_{\mathcal{A} \in \mathfrak{A}}(P(\mathcal{A}, \mathcal{WR}))$
  $\mathfrak{A} = \mathfrak{A} \setminus \mathcal{A}$
  Consider the remaining unexplained fiats of $\mathcal{A}$ as explained
  $\mathcal{A} = \mathcal{A} \cup \mathcal{A}_0 \cup \Gamma'$
  if $\mathcal{A}$ *is consistent* **then**
    $\mathcal{A} = \mathcal{A} \cup forward\_chain(\Sigma, \mathcal{FR}, \mathcal{A})$
    if $\mathcal{A}$ *is consistent* **then**
      $\mathfrak{A}' = \mathfrak{A}' \cup \mathcal{A}$
      $(\mathfrak{A}, newI) := interpret(\mathfrak{A}', \Gamma', \Sigma, \mathcal{FR}, \mathcal{BR}, \mathcal{WR})$
      $currentI := newI$
    else
      goto *label*
  else
    goto *label*
until *stopProcessing=true or die*();

The *MI_Agent* uses a set of standard functional programming patterns such as *filter*, and *zip*. Furthermore, a function *select* is defined which uses the latter two functions. In the following, the function *filter* is defined:

$$filter(f, X) := \bigcup_{x \in X} \begin{cases} \emptyset & \text{if } f(x) = false \\ \{f(x)\} & \text{else} \end{cases}$$

The function *filter* takes as parameters a function f and a set X and returns a set consisting of the values of f applied to every element $x \in X$ if $f(x) \neq false$. The

function *zip* is defined as follows:

$$zip(X, Y) := \bigcup_{x \in X, y \in Y} \{(x, y)\}$$

The function *zip* produces as output a set of tuples by taking as parameters two sets X and Y and pairing their successive elements.

In order to select elements y from an ordered set Y using a bit vector which is also represented by an ordered set X, the function *select* is defined as follows:

$$select := \lambda(X, Y).filter(\lambda(x, y).\textbf{if } x \textbf{ then } y \textbf{ else } false, zip(X, Y))$$

The *select* function determines the positive ground atoms of the most probable world which also exist in the Abox. Since $\mathcal{A}$ might contain probabilistic assertions, the most probable world has to be determined by MAP operation [DLK+08]. By applying MAP operation, we convert the probabilistic logic into the classical logic where the assertions are strict. The positive ground atoms which also exist in $\mathcal{A}$ are determined by MAP operation. These positive ground atoms stand as evidences in the database file. Each assertion of the database file indicates the truth value of the corresponding ground atom in the Markov logic network.

In the MI_Agent function, the current interpretation *currentI* is initialized to empty set and the agenda $\mathfrak{A}'$ to a set containing empty set. Since the agent performs an incremental process, it is defined by a repeat-loop. In case the agent receives a percept result Γ, it is sent to the queue $\mathcal{Q}_\Gamma$. In order to take the observations Γ from the queue $\mathcal{Q}_\Gamma$, the MI_Agent calls the *extractObservations* function.

The function $select(W, \Gamma)$ then selects the positive assertions from the input Abox Γ using the bit vector W. The selected positive assertions are the assertions which require explanations. The *select* operation returns as output an Abox Γ'. The determination of the most probable world by the MAP function and the selection of the positive assertions is carried out on $\Gamma' \cup \mathcal{A} \cup \mathcal{A}_0$. In the next step, a set of forward chaining rules $\mathcal{FR}$ is applied to $\Gamma' \cup \mathcal{A} \cup \mathcal{A}_0$. The generated assertions in this process are added to $\Gamma' \cup \mathcal{A} \cup \mathcal{A}_0$. In the next step, only the consistent Aboxes are selected and the inconsistent Aboxes are removed. Then the *interpret* function is called which determines the interpretation Aboxes. Afterwards, the Abox *currentI* is assigned to *newI*. The termination condition of the MI_Agent function is that the *stopProcessing* is activated externally by the user or the *die*() function is true. In case of $\mathcal{Q}_\Gamma = \emptyset$, the MI_Agent waits at the function call *extractObservations*.

Example 6 Let us consider the Tbox axiom:

$$DoorSlam \sqsubseteq \neg EngineSound$$

and the following Abox:

$$\begin{aligned} \mathcal{A} \;=\; & \{1.3\ Car(c_1), 1.2\ DoorSlam(ds_1), causes(c_1, ds_1), \\ & 0.6\ EngineSound(ds_1)\} \end{aligned}$$

In order to determine the most probable world, the MAP operation [DLK+08] is applied to the above Abox. The result is the following vector:

$$X = \langle 1, 1, 1, 0 \rangle$$

where each element of X corresponds to the elements of the following vector:

$$Y = \langle Car(c_1), DoorSlam(ds_1), causes(c_1, ds_1), EngineSound(ds_1) \rangle$$

The $zip(X, Y)$ function generates the next set of pairs:

$$\begin{aligned} zip(X, Y) &= \{(1, Car(c_1)), (1, DoorSlam(ds_1)), (1, causes(c_1, ds_1)), \\ &\quad (0, EngineSound(ds_1))\} \end{aligned}$$

Applying the function $f(x, y) =$ **if** x **then** y **else** *false* on the above set of pairs returns the following components, respectively:

$$Car(c_1), DoorSlam(ds_1), causes(c_1, ds_1), false$$

The result of the $select(X, Y)$ function is:

$$\begin{aligned} select(X, Y) &= \{Car(c_1)\} \cup \{DoorSlam(ds_1)\} \cup \{causes(c_1, ds_1)\} \cup \emptyset \\ &= \{Car(c_1), DoorSlam(ds_1), causes(c_1, ds_1)\} \end{aligned}$$

The above set contains the positive ground atoms of $\mathcal{A}$ existing in the most probable world. In this work, we are only interested in the positive ground atoms of the most probable world. The reason is that we consider the open-world assumption [BCM+03]. According to the open-world assumption, if the knowledge base Σ does not entail assertion $C(i)$, it does not follow $\neg C(i)$:

$$\Sigma \not\models_{OWA} \neg C(i) \text{ if } \Sigma \not\models_{OWA} C(i) \tag{4.9}$$

This example shows, how the positive ground atoms of an Abox existing in the most probable world are determined.

After having presented the above algorithms, the unanswered questions mentioned above can be discussed. A reason for performing the interpretation process and explaining the fiat assertions is that the probability of $P(\mathcal{A}, \mathcal{WR})$ will increase through the interpretation process [GMN+10a, GMN+10c]. In other words, by explaining the observations the agent's belief of the observations being true will increase.

4.2 Controlling the Interpretation Process

In the previous section, we described the interpretation process. In this section, we define controlling the interpretation process which is the main objective of this work. We control the interpretation process by applying Markov logic formalism [DR07]. We apply Markov logic formalism for ranking interpretation alternatives but not for generating interpretations. We cannot apply Markov logic abduction approach described in [KM09] for generating interpretations since this approach requires fixed names for individuals. The reason for controlling the interpretation process is that the agent's resources for computing explanations are limited. In the following, we define three different approaches for controlling the interpretation process, namely controlling branching, controlling abduction depth, and controlling reactivity.

4.2.1 Controlling Branching

Controlling branching is one of the procedures for controlling the interpretation process. In the following, we define branching problem. Branches are generated if there are more than one explanation Δ for a fiat assertion α. Multiple explanations lead to multiple interpretation alternatives. Thus, the computation of the interpretation procedure is a resource-consuming task. In order to reduce the required resources, we apply controlling branching procedure.

According to controlling branching procedure, we use an agenda to maintain possible interpretation alternatives. The reason is that maintaining all interpretation alternatives is a resource-consuming task. On the agenda, the interpretation alternatives are sorted by scoring values. Let us assume an agenda with size k. An agenda with size k maintains the top k interpretation alternatives. Thus, we keep interpretation alternatives with high ratings and discard interpretations with low ratings. Consequently, we reduce the memory space required for the interpretation procedure, enormously.

In the following, we define controlling branching procedure applied in this work. According to this approach, we apply Beam search to explore the interpretation space. The reason for applying Beam search is that Beam search is non-exhaustive. Thus, we reduce the size of the interpretation space. Additionally, we apply Formula 4.7 as the probabilistic scoring function. Thus, we rate interpretation alternatives, probabilistically. The probabilistic scoring function 4.7 is defined according to Markov logic formalism [DR07]. We consider the set of weighted rules $\mathcal{WR}$ for ranking interpretation alternatives. According to the scoring function 4.7, the rank of an interpretation alternative is the average of the fiat assertion probabilities given the evidences.

For the interpretation process, the agent selects fiat assertions of an interpretation alternative one after the other for explanation. In this work, the weights of the weighted rules are not necessarily equal. Thus, the selection order of fiat assertions affects the ranks of the interpretation alternatives. In this work, we defined strategy algorithm 3 which determines the selection order of fiat assertions. Strategy algorithm 3 selects a fiat

assertion which has a supported weighted rule with the highest weight. Consequently, the ranks of the interpretation alternatives are plausible.

According to Beam search, at each step only the interpretation alternative with the highest score is chosen to explain further fiat assertions. Thus, the fiat assertions of the other interpretations with smaller scores are not explained any more. Consequently, only the interpretation with the highest score is replaced by new interpretations. Thus, we have branchings only for this interpretation alternative. It might happen that the generated interpretations receive smaller scores than the previous ones which were not selected for further explanation of fiat assertions. In this case, the interpretation alternatives on the agenda are sorted by scoring values.

Let us assume an agenda with size k. In the following, we summarize the mentioned steps:

1. To select the interpretation alternative with the highest score $I_{maxScore}$ from the agenda.
2. To select a fiat assertion α of $I_{maxScore}$ according to strategy algorithm 3.
3. To explain fiat assertion α selected in Step 2.
4. To rank interpretation alternatives according to Formula 4.7.
5. To sort interpretation alternatives by their ranks.
6. To maintain the top k interpretation alternatives on the agenda.

The advantage of controlling branching is that we reduce time and memory space required for computation of the interpretation procedure. Thus, we reduce the complexity of the branching problem, enormously. Furthermore, controlling branching reduces the size of the interpretation space.

4.2.2 Controlling Abduction Depth

In addition to controlling branching described above, we define controlling abduction depth which is another procedure for controlling the interpretation process. In 4.2.1, we defined 6 steps which are applied for controlling branching. According to controlling abduction depth, we determine how many times these steps are performed. Thus, the steps in 4.2.1 are inside a loop which iterates m times, maximally. m indicates the maximum abduction depth. Consequently, controlling abduction depth determines how many fiat assertions are maximally explained. The explained fiat assertions are not necessarily from an Abox. The reason is that at Step 1, we explain a fiat assertion of the interpretation alternative with the highest rank $I_{maxScore}$. Note that $I_{maxScore}$ changes over time and does not indicate the same Abox. Thus, an explained fiat assertion is from an Abox which has the highest rank at a point of time.

Let us assume k fiat assertions. Furthermore, we assume that the steps in 4.2.1 are inside a loop which iterates m times, maximally. For $k < m$, the loop iterates k times. Thus, k fiat assertions are explained. For $k \geq m$, the loop iterates m times and m fiat assertions are explained.

The question is how to select variable m, appropriately. By selecting a small number for m, we might lose some interpretation results. The reason is that all fiat assertions might not be explained, completely. By selecting a large value for m, the interpretation space becomes very large. Thus, we cannot control the abduction depth anymore. A future work topic is the selection of variable m, automatically.

Similar to controlling branching, the advantage of controlling abduction depth is that we reduce time and memory space required for the computation of the interpretation procedure. Furthermore, we reduce the size of the interpretation space.

In the following, we give two examples. Example 7 shows how the scoring value of an interpretation alternative increases monotonically by explaining fiat assertions. Example 8 shows how controlling branching and controlling abduction depth defined in this work are applied.

Example 7 Let us consider the following Abox $\mathcal{A}$:

$$\begin{aligned}\mathcal{A} \;=\; \{&Car(c_1), DoorSlam(ds_1), causes(c_1, ds_1),\\ &EnvConference(ec_1), Env(e_1), hasTopic(ec_1, e_1)\}\end{aligned}$$

Furthermore, let us assume that the following weighted rules are given. We consider them for the determination of the scoring values of the interpretation Aboxes:

$$\begin{aligned}5\ \forall x, y, z\ &CarEntry(z) \wedge hasObject(z, x) \wedge hasEffect(z, y) \rightarrow\\ &Car(x) \wedge DoorSlam(y) \wedge causes(x, y)\\ 5\ \forall x, y, z\ &EnvProt(z) \wedge hasEvent(z, x) \wedge hasTheme(z, y) \rightarrow\\ &EnvConference(x) \wedge Env(y) \wedge hasTopic(x, y)\end{aligned}$$

The next backward chaining rules are constructed according to the above weighted rules:

$$\begin{aligned}\forall x, y\ causes(x, y) \;\leftarrow\; &\exists z\ CarEntry(z), hasObject(z, x),\\ &hasEffect(z, y), Car(x), DoorSlam(y) \qquad (4.10)\\ \forall x, y\ hasTopic(x, y) \;\leftarrow\; &\exists z\ EnvProt(z), hasEvent(z, x),\\ &hasTheme(z, y), EnvConference(x), Env(y) \qquad (4.11)\end{aligned}$$

For simplification reasons, we assume there are no forward chaining rules to be applied. In $\mathcal{A}$, the fiat assertions are:

$$Fiats(\mathcal{A}) = \{causes(c_1, ds_1), hasTopic(ec_1, e_1)\}$$

The evidence vector considered for the inference determination contains the observations (except the fiat assertions):

$$\begin{aligned}\vec{e}(\mathcal{A}) \;=\; \langle & Car(c_1) = true,\\ & DoorSlam(ds_1) = true,\\ & EnvConference(ec_1) = true,\\ & Env(e_1) = true\rangle\end{aligned}$$

The probability of an unexplained fiat assertion is considered as 0.5. Since $\mathcal{A}$ has two unexplained fiat assertions, its scoring value is:

$$P(\mathcal{A}, \mathcal{WR}) = \frac{1}{2}\,(0.5 \times 2) = 0.5$$

We begin by explaining the fiat assertion $causes(c_1, ds_1)$. This fiat assertion is explained by the backward chaining rule 4.10. Thus, the interpretation Abox $\mathcal{A}'$ is:

$$\mathcal{A}' = \mathcal{A} \cup \{CarEntry(ind_{42}), hasObject(ind_{42}, c_1), hasEffect(ind_{42}, ds_1)\}$$

The evidence vector $\vec{e}(\mathcal{A}')$ contains the observations, and the hypothesized assertions except the fiat assertions:

$$\begin{aligned}\vec{e}(\mathcal{A}') = \vec{e}(\mathcal{A}) \cup \langle & CarEntry(ind_{42}) = true,\\ & hasObject(ind_{42}, c_1) = true,\\ & hasEffect(ind_{42}, ds_1) = true\rangle\end{aligned}$$

In the following, we determine the probability of the first fiat assertion given the evidences:

$$P(causes(c_1, ds_1) \mid \vec{e}(\mathcal{A}')) = 0.824$$

Thus, the scoring value of the interpretation Abox $\mathcal{A}'$ is:

$$P(\mathcal{A}', \mathcal{WR}) = \frac{1}{2}(0.824 + 0.5) = 0.662$$

By applying 4.11, we explain the second fiat assertion. The new interpretation Abox is:

$$\mathcal{A}'' = \mathcal{A}' \cup \{EnvProt(ind_{44}), hasEvent(ind_{44}, ec_1), hasTheme(ind_{44}, e_1)\}$$

The evidence vector $\vec{e}(\mathcal{A}'')$ is extended by the hypothesized assertions:

$$\begin{aligned}\vec{e}(\mathcal{A}'') = \vec{e}(\mathcal{A}') \cup \langle & EnvProt(ind_{44}) = true,\\ & hasEvent(ind_{44}, ec_1) = true,\\ & hasTheme(ind_{44}, e_1) = true\rangle\end{aligned}$$

In order to determine the scoring value, we determine the probability of the fiat assertions:

$$\begin{aligned} P(causes(c_1, ds_1) \mid e(\vec{\mathcal{A}}'')) &= 0.824 \\ P(hasTopic(ec_1, e_1) \mid e(\vec{\mathcal{A}}'')) &= 0.824 \end{aligned}$$

The new scoring value is:

$$P(\mathcal{A}'', \mathcal{WR}) = \frac{1}{2}(0.824 + 0.824) = 0.824$$

The next table shows the summary of the results:

$P(\mathcal{A}, \mathcal{WR})$	=	0.5
$P(\mathcal{A}', \mathcal{WR})$	=	0.662
$P(\mathcal{A}'', \mathcal{WR})$	=	0.824

Table 4.1: The summary of the results

This example shows that by successively explaining fiat assertions of an Abox, the scores of the generated interpretation Aboxes increase monotonically (the weights used in the weighted rules are assumed to be positive).

Example 8 In this example, we discuss how the branching and depth controlling described in this work are applied. Let us assume that an Abox $\mathcal{A}$ is given, with four unexplained fiat assertions. For simplification reasons, the maximum number of fiat assertions to be explained m is 2. In this example, we discuss the first three steps S_1, S_2, S_3 according to the algorithm. We assume at step S_1, two backward chaining rules are applicable to explain one of the fiat assertions. Thus, $\mathcal{A}$ is replaced by $\mathcal{A}_1$ and $\mathcal{A}_2$ where the scores are 0.8 and 0.7, respectively. Thus, $\mathcal{A}_1$ and $\mathcal{A}_2$ have one explained and three unexplained fiat assertions.

Since we apply Beam search for finding the final interpretation Abox, we continue the next step with the Abox with the highest score, namely $\mathcal{A}_1$. At S_2, we explain one of the unexplained fiat assertions of $\mathcal{A}_1$. Let us assume that two backward chaining rules are again applicable to explain one of the unexplained fiat assertions. Thus, $\mathcal{A}_1$ is replaced by $\mathcal{A}_3$ and $\mathcal{A}_4$ where the scores are 0.9 and 0.6. $\mathcal{A}_3$ and $\mathcal{A}_4$ have two explained and two unexplained fiat assertions.

Let us assume the agenda size is three. Since the interpretation alternatives on the agenda have to be sorted by scoring values, at S_3 we sort the interpretations on the agenda. This example is depicted in Figure 4.1:

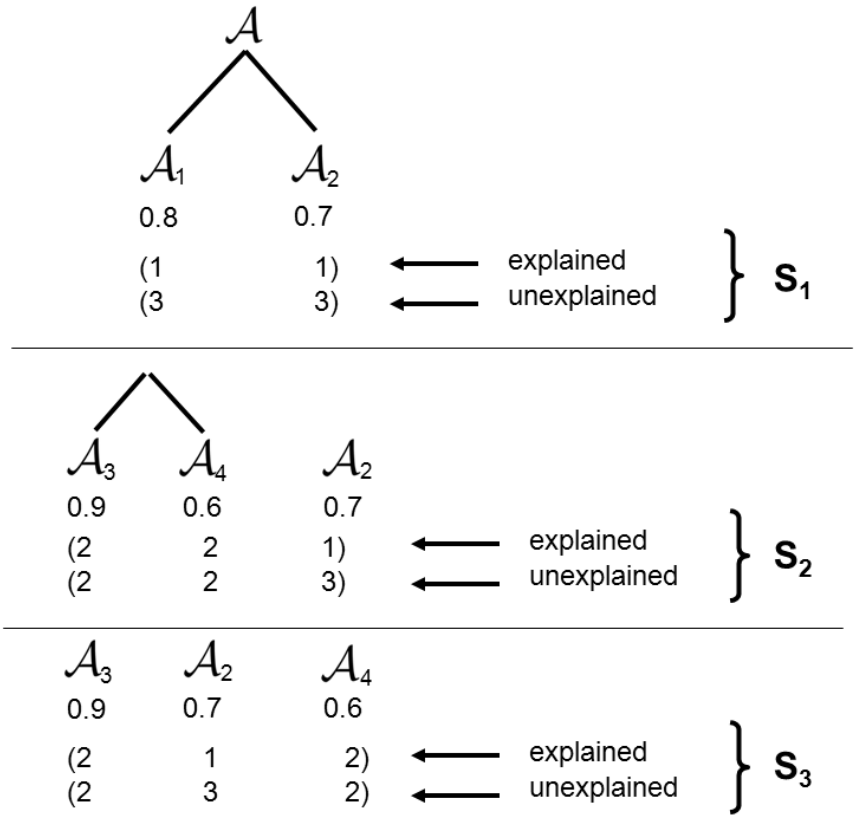


Figure 4.1: Example for branching and depth controlling

In the above example, we show branching and depth controlling in this work. At each step, only a fiat assertion of the Abox with the maximum score is explained. Thus, we avoid the generation of many interpretation Aboxes with possibly lower scores. The procedure realizes a kind of beam search and is analogous to the procedure implemented in [BKN11] as explained in Section 3.4. What is different is the way we compute the scoring values. This example is similar to the example given in [Has11].

4.2.3 Controlling Reactivity

Controlling reactivity is the processing procedure of bunches of input data. Since the observations arrive incrementally [HBB+13,GMN+10c], we have to apply approaches to incrementally process data. In this section, we discuss controlling reactivity procedure applied in this work. We apply two different approaches when a new bunch of assertions arrives:

- stop-processing
- non-stop-processing

According to the stop-processing approach, we stop processing the current bunch of assertions when the following bunch arrives. Thus, the remaining unexplained fiat assertions of the current bunch are not explained any more.

According to the non-stop-processing approach, we continue processing the unexplained fiat assertions of the current bunch although the following bunch has already arrived and waits for processing in the analysis queue.

Each approach has advantages and disadvantages. The disadvantage of the stop-processing approach is that we might lose some data since we do not explain the remaining unexplained fiat assertions. The advantage of this approach is that we switch quickly to the new input data and might gain more important data.

The disadvantage of the non-stop-processing approach is that the processing of some bunches are delayed since the previous bunches might have many assertions to be explained. Thus, with delay we will gain the new information which might be very important. The advantage of this approach is that all observations are explained completely and we do not lose any data.

4.3 Comparison with Independent Choice Logic

In this section, we compare the preference score in this work with the one used for Independent Choice Logic [Poo91]. In both contexts, preference scores are used for two different purposes. Poole's preference score is defined for diagnostic frameworks whereas in this work the preference score is defined for controlling Abox abduction in the context of multimedia interpretation. According to the Poole's approach, the probability of observations *obs* given hypothesis D is 1:

$$P(obs \mid D) = 1 \tag{4.12}$$

where each hypothesis $D = \{h_1, \ldots, h_n\}$ is determined according to a rule in ICL. In order to determine hypothesis D, we consider only the rules where the predicate of the observation matches with the head of the rule. The following example shows how an explanation for a set of observations is determined. Let us consider the observation $obs = \{carEntry\}$ and the following rules:

$$carEntry \leftarrow car \wedge doorSlam \wedge carEntryIfCarDoorSlam \tag{4.13}$$

$$carEntry \leftarrow car \wedge \neg doorSlam \wedge carEntryIfCarNoDoorSlam \tag{4.14}$$

Two explanations for $obs = \{carEntry\}$ are:

$$D_1 = \{car, doorSlam, carEntryIfCarDoorSlam\} \tag{4.15}$$

$$D_2 = \{car, \neg doorSlam, carEntryIfCarNoDoorSlam\} \tag{4.16}$$

In this work, the observations are uncertain and we support the observations by the interpretation procedure. Thus, it holds that

$$P(obs \mid D) \neq 1 \tag{4.17}$$

Additionally, the instances of hypotheses in Poole's approach are logically independent, which does not hold in our work. The preference score in [Poo91] is defined according to the Maximum Likelihood approach [RN03] as follows:

$$argmax_D P(D \mid obs) \tag{4.18}$$

whereas the preference score of this work is determined according to the Maximum A Posteriori approach [DLK+08]:

$$argmax_D P(obs \mid D) \tag{4.19}$$

Note that the determination of the scoring value depends on the context it is used for. We cannot apply Poole's approach in this work since the logical independency does not hold among the assertions of an explanation in our context. The other important difference is that Poole determines the probability of an explanation whereas our approach is more general and determines the probability of an interpretation, which still might contain unexplained assertions. Additionally, unlike Poole, we deal with uncertain data $P(obs \mid D) \neq 1$.

4.4 Conversion of the Knowledge Base into ML Notation

The applied knowledge base $\Sigma = (\mathcal{T}, \mathcal{A})$ in this work is a tuple composed of a Tbox $\mathcal{T}$ consisting of Description Logic axioms and an Abox $\mathcal{A}$ consisting of weighted/strict assertions. In this section, we discuss that the Tbox axioms do not have any influence on the determination of the scoring values of the interpretation Aboxes. Thus, the Tbox axioms are not converted into the Markov logic notation.

The other reason why we do not consider Tbox axioms during the determination of the scoring values is that they weaken the influence of the weighted rules and do not allow the weighted rules to play any role. Considering the Tbox axioms during the determination of the scoring values sets the scoring value of each interpretation Abox to 1 and it is not any more possible to see how the scoring values step by step increase by explaining the fiat assertions.

Example 9 Let us assume the following Abox $\mathcal{A}$:

$$\mathcal{A} = \{Car(c_1), DoorSlam(ds_1), causes(c_1, ds_1)\}$$

Furthermore, we consider the next backward chaining rule:

$$\begin{aligned} \forall x, y\ causes(x, y) \quad \leftarrow \quad & \exists z\ CarEntry(z), hasObject(z, x), \\ & hasEffect(z, y), Car(x), DoorSlam(y) \end{aligned}$$

By applying the above backward chaining rule, the following interpretation Abox $\mathcal{A}'$ is generated:

$$\mathcal{A}' = \mathcal{A} \cup \{CarEntry(ind_{42}), hasObject(ind_{42}, c_1), hasEffect(ind_{42}, ds_1)\}$$

Moreover, let us assume the Tbox axiom:

$$\begin{aligned} \forall z\ CarEntry(z) \quad \Rightarrow \quad & Movement(z) \wedge \\ & \exists x\,[hasObject(z, x) \Rightarrow Car(x)] \wedge \\ & \exists y\,[hasEffect(z, y) \Rightarrow DoorSlam(y)] \end{aligned}$$

and the weighted rule:

$$\begin{aligned} 5\ \forall z, x, y\ CarEntry(z) \wedge hasObject(z, x) \wedge hasEffect(z, y) \Rightarrow \\ Car(x) \wedge DoorSlam(y) \wedge causes(x, y) \end{aligned}$$

Thus, the scoring value of $\mathcal{A}'$ is:

$$P(\mathcal{A}', \mathcal{WR}) = 1$$

This example shows that by considering the Tbox axioms in the knowledge base, the scoring values of the interpretation Aboxes are 1. Thus, with strict Tbox axioms considered it is not possible any more to sensibly compare the scoring values of the generated interpretation Aboxes.

We emphasize that the Tbox is used during the abduction process as a filter to remove the inconsistent interpretation Aboxes. Thus, for the determination of the scoring values, a Tbox is not required.

According to the above discussion, we consider only the weighted rules $\mathcal{WR}$ for the determination of the scoring values of Abox $\mathcal{A}$. Thus, we use the notation $P(\mathcal{A}, \mathcal{WR})$ for the determination of scores. In the following, the function $AlchemyMLN(\mathcal{A}, \mathcal{WR})$ is given which describes how weighted rules $\mathcal{WR}$ and the Abox $\mathcal{A}$ are converted to Alchemy's Markov logic notation [KSR$^+$10]. See Appendix A for details about the systems. The function $AlchemyMLN$ takes as input two parameters, namely an Abox $\mathcal{A}$, and a set of weighted rules $\mathcal{WR}$. Based on Alchemy's processing techniques, the type of each concept must be specified where the type is a non-empty set containing constants of that type. Note that constants in the context of Markov logic formalism [DR07] correspond to individuals in Description Logics [BCM$^+$03]. In case of a resulting empty set for certain types, individuals are randomly generated and assigned to the extension of the types. Similarly, the domain and range types of each role are specified by non-empty types. In case of having empty types, the MLN cannot be constructed by Alchemy [KSR$^+$10].

The output of the $AlchemyMLN$ function is a knowledge base $\Sigma = (\mathbf{T}, \mathbf{P}, \mathbf{F})$ containing three sets $\mathbf{T}$, $\mathbf{P}$ and $\mathbf{F}$ which indicate the type set, predicate set, and the formula

set, respectively. At the beginning of the *AlchemyMLN* function these sets are initialized to empty sets. Since the role names in the Markov logic notation begin with capital letters [KSR$^+$10], the function *CapitalLetter* is applied to the role names of the DL-knowledge base. The set of formulas **F** contains the weighted rules. In the next algorithm, we define types for all concepts and roles mentioned in the weighted rules.

Algorithm 5: The knowledge base conversion algorithm

Function AlchemyMLN($\mathcal{A}$, $\mathcal{WR}$):
Input: an Abox $\mathcal{A}$, and the set of weighted rules $\mathcal{WR}$
Output: a knowledge base $\Sigma = (\mathbf{T}, \mathbf{P}, \mathbf{F})$ with the type set $\mathbf{T}$, predicate set $\mathbf{P}$, and the formula set $\mathbf{F}$

$\Sigma = (\emptyset, \emptyset, \emptyset)$;
foreach $C \in Concepts(\mathcal{WR})$ **do**
  $t := GenerateNewTypeName()$;
  $q := \{(x) \mid C(x)\}$;
  $result = evaluate(q)$;
  if $result = \emptyset$ **then**
    $NewInd := GenerateNewIndividual()$;
    $\mathbf{T} := \mathbf{T} \cup \{t + " = " + "\{" + NewInd + "\}"\}$;
  else
    $\mathbf{T} := \mathbf{T} \cup \{t + " = " + "\{" + CapitalLetter(result) + "\}"\}$;
  $\mathbf{P} = \mathbf{P} \cup \{C(t)\}$;
foreach $r \in Roles(\mathcal{WR})$ **do**
  $t_1 = GenerateNewTypeName()$;
  $t_2 = GenerateNewTypeName()$;
  $q := \{(x, y) \mid r(x, y)\}$;
  $result = evaluate(q)$;
  if $result = \emptyset$ **then**
    $NewInd_1 := GenerateNewIndividual()$;
    $\mathbf{T} := \mathbf{T} \cup \{t_1 + " = " + "\{" + NewInd_1 + "\}"\}$;
    $NewInd_2 := GenerateNewIndividual()$;
    $\mathbf{T} := \mathbf{T} \cup \{t_2 + " = " + "\{" + NewInd_2 + "\}"\}$;
  else
    $\mathbf{T} := \mathbf{T} \cup \{t_1 + " = " + "\{" + CapitalLetter(MapFirst(result)) + "\}"\}$;
    $\mathbf{T} := \mathbf{T} \cup \{t_2 + " = " + "\{" + CapitalLetter(MapSecond(result)) + "\}"\}$;
  $R = CapitalLetter(r)$;
  $\mathbf{P} = \mathbf{P} \cup \{R(t_1, t_2)\}$;
foreach $wr \in \mathcal{WR}$ **do**
  $\mathbf{F} = \mathbf{F} \cup \{wr\}$;
return Σ;

The generated knowledge base Σ is saved in a file with the extension .mln [KSR+10]. In the above algorithm, $Concepts(\mathcal{WR})$ and $Roles(\mathcal{WR})$ return a set containing concepts and roles mentioned in the weighted rules $\mathcal{WR}$, respectively.

The function *GenerateNewTypeName* generates type names beginning with non-capital letters and the function *GenerateNewIndividual* generates individual names beginning with capital letters.

The functions *MapFirst* and $MapSecond$ are defined as follows:

$$MapFirst((X,Y)) := \bigcup_{x \in X} \{x\} = X \tag{4.20}$$

$$MapSecond((X,Y)) := \bigcup_{y \in Y} \{y\} = Y \tag{4.21}$$

The functions *MapFirst* and $MapSecond$ take as parameter a set of pairs (X, Y) and return a set consisting of the first and the second elements of the pairs, respectively.

The next query which is applied in the previous algorithm determines instances which have the same type:

$$q = \{(X) \mid C(X)\}$$

Let us consider the next query which retrieves the instances of the concept Car:

$$q = \{(X) \mid Car(X)\}$$

Furthermore, let us assume that the result of the above query is:

$$r = \{c_1, c_2\}$$

Let us consider the following query:

$$q = \{(X,Y) \mid causes(X,Y)\}$$

and let us assume that the result of the query q is as follows:

$$r = \{(c_1, ds_1), (c_2, ds_2)\}$$

Applying *MapFirst* and $MapSecond$ returns the domain and range instances of *causes*, respectively:

$$\begin{aligned} MapFirst(r) &= \{c_1, c_2\} \\ MapSecond(r) &= \{ds_1, ds_2\} \end{aligned}$$

In the following, we discuss how the Abox assertions are converted into a proper form for inference according to the Markov logic formalism [DR07]. For Markov logic inference, there is a separate file called evidence file or database file with extension .db [KSR+10]. The database file contains evidences which are strict assertions of $\mathcal{A}$. The Abox which

should be converted into an evidence file is consistent and contains weighted/strict assertions where some assertions are fiat assertions. Since the evidence file contains only strict assertions, the most probable world given the Abox is determined. Then, the positive assertions of the most probable world existing in $\mathcal{A}$ are selected. Among the positive assertions, there are some fiat assertions. The fiat assertions are used for the generation of the query at each step. Except fiat assertions, the remaining positive assertions stand as strict assertions in the evidence file. The following table depicts the correspondence among the strict Abox assertions in Description Logics [BCM+03] and in Markov logic notation [KSR+10]. Note that according to the Markov logic notation the role names and individual names begin with capital letters.

Abox assertion	Markov logic evidence
$A(ind)$	$A(Ind)$
$r(ind_1, ind_2)$	$R(Ind_1, Ind_2)$

Table 4.2: The correspondence among the Abox assertions in DL and in MLN

In the above table, A and r indicate an atomic concept name and an atomic role name, respectively. The algorithm *AlchemyDB* describes how the evidence file is generated. This algorithm takes as parameter the set of concept names $\mathbf{C}$, the set of role names $\mathbf{R}$, Tbox $\mathcal{T}$ and an Abox $\mathcal{A}$. The Abox $\mathcal{A}$ contains weighted/strict assertions where some assertions are fiat assertions. The output of this algorithm is an evidence vector $\vec{e}$ which contains strict non-fiat assertions. The evidence vector is the content of an evidence file with extension `.db`. At the beginning of the algorithm, the evidence vector is initialized to an empty set. In order to consider only positive assertions of the most probable world given the Abox $\mathcal{A}$, *Select* and MAP functions are applied. Since according to the Markov logic formalism, the individual names and the role names begin with capital letters [KSR+10], we apply the function *CapitalLetter* to these names. Each role assertion is examined whether it is a non-fiat assertion. In this case, it is added to the evidence vector $\vec{e}$. The fiat assertions are considered for the query generation.

Algorithm 6: The database generation algorithm

Function AlchemyDB($\mathbf{C}, \mathbf{R}, \mathcal{T}, \mathcal{A}$):
Input: a set of concept names $\mathbf{C}$, a set of role names $\mathbf{R}$, a Tbox $\mathcal{T}$, and an Abox $\mathcal{A}$
Output: an evidence vector $\vec{e}$
$\vec{e} = \emptyset$;
$\mathcal{A} = Select(MAP(\mathcal{A}, \mathcal{T}), \mathcal{A})$;
foreach $C(ind) \in \mathcal{A}$ **where** $C \in \boldsymbol{C}$ **do**
  $\vec{e} = \vec{e} \cup \{C(Ind)\}$;
foreach $r(ind_1, ind_2) \in \mathcal{A}$ **where** $r \in \boldsymbol{R}$ **and** $fiat(r(ind_1, ind_2)) = false$ **do**
  $R = CapitalLetter(r)$;
  $\vec{e} = \vec{e} \cup \{R(Ind_1, Ind_2)\}$;
return $\vec{e}$;

In Algorithm 6, we did not consider the fiat assertions of the Abox $\mathcal{A}$ for the generation of the evidence file. Furthermore, we mentioned that the fiat assertions are considered for the generation of the queries. The following algorithm defines how the query is generated. A query is generated based on the fiat assertions of the Abox $\mathcal{A}$. We assume that some role assertions are fiat assertions. The input to the algorithm *AlchemyQuery* is the Abox $\mathcal{A}$ and the output is the query Q. Since the role names and individual names in MLN notation begin with capital letters [KSR+10], *CapitalLetter* function is applied to these names. Again we have to determine the most probable world and select the positive assertions of the most probable world. At each step, the fiat assertions are divided into explained and unexplained fiat assertions. Q is an *explained* fiat assertion of the Abox $\mathcal{A}$. In the following algorithm, we select an explained fiat assertion of $\mathcal{A}$, randomly. We assume that the probability of an *unexplained* fiat assertion to be true is 0.5. Thus, we do not generate queries for unexplained fiat assertions. Consequently, we reduce the required time for query generation and query answering of unexplained fiat assertions. In the following algorithm, the function $Explained\_Fiats(\mathcal{A})$ returns a set containing explained fiat assertions of $\mathcal{A}$.

Algorithm 7: The query generation algorithm

Function AlchemyQuery($\mathcal{A}$,$\mathcal{T}$):
Input: an Abox $\mathcal{A}$, and a Tbox $\mathcal{T}$
Output: a query Q
$A = Select(MAP(\mathcal{A}, \mathcal{T}), \mathcal{A})$;
$r(ind_1, ind_2) = random\_select(Explained\_Fiats(\mathcal{A}))$;
$Explained\_Fiats(\mathcal{A}) = Explained\_Fiats(\mathcal{A}) \setminus \{r(ind_1, ind_2)\}$;
$R = CapitalLetter(r)$;
$I_1 = CapitalLetter(ind_1)$;
$I_2 = CapitalLetter(ind_2)$;
$Q = R(I_1, I_2)$;
return Q;

Example 10 The next example shows how the equivalence of an Abox $\mathcal{A}$ in Markov logic notation is determined. Let us assume the following sets of concept - and role names which are determined according to the Tbox $\mathcal{T}$:

$$\begin{aligned} \mathbf{C} &= \{Car, DoorSlam, CarEntry, CarExit\} \\ \mathbf{R} &= \{causes, hasObject, hasEffect\} \end{aligned}$$

Consider the next Abox:

1.3 $Car(c_1)$
1.2 $DoorSlam(ds_1)$
$causes(c_1, ds_1)$
$CarEntry(ind_{42})$
$hasObject(ind_{42}, c_1)$
$hasEffect(ind_{42}, ds_1)$

Table 4.3: Example for an Abox

Unlike in description logics [BCM+03], the types must be specified according to the Markov logic formalism [DLK+08]. The types are non-empty sets which contain individuals. In order to determine the constants of a concept type, the following queries are generated:

$$\begin{aligned} q_1 &= \{(X) \mid Car(X)\} \\ q_2 &= \{(X) \mid DoorSlam(X)\} \\ q_3 &= \{(X) \mid CarEntry(X)\} \\ q_4 &= \{(X) \mid CarExit(X)\} \end{aligned}$$

The results of the above queries are as follows:

$$\begin{aligned} result_1 &= \{c_1\} \\ result_2 &= \{ds_1\} \\ result_3 &= \{ind_{42}\} \\ result_4 &= \emptyset \end{aligned}$$

Similarly, in order to determine the constants of a domain and range type of a role, the next queries are generated:

$$\begin{aligned} q_5 &= \{(X,Y) \mid causes(X,Y)\} \\ q_6 &= \{(X,Y) \mid hasObject(X,Y)\} \\ q_7 &= \{(X,Y) \mid hasEffect(X,Y)\} \end{aligned}$$

The results of the above queries are as follows:

$$\begin{aligned} result_5 &= \{(c_1, ds_1)\} \\ result_6 &= \{(ind_{42}, c_1)\} \\ result_7 &= \{(ind_{42}, ds_1)\} \end{aligned}$$

In order to determine the individuals which belong to the domain type of each relation the *MapFirst* function should be applied:

$$\begin{aligned} MapFirst(result_5) &= \{c_1\} \\ MapFirst(result_6) &= \{ind_{42}\} \\ MapFirst(result_7) &= \{ind_{42}\} \end{aligned}$$

Similarly, in order to determine the individuals which belong to the range type of each relation the $MapSecond$ function should be applied:

$$\begin{aligned} MapSecond(result_5) &= \{ds_1\} \\ MapSecond(result_6) &= \{c_1\} \\ MapSecond(result_7) &= \{ds_1\} \end{aligned}$$

Thus, in total there are 10 types $t_1, \ldots, t_{10}$ where $t_1, \ldots, t_4$ indicate the types of the concepts Car, $DoorSlam$, $CarEntry$, and $CarExit$, and t_5, t_6, t_7 denote the domain types of the relations *causes*, *hasObject*, and *hasEffect* and t_8, t_9, t_{10} denote the range type of these relations, respectively. Since $result_4$ is an empty set, a new individual $newInd_1$ is generated and assigned to t_4. Since the individual names begin with capital letters [KSR+10], the function $CapitalLetter$ is applied to the individual names. The type set $\mathbf{T}$ contains the generated types and the corresponding assigned individuals. Unlike the type names which do not begin with capital letters, the individual names begin with capital letters. In the following, the knowledge base $\Sigma = (\mathbf{T}, \mathbf{P}, \mathbf{F})$ generated by the $AlchemyMLN$ function is given where $\mathbf{T}$, $\mathbf{P}$, $\mathbf{F}$ indicate the type set, the predicate set and the formula set, respectively. The knowledge base Σ is in a file with extension .mln called in this example example.mln:

$t_1 = \{C_1\}$
$t_2 = \{Ds_1\}$
$t_3 = \{Ind_{42}\}$
$t_4 = \{NewInd_1\}$
$t_5 = \{C_1\}$
$t_6 = \{Ind_{42}\}$
$t_7 = \{Ind_{42}\}$
$t_8 = \{Ds_1\}$
$t_9 = \{C_1\}$
$t_{10} = \{Ds_1\}$

$Car(t_1)$
$DoorSlam(t_2)$
$CarEntry(t_3)$
$CarExit(t_4)$
$Causes(t_5, t_8)$
$HasObject(t_6, t_9)$
$HasEffect(t_7, t_{10})$

$5 \;\; CarEntry(z) \wedge HasObject(z, x) \wedge HasEffect(z, y) \Rightarrow Car(x) \wedge DoorSlam(y) \wedge Causes(x, y)$
$5 \;\; CarExit(z) \wedge HasObject(z, x) \wedge HasEffect(z, y) \Rightarrow Car(x) \wedge DoorSlam(y) \wedge Causes(x, y)$

Table 4.4: example.mln file

Note that the .mln file contains only the weighted rules. The forward and the backward chaining rules are not in this file. The *AlchemyDB* function generates the evidence vector $\vec{e}$ which builds the database file with the extension .db. This file contains only the strict non-fiat assertions and in this example is called evidence.db.

$Car(C_1)$
$DoorSlam(Ds_1)$
$CarEntry(Ind_{42})$
$HasObject(Ind_{42}, C_1)$
$HasEffect(Ind_{42}, Ds_1)$

Table 4.5: The database file evidence.db

The *AlchemyQuery* function generates the query Q which is an explained fiat assertion of the Abox. Since there is only one explained fiat assertion in the Abox, the query is $Q = Causes(C_1, Ds_1)$ and the conditional probability we are interested in is:

$$P(Causes(C_1, Ds_1) \mid \vec{e}) \tag{4.22}$$

The command which determines the above conditional probability in Alchemy system [KSR+10] is:

infer -i example.mln -r example.result -e evidence.db -q $Causes(C_1, Ds_1)$

The value of the conditional probability expression given in 4.22 is the scoring value of the Abox $\mathcal{A}$ and is given in an output file. In the following, the output file of the query 4.22 is given which is called in this example example.result. Note that the output file has the extension .result.

$Causes(C_1, Ds_1)$	0.91

Table 4.6: The output file example.result

The output file example.result indicates that:

$$P(Causes(C_1, Ds_1) \mid \vec{e}) = 0.91$$

The output file example.result for query Q has the following form:

Q	p

Table 4.7: The general form of the output file

The above output file has one row and two columns. The first column indicates the query Q and the second column denotes the corresponding probability p where:

$$p = P(Q \mid \vec{e})$$

where $\vec{e}$ indicates the evidence vector. The probability value we are interested in is p.

Chapter 5

Evaluation

In the previous chapter, we presented an approach for Abox abduction using probabilistic branch and depth control. In this chapter, we evaluate the results of our approach. The performance and the quality of the results are evaluated. The experiments were run on a Linux CALLEO 552 server (Ubuntu 10.04.3 LTS) with an AMD Eight-Core Opteron 6136 (2.4 GHz) processor and 128 GB of main memory. In these experiments, a set of videos from the environmental domain has been used. The videos were analyzed by state-of-the-art analysis tools [HBB+13]. The analysis results which are sent incrementally build an Abox.

The analysis results are sent to the interpretation system which employs the probabilistic interpretation algorithm presented in the previous chapter.

5.1 Optimization Techniques

A Markov logic network [DR07] consists of a set of weighted/strict first-order formulas and a set of constants. In this work, the weighted rules are considered as formulas, and the individuals mentioned in the observations or in the hypothesized assertions are the constants of the network. These constants are associated to the corresponding predicate types. A predicate type is a non-empty set. Thus, for the remaining predicates of the weighted rules with no constants, new constants are generated and assigned. The newly introduced constants represent objects of a certain predicate which might be used later during the inference.

In our experiments, the constructed Markov logic networks were quite large, despite the fact that not too many weighted rules were specified. Since for answering a particular query the entire network is not required, we construct only a relevant subnetwork. Thus, we vastly reduce time and memory space required for inference. The required constants for constructing the subnetwork are mentioned in the observations and in the hypothesized assertions. According to the predicates which appear in the observations and the hypothesized assertions, the relevant subset of the weighted rules for answering a particular query is determined. The selected weighted rules are the

relevant formulas for constructing the subnetwork. The remaining weighted rules are irrelevant and are left out consequently. The approach explained above is inspired by knowledge-based model construction (KBMC) approaches (e.g., [NH97]) where only a fraction of the entire knowledge base is selected and instantiated. The selected fraction is relevant for answering a particular query. Thus, based on the interpretation Abox $\mathcal{A}$, we determine the required knowledge base Σ for the inference procedure. Consequently, the considered knowledge base is a function of the interpretation Abox $\Sigma(\mathcal{A})$. The KBMC approach is also applied for inference in the context of Markov logic networks in [DLK+08]. According to [DLK+08], the results are the same if we consider Σ or $\Sigma(\mathcal{A})$ as the knowledge base.

For optimization purposes, Alchemy [KSR+10] uses sampling algorithms for inference e.g. MC-SAT [PD06], Gibbs sampling [GG84]. Unfortunately, the sampling algorithms implemented in Alchemy do not provide for required optimizations if we consider a knowledge base with many axioms and too many assertions. Thus, in this work we had to provide the Alchemy engine with KBMC reduced set of formulas for optimization purposes.

Example 11 In this example, we show how the KBMC approach is applied in the context of this work. During the experiments, the query and the database file are constant but the number of weighted rules to be considered changes. Let us assume the following weighted rules:

$$
\begin{aligned}
5 \quad & \forall z, x, y \; CarEntry(z) \wedge HasObject(z, x) \wedge HasEffect(z, y) \Rightarrow \\
& \quad Car(x) \wedge DoorSlam(y) \wedge Causes(x, y) \\
0.5 \quad & \forall z, x, y \; CarExit(z) \wedge HasObject(z, x) \wedge HasEffect(z, y) \Rightarrow \\
& \quad Car(x) \wedge DoorSlam(y) \wedge Causes(x, y) \\
3 \quad & \forall z, x, y \; EnvWorkshop(z) \wedge HasSubEvent(z, x) \wedge HasLocation(z, y) \Rightarrow \\
& \quad CarEntry(x) \wedge Building(y) \wedge OccursAt(x, y) \\
0.5 \quad & \forall z, x, y \; EnvProt(z) \wedge HasEvent(z, x) \wedge HasTheme(z, y) \Rightarrow \\
& \quad EnvConference(x) \wedge Env(y) \wedge HasTopic(x, y) \\
5 \quad & \forall z, x, y \; RenewableEnergy(z) \wedge HasPart(z, x) \wedge HasPart(z, y) \Rightarrow \\
& \quad Energy(x) \wedge Winds(y) \wedge EnergyToWinds(x, y) \\
& \qquad \vdots
\end{aligned}
$$

Furthermore, let us assume that the database file of the Abox $\mathcal{A}$ contains the following assertions:

$$
\begin{aligned}
\vec{e}(\mathcal{A}) \quad = \quad & \{Car(C_1), DoorSlam(DS_1), CarEntry(Ind_{42}), HasObject(Ind_{42}, C_1), \\
& HasEffect(Ind_{42}, DS_1), Building(Ind_{43}), EnvWorkshop(Ind_{45}), \\
& HasSubEvent(Ind_{45}, Ind_{42}), HasLocation(Ind_{45}, Ind_{43})\}
\end{aligned}
$$

There are two explained fiat assertions in $\mathcal{A}$, namely:

$$Causes(C_1, DS_1), OccursAt(Ind_{42}, Ind_{43})$$

Then, the queries are:

$$\begin{aligned} Q_1(\mathcal{A}) &= \langle Causes(C_1, DS_1) = true \rangle \\ Q_2(\mathcal{A}) &= \langle OccursAt(Ind_{42}, Ind_{43}) = true \rangle \end{aligned}$$

In the following, we determine the values of the above mentioned queries:

$$\begin{aligned} P_{MLN}(\mathcal{A}, \mathcal{WR})(Q_1(\mathcal{A}) \mid \vec{e}(\mathcal{A})) &= 0.83 \\ P_{MLN}(\mathcal{A}, \mathcal{WR})(Q_2(\mathcal{A}) \mid \vec{e}(\mathcal{A})) &= 0.76 \end{aligned}$$

The scoring value of $\mathcal{A}$ is determined according to the next formula:

$$\begin{aligned} P(\mathcal{A}, \mathcal{WR}) &= \frac{1}{2}\left(P_{MLN}(\mathcal{A}, \mathcal{WR})(Q_1(\mathcal{A}) \mid \vec{e}(\mathcal{A})) + P_{MLN}(\mathcal{A}, \mathcal{WR})(Q_2(\mathcal{A}) \mid \vec{e}(\mathcal{A}))\right) \\ &= 0.80 \end{aligned}$$

For answering the query, only the first three weighted rules are relevant since the predicates in the database and in the query are mentioned only in the first three weighted rules. Thus, the remaining rules are irrelevant and can be left out. Thus, we reduce the knowledge base and the problem complexity.

In the next table, the results of removing more and more irrelevant rules are given:

Number of irrelevant weighted rules	Inference value
0	0.80
1	0.80
2	0.80
3	0.80
⋮	⋮

Table 5.1: Results of the KBMC approach in this example

The above table shows that the inference value does not change by increasing the number of irrelevant weighted rules. Thus, we have the same results with $\Sigma(\mathcal{A})$ and Σ.

According to Algorithm 5, the performance of the Alchemy engine [KSR+10] decreases with a large set of weighted rules. In the following, we reduce the size of the knowledge base (the weighted rules).

The required subset of the weighted rules is determined according to the interpretation Abox $\mathcal{A}$. A weighted rule W is considered as relevant if a mentioned predicate in

$\mathcal{A}$ is a conjunct of W. According to Algorithm 5, the type of each predicate mentioned in the weighted rules is defined.

Note that the number of the relevant weighted rules increases during the inference procedure since newly generated hypothesized assertions are added to the interpretation Abox. Thus, the new predicates mentioned in the Abox leads to considering more relevant weighted rules. Consequently, the considered subnetwork increases over the time. Accordingly, we define types for the concepts and relations mentioned in the newly considered weighted rules.

The next algorithm is an improvement of Algorithm 5 where we determine the relevant weighted rules. Accordingly, we determine the required predicates. Thus, the knowledge base and consequently the runtime are reduced.

Algorithm 8: The knowledge base conversion algorithm

Function AlchemyMLN($\mathcal{A}$, $\mathcal{WR}$):
Input: an Abox $\mathcal{A}$, and the set of weighted rules $\mathcal{WR}$
Output: a knowledge base $\Sigma = (\mathbf{T}, \mathbf{P}, \mathbf{F})$ with the type set $\mathbf{T}$, predicate set $\mathbf{P}$, and the formula set $\mathbf{F}$

$\Sigma = (\emptyset, \emptyset, \emptyset)$;
foreach $\alpha \in \mathcal{A}$ **do**
  foreach $wr \in \mathcal{WR}$ **do**
    if $predicateSym(\alpha) \in predicateSymbols(wr)$ **then**
      $\mathbf{F} = \mathbf{F} \cup \{wr\}$;
foreach $C \in Concepts(\boldsymbol{F})$ **do**
  $t := GenerateNewTypeName()$;
  $q := \{(x) \mid C(x)\}$;
  $result = evaluate(q)$;
  if $result = \emptyset$ **then**
    $NewInd := GenerateNewIndividual()$;
    $\mathbf{T} := \mathbf{T} \cup \{t + " = " + "\{" + NewInd + "\}"\}$;
  else
    $\mathbf{T} := \mathbf{T} \cup \{t + " = " + "\{" + CapitalLetter(result) + "\}"\}$;
  $\mathbf{P} = \mathbf{P} \cup \{C(t)\}$;
foreach $r \in Roles(\boldsymbol{F})$ **do**
  $t_1 = GenerateNewTypeName()$;
  $t_2 = GenerateNewTypeName()$;
  $q := \{(x, y) \mid r(x, y)\}$;
  $result = evaluate(q)$;
  if $result = \emptyset$ **then**
    $NewInd_1 := GenerateNewIndividual()$;
    $\mathbf{T} := \mathbf{T} \cup \{t_1 + " = " + "\{" + NewInd_1 + "\}"\}$;
    $NewInd_2 := GenerateNewIndividual()$;
    $\mathbf{T} := \mathbf{T} \cup \{t_2 + " = " + "\{" + NewInd_2 + "\}"\}$;
  else
    $\mathbf{T} := \mathbf{T} \cup \{t_1 + " = " + "\{" + CapitalLetter(MapFirst(result)) + "\}"\}$;
    $\mathbf{T} := \mathbf{T} \cup \{t_2 + " = " + "\{" + CapitalLetter(MapSecond(result)) + "\}"\}$;
  $R = CapitalLetter(r)$;
  $\mathbf{P} = \mathbf{P} \cup \{R(t_1, t_2)\}$;
return Σ;

where $predicateSym(\alpha)$ returns the mentioned predicate p in the assertion α:

$$predicateSym(\alpha) = p \tag{5.1}$$

Furthermore, $predicateSymbols(wr)$ returns a set containing predicates mentioned in the weighted rule wr:

$$predicateSymbols(wr) = \{p \mid p \text{ is a mentioned predicate in } wr\} \tag{5.2}$$

Moreover, $Concepts(\mathbf{F})$ and $Roles(\mathbf{F})$ return a set containing concepts and roles mentioned in the formula set, respectively. The above algorithm reduces runtime while preserving correct probability values.

5.2 Case Study: CASAM

The goal of the CASAM project [HBB+13] is the annotation of multimedia documents through the collaboration of human annotators and machine intelligence. Thus, it speeds up the task of manually annotation of multimedia documents. Additionally, it reduces the effort and increases the accuracy to attach annotations to the multimedia documents. The CASAM project is a European research project under the seventh framework programme (FP7) for research and technological development. CASAM stands for Computer Aided Semantic Annotation of Multimedia (FP7-217061). The main components of the CASAM project are KDMA, RMI and HCI. In the following, these components and their functionalities are described:

- The analysis component of CASAM called "Knowledge-Driven Multimedia Analysis" (KDMA) processes the multimedia documents by exploiting media analysis tools and detects objects within the multimedia document. The output of this component is an Abox which contains surface level information. The generated Aboxes are sent to the RMI component described below.

- The reasoning component of CASAM called "Reasoning for Multimedia Interpretation" (RMI) is the core of the CASAM project. This component receives assertions produced by KDMA and infers higher level interpretations of the multimedia content. The generated interpretations are sent to the next component called HCI. Additionally, in order to disambiguate the interpretations, queries are generated [GMN+10b, HBB+13] and sent to the KDMA and HCI components in order to indicate what would be relevant information to discriminate between similarly ranked interpretation possibilities. In this work, we do not discuss query generation and feedback issues.

- The end-user component of CASAM called "Human Computer Interaction" (HCI) displays the multimedia document to the user. Furthermore, the surface level assertions generated by KDMA and the interpretations generated by RMI are

presented to the user through a graphical user interface which was developed during the CASAM project [HBB+13]. The user decides whether the annotation process can be terminated. This will happen when the user recognizes that the multimedia document is sufficiently annotated.

The objective of the CASAM project is to develop a system for semantic annotation of multimedia documents. For the CASAM project, the environmental domain has been chosen as the application scenario [GMN+09b]. It is a large domain covering many aspects such as environmental pollution, conferences, catastrophes, hazards and conservation attempts. In order to represent general knowledge of this domain, a domain ontology has been developed. The set of concept and role names are called the signature of the ontology.

Two different ontologies have been defined during the CASAM project [HBB+13], namely the Environmental Domain Ontology (EDO) [GMN+09b] and the Multimedia Content Ontology (MCO) [GMN+11]. In EDO, we define the concepts and relations in terms of the environmental domain. The MCO defines the concepts and relations required for describing the structure of a multimedia document. The concepts and relations defined in the MCO are modality specific since each multimedia document might contain multiple modalities including audio, video, text. According to MCO, it is possible to indicate from which modality the assertions have been generated.

5.3 Hypotheses and Results

Before defining the hypotheses, we briefly explain the video analysis approach. A video document is divided into multiple video segments [HBB+13, GMN+11] which do not necessarily have the same length. Video segments (shots) are determined automatically according to certain patterns of scene changes in the video. Each video segment is analyzed by a set of video analysis tools.

The analysis component delivers description in a piecewise manner [HBB+13]. Each "piece" is a set of Abox assertions, which we call "bunch" here. A bunch of assertions contains the analysis results of a video segment. Since the video segments do not necessarily have the same length, the bunches of assertions do not arrive at equal time distances at well-defined time points. The other consequence is that the number of fiat assertions in the bunches is not necessarily the same. Let us consider two bunches of assertions where a bunch with many fiats is followed by a bunch with only a few fiats. Accordingly, there is a delay to handle the second bunch. Processing it in time might be very beneficial, however. Thus, the processing strategy must be adaptive.

The CASAM dataset is just an example for many similar datasets, with the characteristics that one bunch of assertions has not yet been processed completely while the next one already arrives. Thus, there is a tradeoff between spending computational power on completely interpreting a bunch and focusing on current observations, while

skipping possible interpretations for older input. Multiple strategies are investigated in experiments and the results of the experiments are described in the following.

Let us assume that t_i is the arrival time of bunch B_i. Similarly, the following bunch is B_{i+1} with the arrival time t_{i+1}. We assume furthermore that m indicates the maximum number of fiat assertions to be explained in a bunch. Let us assume that Bunch B_i has k fiat assertions. We consider t as the required time for explaining k fiats of B_i. In the following, we consider two different cases and we assume that fiat assertions of B_{i-1} are explained in time and there is no time overrun:

- $t_i + t \leq t_{i+1}$ and $k \leq m$: Thus, all fiat assertions of B_i are explained before t_{i+1} and there is no time overrun.
- $t_i + t > t_{i+1}$ and $k > m$: Thus, m fiat assertions of B_i are explained before t_{i+1}. The system behaviour at time point t_{i+1} is to stop explaining the remaining $k-m$ unexplained fiat assertions of B_i and start explaining the fiat assertions of B_{i+1}.

The selection of m affects the interpretation results. If m is too large, there might be a delay to explain fiat assertions of the following bunches. Thus, the interpretation results of the following bunches might be generated later than required. Since the interpretation results might be relevant, the system might react later than it should do. In this case, we ignore some bunches. If m is a small number, we might not explain some fiat assertions of a bunch. Thus, we might lose the corresponding interpretation results. There is a tradeoff between the maximum number of fiat assertions to be explained in a bunch and the number of bunches to be processed. A future work topic could be to discuss how variable m should be selected automatically.

Stop processing can arrive at any point of time since it is activated externally by the user. In the following, the next set of hypotheses and the evaluation results are given where the data set is the surface-level analysis results of a video with the topic *Wind Energy Approval*.

1. **By increasing the sampling size, the processing time increases, linearly.**

   The sampling size (maxSteps) is one of the Alchemy parameters used for the determination of the probability [KSR+10]. In order to have more accurate results, we increase the sampling size. In the next experiment, we determine the required time for the interpretation of a data set with 9 bunches of assertions. Figure 5.1 shows the required processing time as a function of sampling size according to MC-SAT algorithm [PD06]. The time is the accumulated time to process all bunches of assertions in seconds. Figure 5.1 shows that by increasing the sampling size, the processing time increases, linearly.

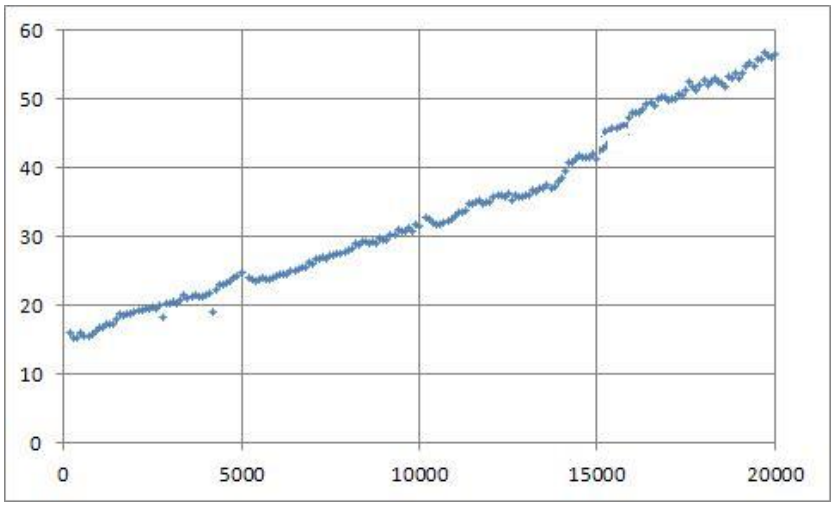


Figure 5.1: The number of samples (x) and the time (y) spent in seconds according to MC-SAT algorithm for the interpretation of a data set with 9 bunches of assertions.

2. **By increasing the sampling size, the final score converges to a limit.**

   In order to have more precise scores, we increase the sampling size (maxSteps). Sampling size (maxSteps) is one of the Alchemy parameters used for the determination of the probability [KSR+10]. Figures 5.2, and 5.3 depict the scores of the final interpretation Abox as a function of sampling size according to MC-SAT [PD06] and Gibbs sampling algorithms [GG84], respectively. Figures 5.2, and 5.3 show that by increasing the sampling size, the final score converges to a limit. Furthermore, the next figures show that MC-SAT algorithm is faster than Gibbs sampling to reach convergence. Additionally, the next figures show that Gibbs sampling requires more samples than MC-SAT algorithm to reach convergence.

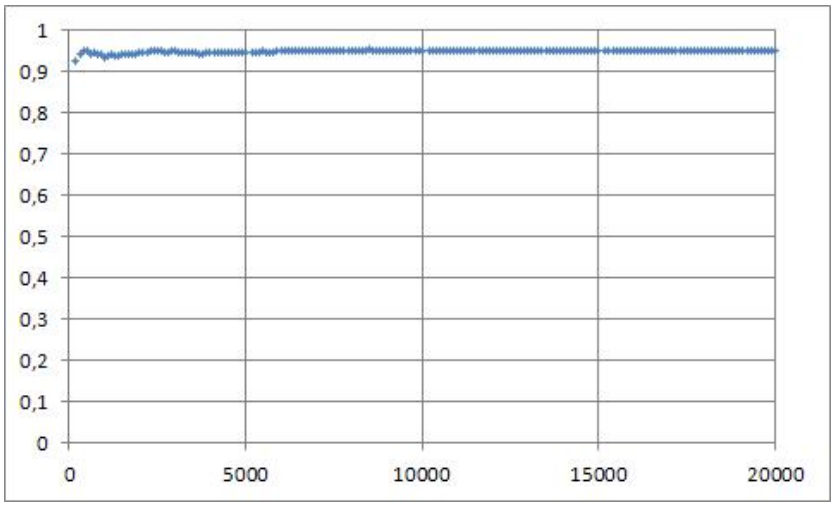


Figure 5.2: The number of samples (x) and the final score (y) determined according to MC-SAT algorithm for the interpretation of a data set with 9 bunches of assertions.

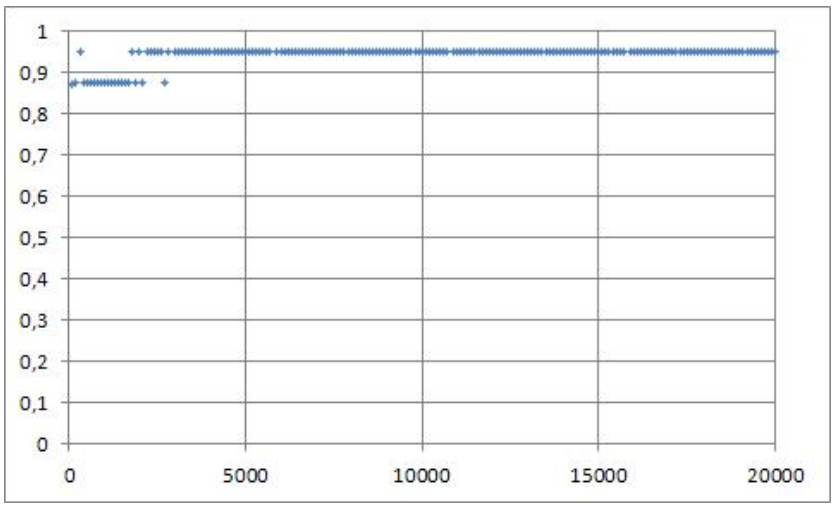


Figure 5.3: The number of samples (x) and the final score (y) determined according to Gibbs sampling algorithm for the interpretation of a data set with 9 bunches of assertions.

3. **By increasing the number of fiat assertions to be explained in a bunch, the number of ignored bunches increases. Additionally, the agenda size does not have any influence on the number of ignored bunches.**

   In order to control abduction depth procedure, we defined in this work variable m which indicates the maximum number of fiat assertions to be explained in a bunch of assertions. In the next experiment, we determine whether m and the

agenda size has influence on the number of ignored bunches. For this purpose, we iterate m and the agenda size in the next experiment. Figure 5.4 depicts the number of ignored bunches as a function of m and the agenda size. The results show that the agenda size does not have any influence on the number of ignored bunches. The reason is that there are not so many alternatives. Unlike the agenda size, m has an influence on the number of ignored bunches. According to the next figure, bunches are ignored when m is greater than 20. Thus, there is a tradeoff between the maximum number of fiat assertions to be explained in a bunch and the number of ignored bunches. By selecting a large value for m, we have many ignored bunches. Thus, we lose the corresponding interpretation results. By selecting a small value for m, all fiat assertions of a bunch might not be explained, completely. Thus, we might not have the complete interpretation results of each bunch. Consequently, selecting a small/large value for m affects the interpretation results. Note that the reason for ignoring bunches is that stop processing is activated, externally. In the next figure, the color represents the height of the surface.

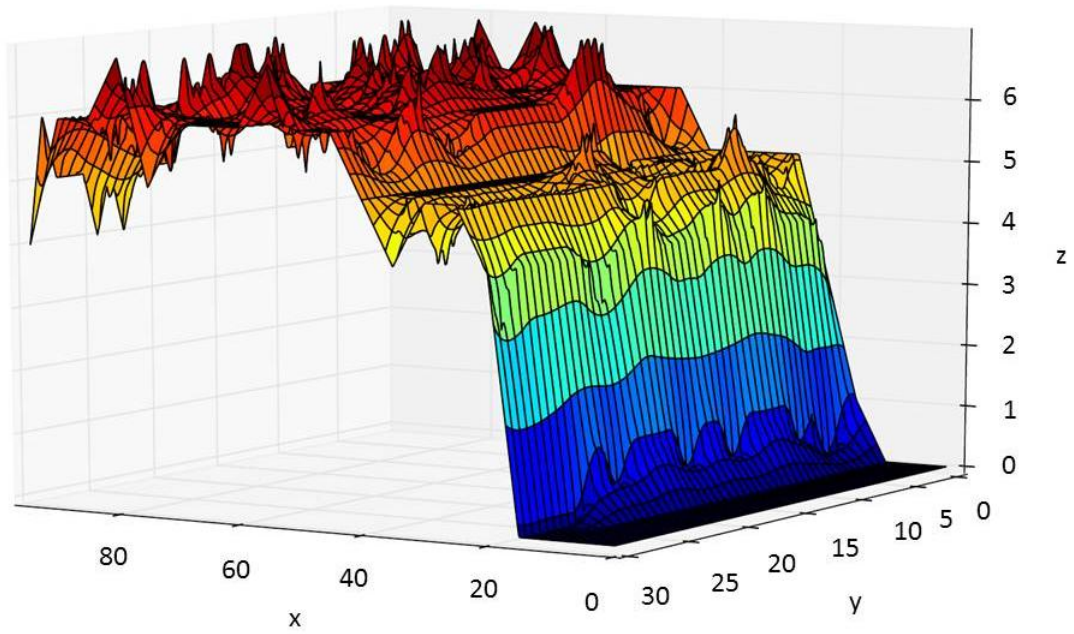


Figure 5.4: The number of fiat assertions to be explained in a bunch (x), the agenda size (y), and the number of ignored bunches (z).

4. **By successively explaining fiat assertions of a bunch, the scores increase, monotonically.**

   The main requirement of this work is that by explaining observations successively, the ranks of the interpretation alternatives increase, monotonically. In

this case, the agent's belief in its observations increases [GMN+10a, GMN+10c]. Additionally, we reduce the uncertainty of observations. In this work, we control abduction depth through the maximum number of fiat assertions m to be explained in a bunch of assertions. Note that each fiat assertion represents an observation.

In the next experiment, we test whether variable m has an influence on the scoring value of an interpretation alternative. Note that the input data arrives incrementally to the interpretation engine [HBB+13, GMN+10c]. In the following, we iterate the number of fiat assertions m to be explained in a bunch. In Figure 5.5, the scoring value of an interpretation alternative is a function of m and the required time for processing fiat assertions. The x-axis, y-axis, and z-axis indicate the required time for explaining fiat assertions without any delays and any idle times, m, and the scoring values, respectively. Additionally, vertical lines indicate the beginning of a new bunch. Since x-axis does not contain any delays and any idle times, Figure 5.5 depicts the compact representation of results. Figure 5.5 shows that by successively explaining fiat assertions of a bunch, the preference scores increase, monotonically. Thus, the main requirement of this work has been met.

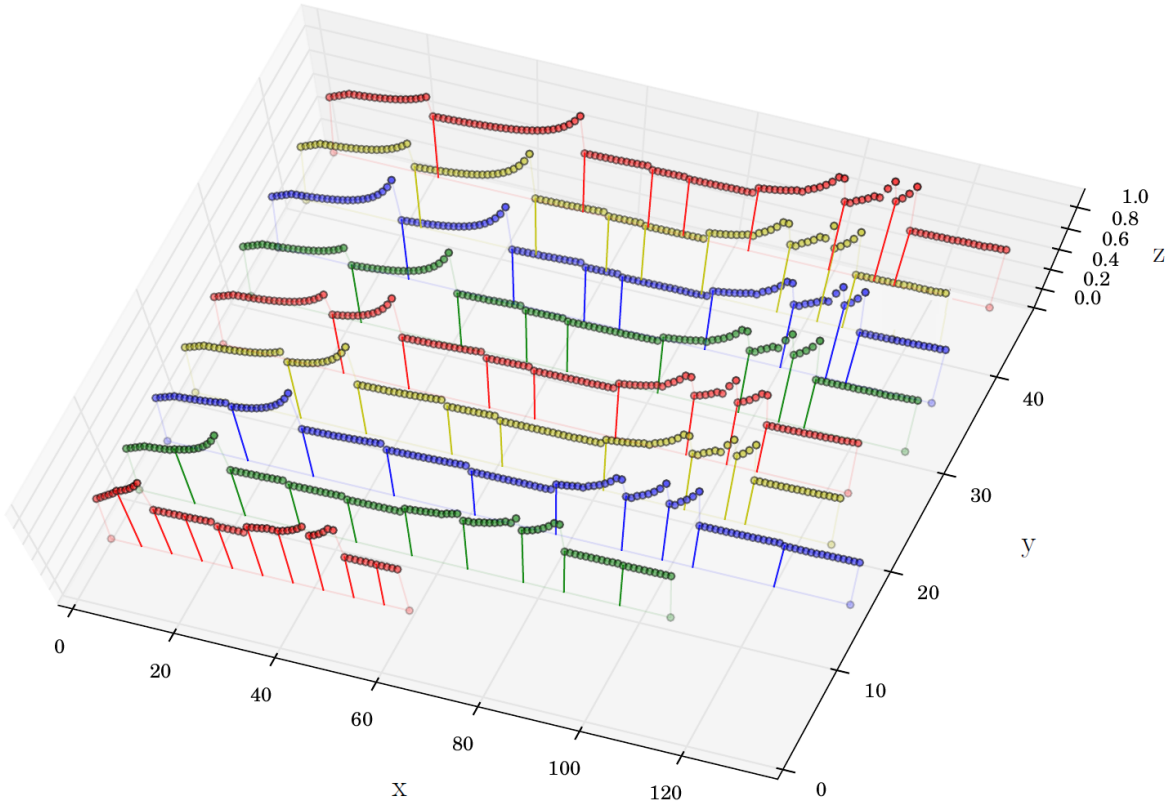


Figure 5.5: The required time for explaining fiat assertions without any delays and any idle times (x), the number of fiat assertions to be explained in a bunch (y), and the scoring value (z).

We consider two strategies when the current bunch is being processed and a new bunch arrives:

- stop-processing
- non-stop-processing

The first strategy that we consider is to stop explaining fiat assertions of the current bunch when the following bunch arrives. In this case, we switch to the following bunch and stop processing the current bunch. Thus, the remaining unexplained fiat assertions of the current bunch are deleted. Figure 5.6 depicts the results when the maximum number of fiat assertions to be explained in a bunch is $m = 28$. This means if a bunch contains less or equal 28 fiats, and the following bunch has not still arrived, the fiats of the bunch are explained completely and there might be an idle time for the arrival of the following bunch. Figure 5.6 shows that by explaining fiat assertions of a bunch, the scores increase, monotonically. Furthermore, we can see that the bunches do not arrive at equal

time distances. A longer bunch contains more fiat assertions than a short bunch. The red lines indicate the idle times.

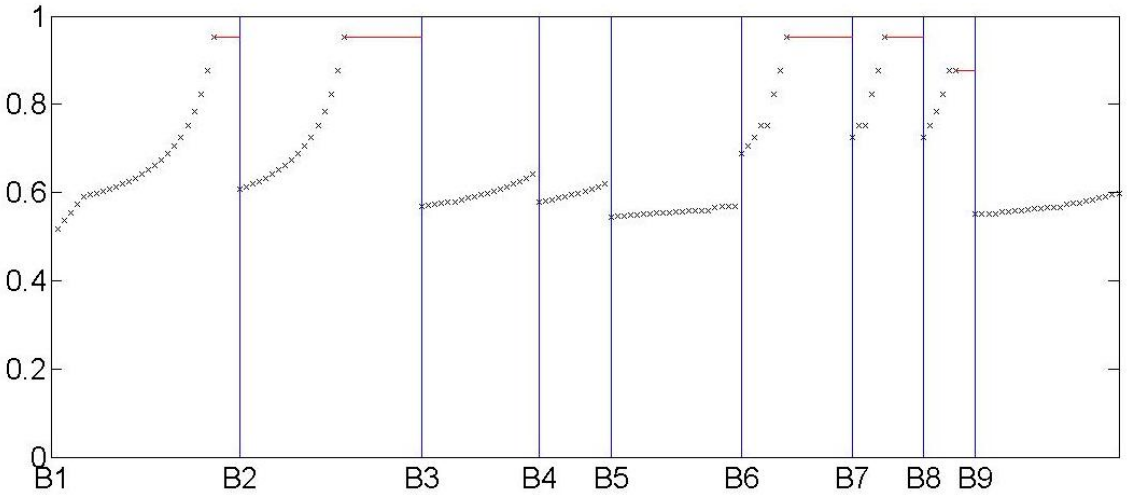


Figure 5.6: Strategy is stopProcessing=true, the time axis indicated with the arrival time of bunches (x), the scoring value of the interpretation Abox by explaining fiat assertions (y)

As a new bunch of assertions arrives, the final scores either decrease or increase:

- The final score decreases if there are more unexplained fiat assertions (with probability $p = 0.5$) in the following bunch than explained fiat assertions (with probability $p > 0.5$) in the previous bunch. Thus, the average is smaller than the final score in the previous bunch.
- The final score increases if in the previous bunch the number of unexplained fiat assertions (with probability $p = 0.5$) is much more than the number of explained fiat assertions (with probability $p > 0.5$).

In Figure 5.6, by the arrival of a new bunch, the scores decrease in most of the cases. Only by the arrival of B_6 the final score increases.

The advantage of stop-processing approach is that bunches of assertions are processed on time. Thus, there is no delay for processing the following bunches. Additionally, there are not any ignored bunches. The disadvantage of this approach is that all fiat assertions of a bunch might not be explained, completely. Thus, we might not have the complete interpretation results of each bunch.

The next strategy is to continue processing the current bunch although the following bunch has arrived. Figure 5.7 shows the results. In this figure, before

switching to the following bunch, all fiat assertions of a bunch are explained completely. The red lines indicate the idle times and the blue lines indicate the delays. There are three ignored bunches in the next figure.

The advantage of non-stop-processing approach is that all fiat assertions of each bunch are explained, completely. Thus, we have the complete interpretation results of each bunch. The disadvantage of this approach is that there are delays for processing the following bunches. Additionally, there are ignored bunches. Consequently, we lose the corresponding interpretation results. Thus, there is a tradeoff between the maximum number of fiat assertions to be explained in a bunch of assertions and the number of ignored bunches.

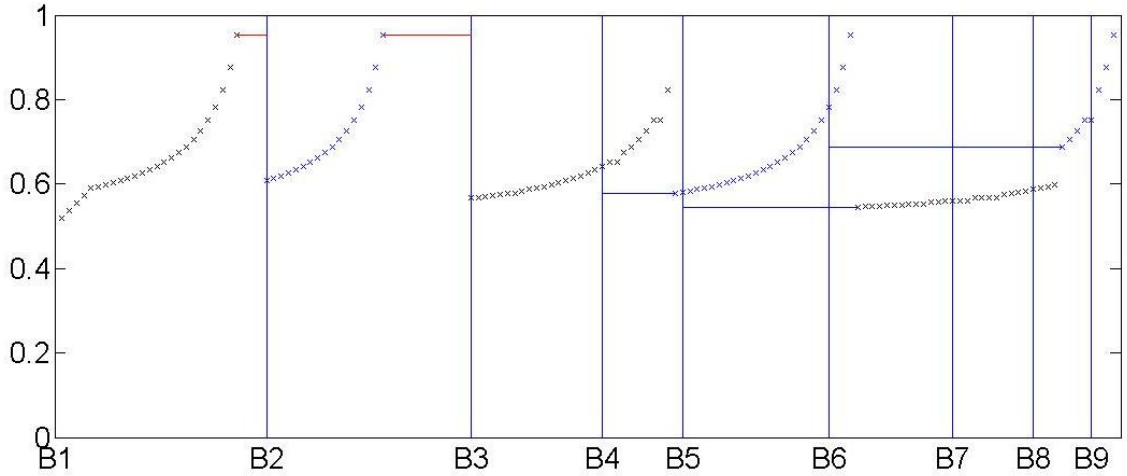


Figure 5.7: Strategy is stopProcessing=false, the time axis indicated with the arrival time of bunches (x), the scoring value of the interpretation Abox by explaining fiat assertions (y)

5.4 Quality of the Interpretation Results

In the previous chapter, we have presented an algorithm for the interpretation of the multimedia documents. In this section, we evaluate the quality of the interpretation results generated by the interpretation engine. Important measurements for the evaluation of the results are recall and precision. In accordance with Khoshafian and Baker's notation [KB96, page 358], we define recall and precision in the context of multimedia interpretation:

$$Recall = \frac{Number\ of\ Relevant\ Objects\ Returned}{Total\ Number\ of\ Relevant\ Objects\ in\ the\ Collection} \tag{5.3}$$

$$Precision = \frac{Number\ of\ Relevant\ Objects\ Returned}{Total\ Number\ of\ Objects\ Returned} \tag{5.4}$$

For the determination of the recall values, the total number of relevant objects in the video documents should be determined. For this purpose, the video documents should be annotated completely. Manual video annotation is an extremely time-consuming and cost-intensive task. Thus, in the following we determine only the precision values. Note that manual effort is also required for precision determination. But the manual effort for precision determination is less than the effort required for the recall determination. The reason is that for the determination of recall, the video document should be annotated completely. But for the precision determination, we check whether the generated high level annotations occur in the video document and then we count their numbers.

We consider 6 videos for which we have the surface-level analysis results. The videos have different topics. The surface-level analysis results of the videos are sent to the interpretation engine. Thus, for each video an interpretation Abox is generated. We determine the total number of the deep-level concept assertions generated by the interpretation engine. In order to determine the precision values of each video, a human expert checks manually whether each deep-level concept assertion of the interpretation Abox exists in the video. Then, the human expert counts the deep-level concept assertions existing in the interpretation Abox.

Table 5.2 depicts the precision values for the 6 videos. The columns of the next table depict the video topic, the total objects returned for each video, the number of relevant objects returned, and the precision values, respectively. The total number of objects returned are the deep-level objects generated during the Abox interpretation. Thus, the surface-level analysis results are not considered. A human expert checks whether the deep-level objects occur in the video document. In this case, their numbers are counted and given in the third column of the following table. The forth column gives the precision values which are determined by the formula $|Relevant\_Objects|/|Total\_objects|$.

Video topic	$\|Total\_objects\|$	$\|Relevant\_Objects\|$	Precision
Wind energy approval	67	64	0.96
Wind power station	45	43	0.96
Electricity generation	64	59	0.92
Economic espionage	56	47	0.83
Wind power park	62	54	0.87
Wood gas	63	60	0.95

Table 5.2: Precision values for videos with different topics

The reason for precision determination is to check whether the controlling Abox abduction approaches applied in this work generate correct results. It might be the case that the generated deep-level objects do not occur in the video documents at all. But the precision values given in Table 5.2 show that we produced acceptable results. Thus, the approaches applied in this work, namely controlling branching, controlling abduction depth, and controlling reactivity for the interpretation process, produce high-

quality interpretation results.

Chapter 6

Conclusions

Fully manual annotation of multimedia documents is a resource-consuming task. An objective of this work is to generate annotations, automatically. Thus, we speed up the annotation task and decrease the required resources. The generated annotations are attached to the multimedia documents and are used for the semantics-based retrieval process. In the following, we mention the objectives of this work:

1. To define an approach to deal with uncertain observations which are the input data to the interpretation engine.

2. To define a media interpretation agent to generate high-level explanations for the low-level observations.

3. To increase the expressivity of the knowledge representation language described in [Kay11] by considering recursive Horn rules.

4. To define a probabilistic scoring function according to the Markov logic formalism [DR07] for ranking interpretation alternatives.

5. To control abduction procedure in terms of branching and depth by applying Markov logic formalism.

6. To define an approach to incrementally process the input data to the probabilistic interpretation engine.

7. To increase the ranks of interpretation alternatives monotonically by successively explaining observations. This leads to reducing the uncertainty of observations and increasing the agent's belief in its observations. This point is the main requirement of this work.

In this work, we achieved the above mentioned objectives. In the following, we summarize this work, conclude the results, and present some topics for the future work.

In this work, the surface-level analysis results generated by state-of-the-art analysis tools are uncertain. Thus, the interpretation engine deals with uncertain input data. To handle the uncertainty in input data, we applied Maximum A Posteriori [DLK+08]. This approach determines the most probable world given the evidence. Additionally, we defined a media interpretation agent to generate high-level explanations for the surface-level analysis results which are observations of the agent.

Moreover, we selected Markov logic [DR07] as the probabilistic formalism. An objective of this work is to increase the expressivity of the knowledge representation language described in [Kay11] by supporting recursive Horn rules. Furthermore, we applied Markov logic formalism to define an additional knowledge base which is composed of a set of weighted rules. Thus, we increased the expressivity of the knowledge representation language. Additionally, we defined a probabilistic scoring function according to the Markov logic formalism. According to this knowledge base, we ranked interpretation alternatives, probabilistically.

Note that we applied Markov logic formalism only for ranking interpretation alternatives but not for generating interpretations. The reason is that according to the Markov logic abduction defined in [KM09], we require fixed names for individuals. Since during Abox abduction new individuals are generated, we cannot apply the approach defined in [KM09].

The scoring function defined in this work is more general than the one defined in [EKM09]. The reason is that our scoring function deals with uncertain observations and determines the rank of an interpretation alternative whereas the one defined in [EKM09] deals with strict observations and determines the score of an explanation. Furthermore, our scoring function considers the complete set of assertions (including observations) to determine the rank of an interpretation alternative whereas the one defined in [EKM09] considers for ranking process only the assertions which exist in the explanation. Moreover, our scoring function is probabilistic whereas the one in [EKM09] is non-probabilistic. The other important point is that the scoring function defined in [EKM09] considers a single abduction step. Thus, it is possible that the interpretation engine generates too many interpretation alternatives which have equal ranks.

The main goal of this work is to control Abox abduction procedure. The reason for controlling Abox abduction is that the agent's resources for computing explanations are limited. In this work, we showed that by applying Markov logic formalism [DR07], we can control Abox abduction procedure. For the abduction procedure, the agent selects observations one after the other for explanation. Generally, there are two different runtime problems in the Abox abduction, namely branching problem and depth problem.

To solve branching problem during Abox abduction, we defined controlling branching procedure. For this purpose, we applied Beam search to explore the interpretation space. At each step, only the Abox with the highest score is chosen to explain further observations. Consequently, we have branchings only for this Abox. According to this approach, we avoid branchings for Aboxes with lower scores.

Moreover, we introduced a procedure for controlling abduction depth. According to this procedure, we defined the maximum number of fiat assertions m to be explained in a bunch of assertions. By selecting a small value for m, we might lose some interpretation results. The reason is that all fiat assertions of a bunch are not explained, completely. By selecting a large value for m, there might be a delay to explain fiat assertions of the following bunches. Thus, the interpretation results of the following bunches might be generated later than required. Furthermore, by selecting a large value for m, we cannot control abduction depth anymore. Thus, the selection of variable m affects the interpretation results.

Controlling branching and controlling abduction depth reduce the space of possible interpretations of media content. Thus, we reduce time and memory space required for computation of the interpretation procedure.

In this work, each multimedia document is divided into several segments and each segment is analyzed, separately. Thus, the surface-level analysis results arrive incrementally to the interpretation engine [HBB+13, GMN+10c]. In other words, the interpretation engine deals with bunches of assertions where the processing of a bunch of assertion has not yet been finished completely while the following bunch arrives. In order to process the data incrementally, we defined controlling reactivity procedure. For this purpose, we introduced two different approaches, namely stop-processing and non-stop-processing. According to stop-processing, the interpretation engine stops processing the current bunch of assertions, and starts processing the newly arrived observations of the following bunch. According to non-stop-processing approach, the interpretation engine continues processing the current observations although a new bunch of assertions has already arrived.

Each approach has advantages and disadvantages. The disadvantage of stop-processing approach is that we might lose some interpretation results. The reason is that we might not explain observations of a bunch, completely. The advantage of this approach is that by quickly switching to the new observations of following bunches, bunches of assertions are processed on time. Thus, we might gain more important data. Moreover, there are no ignored bunches.

The disadvantage of non-stop-processing approach is that the processing of some bunches might be delayed. This happens when a bunch with many fiat assertions is followed by a bunch with few fiat assertions. Thus, with delay we gain the interpretation results of the second bunch. Furthermore, there might be some ignored bunches. The advantage of non-stop-processing approach is that all observations of a bunch are explained, completely. Thus, we do not lose any interpretation results.

The defined approaches for controlling Abox abduction namely, controlling branching, controlling abduction depth, and controlling reactivity can be applied for every other domain. Thus, these approaches are not domain specific.

Each interpretation alternative is composed of large sets of assertions. Thus, maintaining all interpretation alternatives is a resource-consuming task. In order to reduce the memory space required for the interpretation process, we did not keep all inter-

pretation alternatives. In this thesis, we used an agenda which maintained possible interpretation alternatives. On the agenda, the interpretation alternatives are sorted by scoring values. Each agenda has a certain size. An agenda with agenda size k maintains the top k interpretation alternatives. Thus, we reduce the memory space required for the interpretation process, enormously.

We compared our preference score with the one defined in [Poo91]. The preference scores are defined according to two different approaches. The preference score in [Poo91] is defined according to the Maximum Likelihood approach [RN03] whereas our preference score is determined according to the Maximum A Posteriori approach [DLK$^+$08]. Furthermore, the instances of hypotheses in [Poo91] are logically independent, which does not hold in our work. Our preference score is more general than the one defined in [Poo91]. The reason is that our preference score determines the rank of an interpretation alternative whereas the one defined in [Poo91] determines the rank of an explanation. Furthermore, our approach deals with uncertain observations whereas the approach defined in [Poo91] deals with strict observations.

In this thesis, we determined the probabilities according to the Markov logic formalism [DR07]. In order to increase the accuracy of a probability value, we increased the sampling size. The results showed that by increasing the sampling size, the processing time increases, linearly. We considered for this measurement the accumulated time to process all bunches of assertions according to MC-SAT algorithm [PD06]. Furthermore, by increasing the sampling size, the final score of an interpretation alternative converges to a limit. We applied two different algorithms namely, MC-SAT [PD06] und Gibbs sampling [GG84]. The results showed that MC-SAT algorithm converges faster than Gibbs sampling to limit.

In order to control abduction depth procedure, we defined maximum number of fiat assertions m to be explained in a bunch. The results showed that by increasing the number of fiat assertions to be explained in a bunch, the number of ignored bunches increases. Thus, there is a tradeoff between the maximum number of fiat assertions to be explained in a bunch and the number of bunches to be processed. Consequently, value m affects the interpretation results. Furthermore, the results showed that the agenda size does not have any influence on the number of ignored bunches.

During the interpretation procedure, the observations of an Abox are successively explained. The results showed that by successively explaining observations, the score of the interpretation Abox increases, monotonically [GMN$^+$10a, GMN$^+$10c]. Thus, the agent's belief in its observations increases. Consequently, the interpretation procedure reduces the uncertainty of observations. Thus, the main requirement of this work has been met.

We developed the semantic interpretation engine, a software system to automatically generate annotations for multimedia documents. Furthermore, we evaluated the probabilistic interpretation engine in a practical scenario. Finally, the quality of the interpretation results generated by the interpretation engine were evaluated in terms of precision [KB96]. The evaluation results showed that we produced high-quality inter-

pretation results although we applied approaches for controlling Abox abduction. Thus, the applied approaches for controlling Abox abduction did not lead to low-quality interpretation results. Consequently, the quality of the interpretation results is not affected by controlling Abox abduction.

In the following, we present some topics for the future work:

- Before analyzing a video document, the video document is divided into several segments. Then, the surface-level analysis results are associated to the segments. In this work, we did not consider for the interpretation procedure from which video segment the assertions are generated.

  As a future work, we can generate interpretations for each segment separately. Then, we consider the interpretations of all segments for the fusion procedure. For this purpose, we can apply a set of forward chaining rules which consider the chronologically appearance of the video segments. Accordingly, we can generate deep-level assertions.

- In the strategy Algorithm 3, we specified in which order the fiat assertions of an Abox should be explained so that the scoring values assigned to the interpretation Aboxes are plausible. According to Algorithm 3, we determined the supporting weighted rules of each fiat assertion. Then, we determined the weighted rule with the maximum weight. Thus, the next fiat assertion to be explained is the fiat assertion supported by a weighted rule with the highest weight. If we do not consider the weights of the supporting weighted rules, a small scoring value is assigned to an interpretation Abox which could have a high scoring value. A topic for the future work could be to determine the selection order of the fiat assertions for the explanation procedure and examine the results.

- In this work, we used an additional knowledge base which contained weighted rules. We assigned unequal positive weights to the weighted rules. However, the weights of the weighted rules can be learned [LD07, DLK$^+$08]. Learning the weights of the weighted rules could be another topic for future work.

- For controlling abduction depth, we defined maximum number of fiat assertions m to be explained in a bunch of assertions. In this work, we discussed the advantages and disadvantages of selecting a small/large value for m. We also discussed that the selection of variable m affects the interpretation results. A future work topic could be to select variable m, automatically.

Appendix A

Alchemy Knowledge Representation System and Language

Alchemy [KSR+10] is an open-source software package developed at the University of Washington. Alchemy supports Markov logics [DR07] as the underlying formalism. The system provides algorithms for probabilistic logic inference and statistical relational learning. Appendix A is partly taken from [GMN+09a].

A.1 Alchemy Knowledge Representation Language

The knowledge base in Alchemy is based on the following assumptions [DLK+08]:

- **The unique name assumption:** which means that two different names refer to two different objects in the universe.
- **The domain closure assumption:** which means that there are no other objects in the universe than those specified by the constants of the knowledge base.

Based on the domain closure assumption, existentially quantified formulas are replaced by disjunctions of groundings of the quantified formula with constants used in the knowledge base. Universal quantifiers are replaced by the conjunction of the corresponding groundings as well.

Example 12 An example for the domain closure assumption is helpful to clarify the consequences. Let us assume the following formula and two constants $Mary$ and $Sara$:

$$F = \exists x, y \, MotherOf(x, y) \tag{A.1}$$

Applying the domain closure assumption means that the formula F can be replaced by:

$$MotherOf(Mary, Mary) \vee MotherOf(Mary, Sara) \vee$$
$$MotherOf(Sara, Mary) \vee MotherOf(Sara, Sara)$$

For answering probability queries, the number of true groundings needs to be determined (see Section 2.2.3). Thus, answering probability queries corresponds to model-checking first-order formulas. The problem of answering probability queries is PSPACE-complete [Var82]. The same holds for the entailment problem for a probability assertion. The next figure depicts the main units in the Alchemy module:

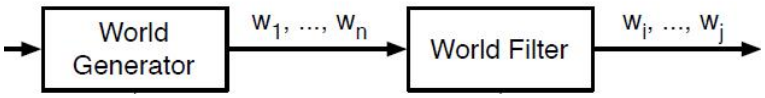


Figure A.1: Alchemy module

The Alchemy module is composed of the following units:

- World Generator: Based on the Tbox and the input Abox this component produces all Markov logic worlds, which are indicated in Figure A.1 by $w_1, w_2, \ldots, w_n$. Note that a world is a vector of all ground atoms of the domain. Let us assume there are m ground atoms. Consequently, the number of worlds is 2^m. The generated worlds are the input to the next component called world filter.

- World Filter: This component removes the impossible worlds. In this step, the subsumption axioms, domain and range restrictions, and the evidences in the evidence file `.db` are considered for the world elimination process. The remaining worlds indicated in the above figure by $w_i, \ldots, w_j$ are known as possible worlds.

A knowledge base consists of three parts namely *types*, *predicates* and *formulas* [KSR+10, DLK+08]. The first two parts are required whereas the last part is optional.

1. Types: In the first part, types are defined where a set of constants is assigned to each type, e.g., $city = \{Hamburg, Berlin\}$ indicates a type *city* with two constants *Hamburg* and *Berlin*. Each defined type must have at least one constant. In Alchemy, a constant can have different types.

2. Predicates: In the second part, the applied predicates and their corresponding types are introduced, e.g., *AirPollution(city)* defines a predicate *AirPollution* with type *city*. *AirPollution(city)* means that the type of the individuals for the predicate $AirPollution$ is *city*. By considering $city = \{Hamburg, Berlin\}$, the only options are:

$$AirPollution(Hamburg)$$
$$AirPollution(Berlin)$$

   The advantage of using types is to speed up the inference process since the world generator of Alchemy produces only the worlds which correspond to the correct typings.

3. Formulas: In the last part of the knowledge base, the so-called hard- and soft formulas (in first-order logic) are defined. Hard formulas, also known as strict formulas, are the formulas which cannot be violated and are valid for every object of the domain. In order to distinguish hard and soft formulas of the knowledge base, hard formulas are terminated by a period, and soft formulas are preceded by a weight. Based on the Markov logic theory [DLK+08], the weight of a hard formula is infinity, but the Alchemy engine [KSR+10] assigns a "high" weight (which is unequal to infinity) to the hard formulas internally. Later, we will discuss how Alchemy determines the weight of a hard formula. Let us assume two predicates *CityWithIndustry(city)* and *CityWithAirPollution(city)* (see item 2. above). In the following, an example for a hard formula is given:

$$CityWithIndustry(x) \Rightarrow CityWithAirPollution(x).$$

Due to the typing constraints, the variable x can only be substituted by individuals of type *city*. Let us consider the predicates *CityWithRain(city)* and *CityWithFlood(city)* (see item 2. above). The next example shows a soft formula:

$$0.1 \;\; CityWithRain(x) \Rightarrow CityWithFlood(x)$$

A weight is assigned to the above formula and consequently, makes it a soft formula. Weighted ground atoms, e.g., 0.6 $CityWithRain(Berlin)$, are considered as formulas and stand in the formula part of .mln file. Unlike weighted ground atoms, the strict ground atoms, e.g. $CityWithRain(Berlin)$, stand as evidences in the evidence file .db. Note that the evidence file contains only strict ground atoms. The weights of formulas can be learned [LD07, DLK+08] using the weight learning tool of Alchemy [KSR+10].

A.2 Interfaces to Alchemy

The command for solving inference problems in Alchemy, e.g., for determining the probabilities based on the given evidences is *infer*, and it has the following form [KSR+10]:

```
infer -i uniform.mln -r uniform.result -e evidence.db -q QueryFormula
```

The options indicate:

- -i: input file "uniform.mln"
- -r: output file "uniform.result"
- -e: evidence file "evidence.db"

- -q: query formula which can be a concept name, a role name, a ground atom or a conjunction of ground atoms. We can specify more than one query formula separated by comma.

In the following, the above mentioned file types for performing inference are introduced:

- Markov logic network file (with extension .mln): The MLN file contains types, predicates, and formulas as introduced above.

- An optional evidence file (with extension .db): The .db file contains evidence ground atoms which can be either true or false. By default, evidence predicates are treated using the closed-world assumption, meaning that if they are not present in the .db file, they are assumed to be false. Non-evidence predicates are treated using the open-world assumption by default (a predicate is called an evidence predicate if at least one grounding of a predicate exists in the evidence file.). The evidence file can be empty, since it is optional. It is also possible to use multiple evidence files.

- Output file (with extension .result): In case of performing probabilistic inference, this file contains the probabilities of query atoms given the evidence file. In case of MAP inference, this file shows the most likely state of query atoms.

A.3 Inference Services

Alchemy [KSR+10] can solve the problem of answering probability queries as well as the entailment problem for probability assertions using exact inference as described in 2.2.3. For exact inference, runtimes usually turn out to be too long in practice. Therefore, Alchemy can be instructed to perform approximate inference based on sampling techniques. The exactness of approximate inference can be controlled. In order to produce more accurate results, the maximum number of steps to run sampling algorithms can be increased (option -maxSteps). By increasing the number of samples, the results of approximate inference converge to the results of exact inference.

Some effects of the approximation techniques used by Alchemy have to be understood, however. We have seen that strict formulas can reduce the number of worlds that have non-zero probability. Based on the theory of Markov logics [DLK+08], the weight of a strict formula is positive infinity. According to the manual [KSR+10], Alchemy assigns (positive) finite weights to strict formulas, however. To determine these weights for strict formulas, Alchemy converts input formulas into conjunctive normal form (CNF). Afterwards, the weight of a formula is divided equally among its CNF clauses.

The weight assigned to a strict formula depends on the inference type. Alchemy performs two types of inference, namely [KSR+10]:

- Probability queries: Probabilistic inference methods currently implemented in Alchemy are based on two general algorithms: Markov Chain Monte Carlo (MCMC) [GRS96] and (lifted) belief propagation (option -bp) [YFW01, SD08]. Based on MCMC different inference algorithms have been implemented in Alchemy: Gibbs sampling (option -p) [GG84], simulated tempering (option -simtp) [MP92], and MC-SAT (option -ms) [PD06]. The default algorithm is lifted belief propagation [SD08]. The advantage of lifted inference in comparison to the fully grounded network is the runtime and memory usage [KSR+10]. For the above-mentioned inference methods, the number of iterations can be specified. In the following, the number of iterations is set to 1000:

  infer -i uniform.mln -r uniform.result -e evidence.db -q Car,DoorSlam
  -maxSteps 1000

  The query formula contains two predicates Car and $DoorSlam$. Let us assume $Car(t_1)$ and $DoorSlam(t_2)$ where:

  $$\begin{aligned} t_1 &= \{C_1, C_2\} \\ t_2 &= \{DS_1, DS_2\} \end{aligned}$$

  As a result, by the above command the next four conditional probabilities are determined:

  $$\begin{aligned} &P(Car(C_1) = true \mid \vec{e}) \\ &P(Car(C_2) = true \mid \vec{e}) \\ &P(DoorSlam(DS_1) = true \mid \vec{e}) \\ &P(DoorSlam(DS_2) = true \mid \vec{e}) \end{aligned}$$

  where $\vec{e}$ indicates the evidence vector. The default weight assigned to the clauses of a strict formula based on MCMC inference [GRS96] is twice the maximum weight mentioned in the complete set of weighted input formulas [KSR+10].

- MAP inference: This type of inference is called Maximum A Posteriori (MAP) (option -a) [DLK+08] and computes the most-likely state of query atoms given the evidence [SD05]. In other words, the output consists of ground atoms associated with zeros and ones (denoting a world). Note that the output file contains all ground atoms of the predicates mentioned in the QueryFormula of the infer command. The default weight assigned to the clauses of a strict formula based on MAP inference is the sum of weights appearing in the set of weighted input formulas plus 10 [KSR+10]. The command which solves the MAP problem in Alchemy has the next form [KSR+10]:

  infer -a -i uniform.mln -r uniform.result -e evidence.db -q QueryFormula

The query formula contains predicate names separated by coma. The following command determines the most probable world considering two predicate names Car and $DoorSlam$:

```
infer -a -i uniform.mln -r uniform.result -e evidence.db -q Car, DoorSlam
```

Example 13 In this section, an example for a MLN file is given.

$city = \{Hamburg, Berlin\}$
$CityWithIndustry\,(city)$
$CityWithAirPollution\,(city)$
$CityWithRain\,(city)$
$CityWithFlood\,(city)$
$CityWithIndustry(x) \Rightarrow CityWithAirPollution(x).$
$0.1\ CityWithRain(x) \Rightarrow CityWithFlood(x)$
$0.3\ CityWithIndustry\,(Hamburg)$

Table A.1: Example for a MLN file

Additionally, an example for a DB file based on the above MLN file is given:

$CityWithRain\,(Hamburg)$
$CityWithAirPollution\,(Berlin)$
$CityWithFlood\,(Berlin)$

Table A.2: Example for a DB file

A query formula for this example could be:

$$CityWithAirPollution(Hamburg) \wedge CityWithFlood(Hamburg) \tag{A.2}$$

A.4 Peculiarities of the Inference Engine Alchemy

In this section, we discuss the problems we faced with as we applied the inference engine Alchemy [KSR+10]. In the Markov logic formalism [DLK+08], it is essential to define the type of each predicate. Note that a type is a non-empty set containing constants. If a type does not have any constant, Alchemy engine generates error. To avoid this type of error, constants are assigned to each predicate type. Note that a constant can have several types. In the following examples, we discuss the important issue of typing.

Example 14 In this example, we define a subsumption Tbox axiom containing two predicates. Additionally, we define the type of each predicate where the types are not the same. Moreover, the super concept type is an empty set:

$$politician = \{P_1\}$$

$$Politician\,(politician)$$
$$Person\,(person)$$

$$Politician(x) \Rightarrow Person(x).$$

$$Politician(P_1).$$

Table A.3: An example for *.mln* file

For the above example, Alchemy generates the following error:

"Type *person* has no constant. A type must have at least one constant."

As it can be seen, no relation is defined among the types in the Alchemy engine. At least, a relation could have been defined among the predicate types of the strict subsumption axioms. In this example, the following relation must hold:

$$politician \subseteq person \tag{A.3}$$

The *person* type must have at least the constant P_1. A solution to this problem is to define a non-empty set type for *person*. In the following, we define three suggestions:

$$\begin{aligned} person &= \{P_1\} \\ person &= \{Ind_{42}\} \\ person &= \{Ind_{42}, P_1\} \end{aligned}$$

The next query can be asked even though $P_1 \notin person$:

$$P(Person(P_1) = true) = 1 \tag{A.4}$$

Example 15 In this example, we have the same subsumption Tbox axiom as in the previous example and the same predicate types. Unlike the previous example, the subconcept type is an empty set.

$person = \{P_1\}$

$Politician\,(politician)$
$Person\,(person)$

$Politician(x) \Rightarrow Person(x).$

Table A.4: An example for *.mln* file

The Alchemy engine generates the next error:

"Type *politician* has no constant. A type must have at least one constant."

Since P_1 could be a Politician, the constant P_1 could have been automatically added to the type *politician*. In the following, we define some *politician* types to avoid the error:

$$\begin{aligned} politician &= \{P_1\} \\ politician &= \{Ind_{42}\} \\ politician &= \{P_1, Ind_{42}\} \end{aligned}$$

We can ask the next query even though $P_1 \notin politician$:

$$P(Politician(P_1) = true) = 0.5$$

This example shows that the relation among predicate types is not defined for strict subsumption Tbox axioms.

Example 16 This example shows that the Alchemy engine [KSR+10] does not produce any errors although the input is inconsistent. This means, during the syntax controlling, the consistency checking according to the given axioms is not performed. In this example, a disjoint axiom is given. Moreover, types are defined for the predicates where the types contain the same constant. According to Alchemy, a constant can have several types. Thus, it does not lead to any error generation.

$interview = \{P\}$
$conference = \{P\}$

$Interview\,(interview)$
$Conference\,(conference)$

$Interview(x) \Rightarrow !Conference(x).$

$Interview(P).$
$Conference(P).$

Table A.5: An example for *.mln* file

The above input is inconsistent since $Interview(P)$ and $Conference(P)$ are true even though the predicates *Interview* and *Conference* are disjoint. The Alchemy engine does not produce any error although the above input is inconsistent. But the answers to the following queries show that Alchemy performs conflict resolution before answering the queries:

$$P(Interview(P) = true) = 1$$
$$P(Conferece(P) = true) = 0$$

The above answers show that Alchemy randomly selects one of the assertions as true and the other one as false. Similarly, the most probable world for a conflicting input is correctly determined.

Example 17 In this example, we discuss the difference between the Description logic [BCM$^+$03] and the weighted logic according to the Markov logic formalism [DR07]. In the following, we define a subsumption Tbox axiom with two predicates. Moreover, the corresponding non-empty types are defined.

$person = \{P_1, P_2\}$
$politician = \{P_3\}$

$Person\,(person)$
$Politician\,(politician)$

$Politician(x) \Rightarrow Person(x).$

$Politician(P_3).$
$Person(P_1).$

Table A.6: An example for *.mln* file

The unique name assumption holds in the Description logic [BCM+03] and the Markov logic formalism [DLK+08]. Thus it follows, the constants P_1, P_2 and P_3 refer to different objects:

$$\begin{aligned} P_1 &\neq P_2 \\ P_1 &\neq P_3 \\ P_2 &\neq P_3 \end{aligned}$$

According to the Description logic, there are two $Person$ instances and only one $Politician$ instance. Since $Politician \sqsubseteq Person$, it follows:

$$P_1 = P_3 \text{ or } P_2 = P_3$$

But according to the unique name assumption, they do not follow. Thus, the subsumption formula $Politician \sqsubseteq Person$ should be wrong. But based on the Markov logic formalism, P_1 and P_2 might be Politician instances. Similarly, P_3 might be a $Person$ instance. Thus, we can calculate the next probabilities:

$$\begin{aligned} P(Politician(P_1) = true) &= 0.51 \\ P(Person(P_3) = true) &= 1 \\ P(Politician(P_2) = true) &= 0.32 \\ P(Person(P_2) = true) &= 0.64 \end{aligned}$$

A solution to this problem is to propagate the constants upwards and downwards according to the subsumption axioms.

$$\begin{aligned} person &= \{P_1, P_2, P_3\} \\ politician &= \{P_1, P_2, P_3\} \end{aligned}$$

Example 18 According to the Description logics [BCM+03], extending the knowledge base with the implicit knowledge does not change the knowledge base at all. In this example, we show that unlike the Description logics, adding an implicit formula to the knowledge base changes the knowledge base and consequently the probabilities. The only formula which can be added to the knowledge base and does not change the knowledge base is a formula with the weight zero. Consider the next *.mln* file:

$person = \{P_1\}$
$politician = \{P_1\}$

$Person\,(person)$
$Politician\,(politician)$

$1\ Politician(x)$
$1\ Politician(x) \Rightarrow Person(x)$

Table A.7: An example for *.mln* file

In the above knowledge base there is an implicit formula, namely:

$$w\ Person(x) \tag{A.5}$$

In order to determine the weight w, we ask the next probability:

$$P(Person(P_1) = true) = 0.65$$

Now we can determine w:

$$w = \ln \frac{p}{1-p} = 0.62$$

By extending the knowledge base with A.5 and asking the previous probability, we notice that the probability value is not the same as before:

$$P(Person(P_1) = true) = 0.79$$

This problem arises since the assertion and the axiom in the above knowledge base are not strict. Considering the above knowledge base with strict axiom and strict assertion, it holds:

$$P(Person(P_1) = true) = 1$$

By adding $Person(P_1)$ to the knowledge base, we have the situation like in Description logics where the integration of the implicit knowledge does not change the knowledge base.

Appendix B

RacerPro and its Module for DLLP Abduction

The interface for DLLP Abduction is part of the RacerPro description logic inference system. In order to explain a single concept or role assertion, we use in RacerPro [HM01]

retrieve-with-explanation()

which is called as follows:

retrieve-with-explanation()(i j roleName)(:Ω)

In the above notation, i and j indicate individual names. Furthermore, the strategy parameter Ω denotes strategies for variable instantiation. Two strategies are defined namely 'reuse existing individuals' and 'use new individuals'. We apply the strategy 'reuse existing individuals' with Ω =reuse-old if we want to use the individuals mentioned in the Abox. If no strategy parameter is mentioned, the applied strategy is 'use new individuals' where new individuals are hypothesized.

RacerPro can be obtained from the website: www.racer-systems.com.

Appendix C

The DLLP-Abduction Based Interpretation System

In order to demonstrate that the results obtained in this thesis are reproducible, the DLLP-abduction based interpretation module is available as open-source software. The data generator and the log file producer have been implemented by Maurice Rosenfeld in Java language. The data flow in CASAM system [HBB+13] is composed of AssertionSender and InterpretationReceiver which run on RMI server with the IP address and port number 134.28.70.136 : 9092. The following figure depicts the data flow [Has11]:

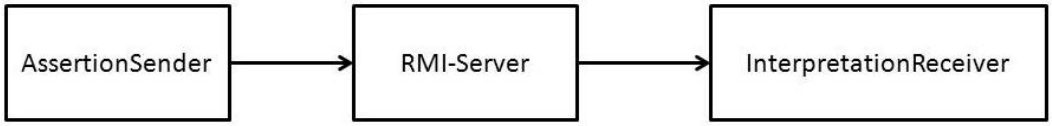


Figure C.1: Data flow in the CASAM system

The RMI test framework including the log file interpreter have been implemented by Björn Hass as part of his Master thesis [Has11] in Python language. The AssertionSender is a real time data source [HBB+13]. Thus, the input data is sent in real time and incrementally to the RMI server [HBB+13, GMN+10c]. Thus, the RMI server incrementally processes the data. The log files are parsed for the relevant data by applying regular expressions.

In order to test the framework, we use command line options. The following command line parses the log file log7.txt and sends the result to the output file 7.rmi [Has11]:

```
Python testscript.py --scan log7.txt 7.rmi
```

In order to plot the output file 7.rmi, we use the following command line [Has11]:

```
Python testscript.py --plot 6 7.rmi --show --what "Intp; r; igBunch"
```

where we use plot number 6. The parameters "Intp", "r", and "igBunch" indicate the number of interpretations, the maximum number of fiat assertions to be explained in a bunch of assertions and the number of ignored bunches, respectively.

Bibliography

[BCM+03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, Cambridge, NY, USA, January 2003.

[BHS05] F. Baader, I. Horrocks, and U. Sattler. Description Logics as Ontology Languages for the Semantic Web. In Dieter Hutter and Werner Stephan, editors, *Mechanizing Mathematical Reasoning: Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday*, volume 2605 of *Lecture Notes in Artificial Intelligence*, pages 228–248. Springer, 2005.

[BKN11] Wilfried Bohlken, Patrick Koopmann, and Bernd Neumann. SCENIOR: Ontology-Based Interpretation of Aircraft Service Activities. Technical report, University of Hamburg, Department of Informatics, Cognitive Systems Laboratory, February 2011.

[BNHK11] Wilfried Bohlken, Bernd Neumann, Lothar Hotz, and Patrick Koopmann. Ontology-Based Realtime Activity Monitoring Using Beam Search. In *Proceedings of ICVS 2011*, volume 6962 of *Lecture Notes in Computer Science*, pages 112–121. Springer, 2011.

[CDD+03] Simona Colucci, Tommaso Di Noia, Eugenio Di Sciascio, Francesco M. Donini, and Marina Mongiello. Concept Abduction and Contradiction in Description Logics. In Diego Calvanese, Giuseppe De Giacomo, and Enrico Franconi, editors, *Proceedings of the 16th International Workshop on Description Logics (DL2003)*, volume 81 of *CEUR Workshop Proceedings*, Rome, Italy, September 2003. CEUR-WS.org.

[CEF+08] Silvana Castano, Sofia Espinosa, Alfio Ferrara, Vangelis Karkaletsis, Atila Kaya, Ralf Möller, Stefano Montanelli, Georgios Petasis, and Michael Wessel. Multimedia Interpretation for Dynamic Ontology Evolution. In *Journal of Logic and Computation*, volume 19 of *5*, pages 859–897. Oxford University Press, 2008.

[Coo90] G. F. Cooper. The Computational Complexity of Probabilistic Inference using Bayesian Belief Networks. *Artificial Intelligence*, 42:393–405, 1990.

[CY90] James Joseph Clark and Alan L. Yuille. *Data Fusion for Sensory Information Processing Systems*, volume 105 of *The Springer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, Norwell, MA, USA, 1990.

[DK02] Marc Denecker and Antonis C. Kakas. Abduction in Logic Programming. In Antonis C. Kakas and Fariba Sadri, editors, *Computational Logic: Logic Programming and Beyond*, volume 2407 of *Lecture Notes in Computer Science*, chapter 16, pages 402–436. Springer, 2002.

[DLK+08] Pedro Domingos, Daniel Lowd, Stanley Kok, Hoifung Poon, Matthew Richardson, and Parag Singla. Just Add Weights: Markov Logic for the Semantic Web. In Paulo Cesar G. da Costa, Claudia d'Amato, Nicola Fanizzi, Kathryn B. Laskey, Kenneth J. Laskey, Thomas Lukasiewicz, Matthias Nickles, and Michael Pool, editors, *Uncertainty Reasoning for the Semantic Web I*, volume 5327 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 2008.

[DR07] Pedro Domingos and Matthew Richardson. Markov Logic: A Unifying Framework for Statistical Relational Learning. In L. Getoor and B. Taskar, editors, *Introduction to Statistical Relational Learning*, pages 339–371. Cambridge, MA: MIT Press, 2007.

[EKM09] Sofia Espinosa, Atila Kaya, and Ralf Möller. Formalizing Multimedia Interpretation based on Abduction over Description Logic Aboxes. In Bernardo Cuena-Grau, Ian Horrocks, and Boris Motik, editors, *Proceedings of the 22nd International Workshop on Description Logics (DL2009)*, Oxford, UK, July 2009.

[EKM11] Sofia Espinosa, Atila Kaya, and Ralf Möller. Logical Formalization of Multimedia Interpretation. In G. Paliouras, C. D. Spyropoulos, and G. Tsatsaronis, editors, *Knowledge-Driven Multimedia Information Extraction and Ontology Evolution*, volume 6050 of *Lecture Notes in Computer Science*, pages 110–133. Springer, 2011.

[Esp11] Sofia Espinosa. *Content Management and Knowledge Management: Two Faces of Ontology-Based Deep-Level Interpretation of Text*. PhD thesis, Hamburg University of Technology (TUHH), Hamburg, Germany, 2011.

[FK00] Peter A. Flach and Antonis C. Kakas. Abductive and Inductive Reasoning: Background and Issues. In Peter A. Flach and Antonis C. Kakas, editors,

Abduction and Induction: Essays on Their Relation and Integration , pages 1–27. Kluwer Academic Publishers, 2000.

[GG84] S. Geman and D. Geman. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

[GM10] Oliver Gries and Ralf Möller. Gibbs Sampling in Probabilistic Description Logics with Deterministic Dependencies. In Thomas Lukasiewicz, Rafael Penaloza, and Anni-Yasmin Turhan, editors, *Proceedings of the First International Workshop on Uncertainty in Description Logics (UnIDL-2010)*, Edinburgh, UK, 2010.

[GMN+08] Oliver Gries, Ralf Möller, Anahita Nafissi, Kamil Sokolski, and Maurice Rosenfeld. Formalisms Supporting First-order Probabilistic Structures. CASAM Project Deliverable D3.1, October 2008.

[GMN+09a] Oliver Gries, Ralf Möller, Anahita Nafissi, Maurice Rosenfeld, Kamil Sokolski, and Michael Wessel. Basic Reasoning Engine: Report on Optimization Techniques for First-Order Probabilistic Reasoning. CASAM Project Deliverable D3.2, September 2009.

[GMN+09b] Oliver Gries, Ralf Möller, Anahita Nafissi, Kamil Sokolski, and Maurice Rosenfeld. CASAM Domain Ontology. CASAM Project Deliverable D6.2, April 2009.

[GMN+10a] Oliver Gries, Ralf Möller, Anahita Nafissi, Maurice Rosenfeld, Kamil Sokolski, and Michael Wessel. A Probabilistic Abduction Engine for Media Interpretation based on Ontologies. In Pascal Hitzler and Thomas Lukasiewicz, editors, *Proceedings of 4th International Conference on Web Reasoning and Rule Systems (RR-2010)*, volume 6333 of *Lecture Notes in Computer Science*, pages 182–194, Bressanone/Brixen, Italy, September 2010. Springer.

[GMN+10b] Oliver Gries, Ralf Möller, Anahita Nafissi, Maurice Rosenfeld, Kamil Sokolski, and Michael Wessel. Meta-Level Reasoning Engine, Report on Meta-Level Reasoning for Disambiguation and Preference Elicitation. CASAM Project Deliverable D3.4, October 2010.

[GMN+10c] Oliver Gries, Ralf Möller, Anahita Nafissi, Maurice Rosenfeld, Kamil Sokolski, and Michael Wessel. Probabilistic Abduction Engine: Report on Algorithms and the Optimization Techniques used in the Implementation. CASAM Project Deliverable D3.3, May 2010.

[GMN+11] Oliver Gries, Ralf Möller, Anahita Nafissi, Maurice Rosenfeld, Kamil Sokolski, and Sebastian Wandelt. Dealing Efficiently with Ontology-Enhanced Linked Data for Multimedia. Technical report, Institute for Software Systems (STS), Hamburg University of Technology, Germany, 2011.

[GRS96] Walter R. Gilks, Sylvia Richardson, and D. J. Spiegelhalter, editors. *Markov Chain Monte Carlo in Practice*. Chapman and Hall, 1996.

[Häh01] Reiner Hähnle. Tableaux and Related Methods. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, chapter 3, pages 101–178. Elsevier Science Publishers B.V., 2001.

[Has11] Björn Hass. Evaluation of Media Interpretation Algorithms. Master's thesis, Hamburg University of Technology (TUHH), Hamburg, Germany, October 2011.

[HBB+13] Robert J. Hendley, Russell Beale, Chris P. Bowers, Christos Georgousopoulos, Charalampos Vassiliou, Petridis Sergios, Ralf Möller, Eric Karstens, and Dimitris Spiliotopoulos. CASAM: Collaborative Human-Machine Annotation of Multimedia. In *Multimedia Tools and Applications Journal (MTAP)*, 2013.

[HM01] Volker Haarslev and Ralf Möller. RACER System Description. In *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR 2001)*, volume 2083 of Lecture Notes in Computer Science, pages 701–705. Springer, 2001.

[HMS04] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reducing SHIQ-Description Logic to Disjunctive Datalog Programs. In *Proceedings of the 9th International Conference on the Principles of Knowledge Representation and Reasoning (KR 2004)*, pages 152–162, 2004.

[Kay11] Atila Kaya. *A Logic-Based Approach to Multimedia Interpretation*. PhD thesis, Hamburg University of Technology (TUHH), Hamburg, Germany, 2011.

[KB96] Setrag Khoshafian and A. Brad Baker. Multimedia and Imaging Databases. *Morgan Kaufmann Publisher*, 1996.

[KES11] Szymon Klarman, Ulle Endriss, and Stefan Schlobach. ABox Abduction in the Description Logic *ALC*. *Journal of Automated Reasoning*, 46(1):43–80, 2011.

[KM09] Rohit J. Kate and Raymond J. Mooney. Probabilistic Abduction using Markov Logic Networks. In *Proceedings of the IJCAI-09 Workshop on Plan, Activity, and Intent Recognition (PAIR-09)*, Pasadena, CA, July 2009.

[Kna11] Ulrich Knauer. *Algebraic Graph Theory: Morphisms, Monoids, and Matrices*. De Gruyter Studies in Mathematics. Walter de Gruyter, 2011.

[Kol50] Andrey N. Kolmogorov. *Foundations of the Theory of Probability*. Chelsea Publishing Company, 1950.

[Kow11] Robert Kowalski. *Computational Logic and Human Thinking: How to be Artificially Intelligent*. Cambridge University Press, 2011.

[KP97] Daphne Koller and Avi Pfeffer. Object-Oriented Bayesian Networks. In Dan Geiger and Prakash P. Shenoy, editors, *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-97)*, pages 302–313, Brown University, Providence, Rhode Island, USA, August 1997. Morgan Kaufmann.

[KSJ97] Henry Kautz, Bart Selman, and Yueyen Jiang. A General Stochastic Approach to Solving Problems with Hard and Soft Constraints. In D. Gu, J. Du, and P. Pardalos, editors, *The Satisfiability Problem: Theory and Applications*, pages 573–586. American Mathematical Society, New York, NY, 1997.

[KSR+10] Stanley Kok, Parag Singla, Matthew Richardson, Pedro Domingos, Marc Sumner, Hoifung Poon, Daniel Lowd, Jue Wang, and Aniruddh Nath. The Alchemy System for Statistical Relational AI: User Manual. Department of Computer Science and Engineering, University of Washington, Seattle, WA, January 2010. http://alchemy.cs.washington.edu/user-manual/.

[Lac98] Nicolas Lachiche. Abduction and Induction from a Non-Monotonic Reasoning Perspective. In *In Abduction and Induction: Essays on their Relation and Integration*, pages 107–116. Kluwer Academic Publishers, 1998.

[Las08] Kathryn Blackmond Laskey. MEBN: A Language for First-Order Bayesian Knowledge Bases. *Artificial Intelligence*, 172(2–3):140–178, February 2008.

[LD07] Daniel Lowd and Pedro Domingos. Efficient Weight Learning for Markov Logic Networks. In Joost N. Kok, Jacek Koronacki, Ramon Lopez de Mantaras, Stan Matwin, Dunja Mladenic, and Andrzej Skowron, editors, *Proceedings of the Eleventh European Conference on Principles and Practice of Knowledge Discovery in Databases*, volume 4702 of *Lecture Notes in Computer Science*, pages 200–211. Springer, 2007.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming*. Symbolic Computation Series. Springer, Berlin, Germany, Second edition, 1987.

[Lov78] Donald W. Loveland. *Automated Theorem Proving: A Logical Basis*, volume 6 of *Fundamental Studies in Computer Science*. North-Holland Publishing Company, 1978.

[LY02] Fangzhen Lin and Jia-Huai You. Abduction in Logic Programming: A New Definition and an Abductive Procedure Based on Rewriting. *Artificial Intelligence*, 140:2002, 2002.

[MN08] Ralf Möller and Bernd Neumann. Ontology-Based Reasoning Techniques for Multimedia Interpretation and Retrieval. In Yiannis Kompatsiaris and Paola Hobson, editors, *Semantic Multimedia and Ontologies: Theory and Applications*, pages 55–98. Springer, Heidelberg, 2008.

[MP92] Enzo Marinari and Giorgio Parisi. Simulated Tempering: A New Monte Carlo Scheme. *Europhysics Letters*, 19:451–458, 1992.

[Nea93] Radford M. Neal. Probabilistic Inference using Markov Chain Monte Carlo Methods. Technical report, University of Toronto, Department of Computer Science, September 1993.

[Neu08] Bernd Neumann. Bayesian Compositional Hierarchies - A Probabilistic Structure for Scene Interpretation. In Anthony G. Cohn, David C. Hogg, Ralf Möller, and Bernd Neumann, editors, *Logic and Probability for Scene Interpretation*, number 08091 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2008. Schloss Dagstuhl - Leibniz - Zentrum fuer Informatik.

[NH97] Liem Ngo and Peter Haddawy. Answering Queries from Context-Sensitive Probabilistic Knowledge Bases. *Theoretical Computer Science*, 171:147–177, 1997.

[NM06] Bernd Neumann and Ralf Möller. On Scene Interpretation with Description Logics. In H.I. Christensen and H.-H. Nagel, editors, *Cognitive Vision Systems: Sampling the Spectrum of Approaches*, volume 3948 of *Lecture Notes in Computer Science*, pages 247–278. Springer, Heidelberg, 2006.

[NM08] Tobias Henrik Näth and Ralf Möller. ContraBovemRufum: A System for Probabilistic Lexicographic Entailment. In *Proceedings of the Twenty-First International Workshop on Description Logics (DL2008)*, 2008.

[Pau93] Gabriele Paul. Approaches to Abductive Reasoning: An Overview. *Artificial Intelligence Review*, 7(2):109–152, 1993.

[PD06] Hoifung Poon and Pedro Domingos. Sound and Efficient Inference with Probabilistic and Deterministic Dependencies. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, pages 458–463, Boston, Massachusetts, July 2006. AAAI Press.

[Pea88] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Series in Representation and Reasoning. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1988.

[Pei78] Charles Sanders Peirce. Deduction, Induction and Hypothesis. In *Popular Science Monthly 13*, pages 470–482, 1878.

[Poo91] David Poole. Representing Bayesian Networks within Probabilistic Horn Abduction. In Bruce D'Ambrosio and Philippe Smets, editors, *Proceedings of the Seventh Annual Conference on Uncertainty in Artificial Intelligence (UAI 1991)*, pages 271–278, University of California at Los Angeles, Los Angeles, CA, USA, July 1991. Morgan Kaufmann Publishers.

[Poo97] David Poole. The Independent Choice Logic for Modelling Multiple Agents under Uncertainty. *Artificial Intelligence*, 94(1–2):7–56, 1997.

[Poo08a] David Poole. *AILog User Manual*. Department of Computer Science, University of British Columbia, Vancouver, B.C., Canada, December 2008.

[Poo08b] David Poole. The Independent Choice Logic and Beyond. In Luc De Raedt, Paolo Frasconi, Kristian Kersting, and Stephen Muggleton, editors, *Probabilistic Inductive Logic Programming: Theory and Application*, volume 4911 of *Lecture Notes In Computer Science*, pages 222–243, Department of Computer Science, University of British Colombia, Vancouver, B.C., Canada, 2008. Springer.

[Rei87] Raymond Reiter. On Closed World Data Bases. In Matthew L. Ginsberg, editor, *Readings in Nonmonotonic Reasoning*, pages 300–310. Morgan Kaufmann Publishers Inc., Los Altos, CA, USA, 1987.

[RN03] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Second edition, 2003.

[Rot96] Dan Roth. On the Hardness of Approximate Reasoning. *Artificial Intelligence*, 82:273–302, 1996.

[RSDD09] Michele Ruta, Floriano Scioscia, Tommaso Di Noia, and Eugenio Di Sciascio. Reasoning in Pervasive Environments: An Implementation of Concept Abduction with Mobile OODBMS. In *Proceedings of IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent*

Technology (WI-IAT 2009), volume 01, pages 145–148, Milan, Italy, September 2009. IEEE Computer Society.

[SD05] Parag Singla and Pedro Domingos. Discriminative Training of Markov Logic Networks. In Manuela M. Veloso and Subbarao Kambhampati, editors, *Proceedings of the 20th National Conference on Artificial Intelligence and the 17th Innovative Applications of Artificial Intelligence Conference*, volume 2, pages 868–873, Pittsburgh, Pennsylvania, USA, July 2005. AAAI Press / The MIT Press.

[SD08] Parag Singla and Pedro Domingos. Lifted First-Order Belief Propagation. In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, pages 1094–1099, Chicago, Illinois, USA, July 2008. AAAI Press.

[Sha05] Murray Shanahan. Perception as Abduction: Turning Sensor Data into Meaningful Representation. *Cognitive Science*, 29(1):103–134, 2005.

[SKC96] Bart Selman, Henry Kautz, and Bram Cohen. Local Search Strategies for Satisfiability Testing. In David S. Johnson and Michael A. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 521–531, Washington, DC, 1996. American Mathematical Society.

[SP06] Evren Sirin and Bijan Parsia. Pellet System Description. In *Proceedings of the 2006 Description Logic Workshop (DL-2006)*. CEUR Electronic Workshop Proceedings, 2006.

[Tha78] Paul R. Thagard. The Best Explanation: Criteria for Theory Choice. *The Journal of Philosophy*, 75(2):76–92, February 1978.

[Var82] Moshe Y. Vardi. The Complexity of Relational Query Languages. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC '82)*, pages 137–146, San Fransisco, California, USA, May 1982. ACM.

[YFW01] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Generalized Belief Propagation. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 689–695. MIT Press, Cambridge, MA, 2001.