ESPRIT 2434

Contribution to:

Activity T2.5

- System Concept for Advanced CIM/AI Controller --

Title:

EVALUATION OF ALLEN'S QUALITATIVE TEMPORAL INFERENCE ALGORITHM FOR LATER HARDWARE IMPLEMENTATION

Author/Company/Copyright:

Author: Jörg - Ingo Jakob

Achievements:

Allen's qualitative temporal logic is analysed regarding (1) improvement of run time performance (in software), (2) identification of hardware components for custom hardware implementation. A suggestion for a speedup of a central part of Allen's algorithm by a factor of 5 to 10 for typical applications is made (in software). A list of hardware building blocks that compose a hardware temporal logic accelerator is presented. The improved software algorithm presented will serve as a performance reference point for a future hardware implementation.

Summary:

Prior to ESPRIT project 2434, a software module based on Allen's temporal logic was developed for a flow line scheduling expert system at Philips Hamburg. Investigations within ESPRIT 2434 showed the slowness of this software module when applied to larger scheduling tasks (ten or more orders). We expect a hardware solution to yield a speedup of another order of magnitude for temporal inference when applied to the domain of order scheduling. Two questions are answered: (1) what is the best software solution for temporal inference ?; (2) can we identify hardware components starting from the best (or any other reasonable) software solution ?

0. Introduction

Allen's approach to qualitative temporal inference was first described in [Allen 1983]. Prior to ESPRIT project 2434, a software module based on Allen's temporal logic was developed for a flow line scheduling expert system at Philips Hamburg [Becker 1988] [Huber 1990]. Investigations within ESPRIT 2434 showed the slowness of this software module when applied to larger scheduling tasks (ten or more orders) [Huber/Jakob 1989]. We expect a hardware solution to yield a rapid speedup for temporal inference when applied to the domain of order scheduling. In order to come to definite terms on the hardware structure of such a 'temporal inference accelerator unit (TIU)', two questions must be answered: (1) what is the best software solution for temporal inference ?; (2) can we identify hardware components starting from the best (or any other reasonable) software solution ?

Note: This paper assumes some familiarity with [Allen 1983].

1. Allen's Temporal Logic

In his original article [Allen 1983], Allen presented several concepts and components which make up a software representation for temporal inference:

- the idea of time intervals (temporal intervals) and temporal relations between them. In principle, each time interval is connected to any other time interval, yielding a network of temporal relations. (In practice, however, reference intervals may be used to reduce computational load).
- thirteen canonical temporal relations between time intervals, namely BEFORE, EQUAL, MEETS, OVERLAPS, DURING, STARTS, FINISHES and their inverses (called 'primitive temporal relations' in the further course of this paper).
- a transitivity table for primitive temporal relations.
- an algorithm CONSTRAINTS which computes temporal relations between any three time intervals. These relations may be non-primitive relations in the sense that they are composed of several primitive temporal relations. (Such relations are called 'compound temporal relations' in the further course of this paper).
- a propagation mechanism for temporal constraints which consists (in its simplest form) of the following components:
  - a queue data structure ToDO that collects unpropagated temporal constraints on the network;
  - an algorithm ToADD for updating the temporal network using the unpropagated constraints of ToDO.

The 13 primitive temporal relations introduced by Allen are by no means the only conceivable set of primitive temporal relations. Allen introduces an alternative set consisting of only nine primitive temporal relations (BEFORE, EQUAL, MEETS, OVERLAPS, DURING and inverses). At Philips, we sometimes use a set of only six primitive relations (BEFORE, OVERLAPS, DURING and inverses); e.g. cf. [Kemper 1990]. The number of primitive temporal relations does not affect the basic algorithm as described by Allen. However, it affects the time required to compute a compound relation using the CONSTRAINTS algorithm. On the other end, there may well exist other sets that feature more than 13 primitive relations for special applications. The larger the number of primitive relations, the higher the semantic information content of each relation.

In general, I assume an arbitrary number of S primitive temporal relations throughout this paper.

2. Speeding up Allen's 'CONSTRAINTS' Algorithm

The CONSTRAINTS algorithm can be written in Common LISP as:

```
(defun CONSTRAINTS (R1 R2)
    (let ((c nil))
        (dolist (an-R1 R1 c)
            (dolist(an-R2 R2)
                (setq c (union c (T an-R1 an-R2))))))
```

Here, T is the transitivity function defined by Allen's transitivity table.

Thus, the number of iterations throughout the nested loops is determined by S^2. Depending on the number N of temporal relations that make up a compound temporal relation, we receive:

No. of Iterations I	Minimum N = 1	Average N = S/2	Maximum N = S
for any S	1	$S^2/4$	S^2
for S=6	1	9	36
for S=9	1	20.25	81
for S=13	1	42.25	169
for S>13	1	> 48	> 195.

Analysis of the Common LISP-based FLEX expert system [Huber/Jakob 1989] confirms that CONSTRAINTS is a time-critical part in practical applications of Allen's temporal logic, due to the large number of iterations spent on CONSTRAINTS when propagating temporal relations. In FLEX, it consumes between 40 to 70 % of run time (depending on the application).

This insight led to the search for a way to speed up CONSTRAINTS. In the sequel, I will describe an alternative algorithm for CONSTRAINTS, called CONSTRAINTS-P, that nearly has a constant time behaviour.

Basic idea behind CONSTRAINTS-P (CONSTRAINTS Probabilistic) is to identify an appropriate subset of all possible compound temporal relations. This subset shall contain all those compound relations that are (on the average) frequently assigned to edges in the temporal network. Then, an extended transitivity table is computed. Its row and column index range does not only consist of the primitive temporal relations, instead the compound relation subset members are also assigned one row and column each. By enlarging the transitivity table this way, function (T ... will be called only once (median value) when calling CONSTRAINTS:

```
(defun CONSTRAINTS-P (R1 R2)
```

```
(if
    (and
        (subset-member-p R1)
        (subset-member-p R2))
    (T R1 R2)
    (CONSTRAINTS R1 R2)))
```

SUBSET-MEMBER-P is a function that decides whether or not a relation is member of the above subset of compound relations.

Choice of subset: so far, no explanation has been given how to choose an appropriate subset of all compound relations, in order to set up an extended transitivity table. There are different ways to this aim, and one simple way shall be presented. Provided a software implementation of Allen's temporal logic is available, some additional code is inserted into the source which monitors the number of times each single compound relation is used. A Monte Carlo simulation is made on arbitrary temporal nets, and the outcome is transformed into a temporal compound relation usage statistics. The compound relations which are used most often are selected for inclusion into the subset. This attempt is a practical one: although there are $2^{13} = 8192$ different compound relations for S = 13, only between twenty and one hundred of them turn out to be candidates for subset inclusion due to their relative frequency (but also limited by the memory size of the transitivity table one intends to implement).

3. Possibilities of Speeding Up Allen's 'ToADD'

Currently under investigation is a technique to replace the queue data structure ToDO. Instead of using ToDO or a similar data structure, a parametrised function (defun PROPAGATE-EDGE (EDGE.NUMBER) ... is introduced. All edges are identified by unique numbers 1 .. MAXEDGE. After the compound relation assigned to an edge numbered E was modified, a call to PROPAGATE-EDGE is made, with the edge number E as parameter. Propagating the modified relation on edge E will very likely trigger modifications on other edges in the temporal network. These modifications will call PROPAGATE-EDGE in turn, etc., until the net is in a steady state. The structure of PROPAGATE-EDGE does not call for a queue data structure or similar, because the necessary modification information is implicitely recorded in the recursive calls to PROPAGATE-EDGE. On the other hand, recursive calls use a stack structure which is maintained by the underlying run time system (provided by the language compiler or interpreter), and it is not yet decided whether this new approach to ToADD will be favourable or not, in terms of run time performance.
This alternative ToADD algorithm is called ToADD-R (R for recursive).

4. Performance Measurements on 'CONSTRAINTS-P'

At Philips, CONSTRAINTS-P was implemented using Allen's set of primitive temporal relations (S=13), and with an alternative set (S=6). The TIC component of the FLEX expert system was monitored when performing production planning tasks. For this application domain (S=13), we found a typical hit rate of more than 97 % (hit rate being defined as the ratio of calls to CONSTRAINTS-P, the result of which are a row and column members in the extended transitivity table, to all calls made to CONSTRAINS-P; this definition accounts for the fact that most edges in a typical temporal net are 'inner' edges that are not explicitely constrained by the application tasks and domain).
Another program (a version of TIC software modules re-programmed in a conventional procedural language) that uses S=6 primitive temporal relation showed a typical hit rate of more than 92 %. This program is not integrated into the FLEX environment and uses a simplified model of temporal relations, therefore we assumed an evenly distribution of compound temporal relations.

This last method can be generalised: no matter how large S may be, by performing a Monte Carlo simulation on calls to (CONSTRAINS-P R1 R2), with two random variables R1 and R2, an approximate hit rate can always be established. This is a remedy against the combinatorical explosion:

all primitive relations	all compound relations	all combinations of R1, R2
S	2^S	2^{2S}
6	64	4,096
9	512	262,144
13	8192	67,108,864
> 13	> 16383	> 268,435,455.

Disregarding the constant time spent on calls to (SUBSET-MEMBER-P ..., we achieve a significant average speedup U of CONSTRAINTS-P over CONSTRAINTS. By applying the values I for the average number of iterations (found in the first table), we receive measures for the average number of iterations relative to one call of CONSTRAINTS-P:

for S = 6 and I = 9: 0.92*1 + 0.08*I vs. 1.00*I
=> U = 5.49

for S = 13 and I = 42.25: 0.97*1 + 0.03*I vs. 1.00*I
=> U = 19.32.

Conclusion: we may expect an average speedup U of CONSTRAINTS-P over CONSTRAINTS of one order of magnitude. This applies for Allen's original temporal logic (S=13) and its usage in a domain similar to the FLEX expert system for production scheduling. Even for "simpler" temporal logics (S=6), a significant speedup can be perceived.

Notes: (1) the technique of speeding up CONSTRAINTS presented so far is purely a software technique; (2) the fact that the part of CONSTRAINTS-P performed in constant time is used in more than 90 % of calls to CONSTRAINTS-P (no matter what value S has) justifies the statement of tendency that "... CONSTRAINTS-P has nearly a constant time behaviour." However, the average number of iterations I may dominate the constant time contribution of CONSTRAINTS-P, depending on the chosen size of the extended transitivity table and the application domain.

5. Practical Experiences and Outlook

The TIC component of the FLEX expert system uses between 40 to 70 % of its run time on functions implementing the (conventional) CONSTRAINTS algorithm as presented by Allen [Huber/Jakob 1989]. (The exact value depends on the scheduling task.) Rewriting this code using CONSTRAINTS P, as presented it this paper, would thus result in an expected run time reduction of between 36 to 67 % of one scheduling run (five minutes instead of ten minutes for a six order scheduling task, on a Symbolics 3640 Lisp machine). Abandoning list representation of data by using conventional procedural languages like C, these times can be reduced by another factor of two or three, making temporal logic-based production scheduling for small and medium sized planning tasks (up to 10 orders) a reality. Beyond ten orders, only a custom hardware accelerator will help.

6. The "Best" Software Solution

The introduction of the CONSTRAINTS-P algorithm (and, possibly, of ToADD-R, as future will show) seems to provide the fastest possible algorithm for Allen's temporal logic on the traditional (von Neumann) monoprocessor. No attempt was made so far to find a formal proof for this assumption. However, due to its elegant brevity, Allen's temporal network algorithm does not seem to provide more clues for optimisation (because not much is left which, possibly, could be optimised).

7. Identification of Possible Hardware Components

An "ideal" hardware temporal network would take advantage of parallelism. It may consists of:

- the network, consisting of N nodes and E = N*(N-1)/2 edges (storage locations for compound relations);
- T = N*(N-1)*(N-2)/6 "triangles" (i.e., valid transitive compound relations between any three nodes). These triangles may be interpreted as T hardware instances of the (extended) transitivity table.
- a propagation discipline: when can exterior constraints be given onto the net; when are interior constraints -- that result from exterior and interior constraints -- propagated. This discipline would presumably use some kind of execution parallelism.

Practical hardware solutions cannot provide T ~ N^3 triangles, etc., even for small N. The temporal network and/or the propagation discipline must

be cut into smaller parts, i. e. appropriate subsets of all temporal net nodes. One such subset can be computed at one time, and all subsets will be computed by computing them one after another sequentially. This loop over subsets of the net will be repeated until the net is in a steady state.

8. Literature

[Allen 1983] J. F. Allen, Maintaining Knowledge about Temporal Intervals; in: Comm. of the ACM Vol. 26 No. 11 (November 1983), pp. 832 - 843.

[Becker 1988] S. U. Becker, Konzeption und Implementation einer temporalen Inferenzkomponente -- Anwendung auf Planungsprobleme in der Fließproduktion; Diplomarbeit am Fachbereich Informatik der Universität Hamburg 1988.

[Huber 1990] A. Huber, Wissensbasierte Überwachung und Planung in der Fertigung; Dissertation am Fachbereich Informatik der Technischen Universität Berlin 1990, pp. 125 ff.

[Huber/Jakob 1989] A. Huber / J. - I. Jakob, Requirements of Temporal Constraint Propagation; in: ESPRIT 2434 6 months report, Task 2.5.

[Kemper 1990] A. Kemper, CIM Acceleration Hardware: Review of System Requirements; in: ESPRIT 2434 12 Months Report, Task 2.5.