Logic Programming with Temporal Constraints *

E. Schwalb L. Vila

Information and Computer Science Dept.
University of California, Irvine

Abstract

In this work we combine logic programming and temporal constraint processing techniques. We propose TCLP, which augments logic programs with temporal constraints. Known algorithms for processing disjunctions in Temporal Constraint Networks are applied. We identify a decidable fragment called, Simple TCLP, which can be viewed as extending Datalog with limited functions to accommodate intervals of occurrence and temporal constraints between them. Some of the restrictions introduced by Simple TCLP are overcome by a syntactic structure which provides with the benefits of reification. The latter allows quantification on temporal occurrences and relation symbols.

1 Introduction

Representing time is central to both the databases and AI communities. In both the information includes temporal attributes which indicate at what time the assertions are true.

In AI, the goal of temporal languages is to describe and reason about a changing world. Most of current research is focused on either general theories of time or restricted computational models which involve temporal constraints. Many temporal theories were proposed, among which [12, 1, 16, 11, 5, 2, 19, 10]. As these languages are based on first order logic, they are sufficiently expressive for representing general knowledge however answering queries is in general undecidable. This motivated identifying many decidable subclasses and analizing their complexity [10]. Independently, the computational model of Temporal Constraint Satisfaction Problems, abbreviated TCSP, was investigated by [7, 17, 9, 14, 13]. Although it is supported by efficient algorithms, it cannot process combinations of temporal and non-temporal sentences.

In the database community, the classical approach is representing relations explicitly, that is, by listing their tuples. Clearly, relations which have infinite sets of tuples cannot be represented explicitly. As an implicit representation of finite relations, it is common to use function free logic programs called Datalog. To provide with a finite implicit representation of infinite relations, Datalog$_{nS}$ was proposed [3], which is an extension of datalog with a restricted class of functions. Unfortunately, Datalog$_{nS}$ is not applicable to AI because it does not support indefinite temporal information, namely disjunctive relations between temporal occurrences.

We propose TCLP, which supports a new constraint class based on Temporal Constraint Satisfaction Problems. The performance benefits can be obtained whenever the temporal constraints are disjunctive. In that case, constraint propagation results in prunning the search space and reduces the number of dead-ends encountered.

Our proposal differs from traditional extensions of logic with temporal data in that we introduce, in addition to the usual temporal sort, a new sort called tokens. The latter was shown to allow obtaining the benefits of reification while avoiding the technical complications of reification [6].

We identify a decidable fragment called Simple TCLP which allows non-constraint predicates to specify at most a single argument of the token sort. Based on results obtained for Datalog$_{nS}$, we show that the least Herbrand model of Simple TCLP programs admits a finite representation on which queries can be answered efficiently.

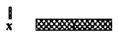
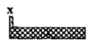
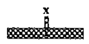
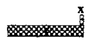
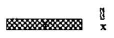
To overcome the syntactic restriction posed by Simple TCLP, we propose a syntactic structure which provides with the benefits of reification as it allows quantification over temporal occurrences and relation symbols. We propose to model the world as a relational database. This requires two additional sorts: relation, attribute sorts. Each relations describe classes of objects, events and actions. Instances of these classes are tuples in these relation. Tuples may specify null (or unknown) values and are associated with either time points or time intervals. Simple TCLP programs are used as an implicit representation of this possibly infinite relational database.

The paper is organized as follows: Section 2 presents the basic language. Section 3 presents the conputational benefits of introducing temporal constriants. Section 4 presents a decidable fragment which admits a finite representation. Section 5 presents the syntactic structure that provides with the benefits of reification.

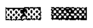
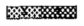
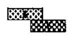
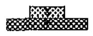
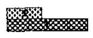
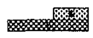
*We would like to thank Rina Dechter for fruitful discussions.

Table 1:

(a) The 5 qualitative point-interval relations (X is a point and Y is an interval).

Relation	Symbol	Inverse	Example
X before Y	b	bi	
X starts Y	s	si	
X during Y	d	di	
X finishes Y	f	fi	
X after Y	a	ai	

(b) The 13 qualitative Interval-Interval relations.

Relation	Symbol	Inverse	Example
X before Y	b	bi	
X equal Y	=	=	
X meets Y	m	mi	
X overlaps Y	o	oi	
X during Y	d	di	
X starts Y	s	si	
X finishes Y	f	fi	

2 The Language

We use the following set of distinguished temporal sorts $S = \{P, I, PT, IT, D\}$: P is a set of time points, I is a set of time intervals, PT is a set of point time tokens and IT interval time tokens and D is the temporal domain assumed to be the set of natural numbers. We use two distinguished function symbols: *point* : $PT \mapsto P$ and *interval* : $PI \mapsto I$. The set of constraint predicates is composed of the metric *Point* relations ($P \times P$), qualitative *Point-Interval* relations ($P \times I$) and the qualitative *Interval* relations ($I \times I$).

We propose Temporal Constraint Logic Programs, TCLP, that are Logic Programs augmented by the following set of temporal constraints:

- disjunction of metric point relations (namely $X_i - X_j \in [a, b]$) [14, 13]
- disjunction of point interval relations (table 1 a) [13]
- disjunction of interval relations relations (namely the *Interval Algebra* in table 1b) [1]

The semantics is defined as follows: For the satisfaction relation of the temporal constraints we use the usual evaluation rules given in [1, 14, 13]. For the non-constraint atoms we use the standard evaluation rules of logic programs. The program characterized is by its *unique least Herbrand model.*

3 Performance Benefits

The goal of combining logic programming and temporal constraints is to improve the performance of resolution algorithms by augmenting them with specialized constraint propagation algorithms. In the presence of disjunctive constraints, existing search mechanism requires testing consistency of every possible selection of a single disjunct from each constraint. A more effective technique performs temporal constraint propagation which reduces the number of disjuncts [15].

We consider metric constraint of the form $X_j - X_i \in I_1 \cup \cdots \cup I_k$ where $I_1, \cdots, I_k$ is a set of disjoint intervals. The corresponding logic program for this constraint consists of the rules:

$$(C_{ij} \leftarrow X_j - X_i \in I_1), \ldots, (C_{ij} \leftarrow X_j - X_i \in I_k)$$

For this class of constraints, algorithm LPC, briefly described below, is capable of removing redundant disjunctions [15].

3.1 Temporal Constraint Networks

A Temporal Constraint Satisfaction Problem (TCSP) consists of a set of variables $X_1, \ldots, X_n$, having *rational* domains, each representing a time point. Each constraint C is a set of intervals

$$C \stackrel{\text{def}}{=} \{I_1, \ldots, I_n\} = \{[a_1, b_1], \ldots, [a_n, b_n]\}.$$

A unary constraint C_i restricts the domain of the variable X_i to the given set of intervals

$$C_i \stackrel{\text{def}}{=} (a_1 \leq X_i \leq b_1) \cup \ldots \cup (a_n \leq X_i \leq b_n).$$

A binary constraint C_{ij} over X_i, X_j restricts the permissible values for the distance $X_j - X_i$; it represents the disjunction

$$C_{ij} \stackrel{\text{def}}{=} (a_1 \leq X_j - X_i \leq b_l) \cup \ldots \cup (a_n \leq X_j - X_i \leq b_n).$$

All intervals are assumed to be open and pairwise disjoint.

A tuple $X = (x_1, \ldots, x_n)$ is called a *solution* if the assignment $X_1 = x_1, \ldots, X_n = x_n$ satisfies all the constraints. The network is *consistent* iff at least one solution exists.

Definition 1 : [composition & loose intersection] Let $T = \{I_1, I_2, \ldots, I_r\}$ and $S = \{J_1, J_2, \ldots, J_s\}$ be

Algorithm Loose Path-Consistency (LPC)

```
1. input: $N$
2. $N'' \leftarrow N$
3. repeat
4.     $N \leftarrow N''$
5.     Compute $N'$ by assigning $T'_{ij} = \cap_{\forall k}(C_{ik} \otimes C_{kj})$, for all $i, j$.
6.     Compute $N''$ by loosely intersecting $T''_{ij} = C_{ij} \triangleleft T'_{ij}$, for all $i, j$.
7. until $\exists i, j \; (T''_{ij} = \phi)$   ; inconsistency, or
         or $\forall i, j \; |T''_{ij}| = |C_{ij}|$ ; no interval removed.
8. if $\exists i, j \; (T''_{ij} = \phi)$ then output "inconsistent."
                              else output: $N''$.
```

Figure 1: The Loose Path-Consistency (LPC) algorithm.

two constraints. The *composition* of T and S, denoted by $T \otimes S$, admits only values r for which there exists $t \in T$ and $s \in S$ such that $r = t + s$. The *loose intersection*, $T \triangleleft S$ consists of the intervals $\{I'_1, \ldots, I'_r\}$ such that $\forall i \; I'_i = [L_i, U_i]$ where $[L_i, U_i]$ are the lower and upper bounds of the intersection $I_i \cap S$.

The number of intervals in C_{ij} is not increased by the operation $C_{ij} \leftarrow C_{ij} \triangleleft (C_{ik} \otimes C_{kj})$. In addition, $\forall k \;\; C_{ij} \supseteq C_{ij} \triangleleft (C_{ik} \otimes C_{kj}) \supseteq C_{ij} \cap (C_{ik} \otimes C_{kj})$ and $T \triangleleft S \neq S \triangleleft T$.

Example 1 : Let $T = \{[1,4],[10,15]\}$ and $S = \{[3,11],[14,19]\}$. Then $T \triangleleft S = \{[3,4],[10,15]\}$, $S \triangleleft T = \{[3,11],[14,15]\}$ while $S \cap T = \{[3,4],[10,11],[14,15]\}$.

Algorithm LPC is presented in Figure 1. The network N' is a relaxation of N and therefore loosely intersecting it with N results in an equivalent network. At every iteration if LPC (except the first and the last) at least one interval is removed. This allows us to conclude that:

Theorem 1 : *Algorithm LPC computes an equivalent network in polynomial time.*

Example 2 : Consider the constraints:

$$\begin{array}{lcl}
X_1 - X_0 & \in & [10,20] \cup [100,110] \\
X_2 - X_1 & \in & [20,40] \cup [100,130] \\
X_3 - X_0 & \in & [80,100] \cup [150,160] \cup [180,190] \\
X_3 - X_1 & \in & [30,40] \cup [130,150] \\
X_3 - X_2 & \in & [50,70] \cup [110,120] \cup [130,140] \cup [160,190]
\end{array}$$

After 3 iterations, algorithm LPC terminates with the network:

$$\begin{array}{lcl}
X_1 - X_0 & \in & [10,20] \\
X_2 - X_0 & \in & [30,50] \\
X_2 - X_1 & \in & [20,30] \\
X_3 - X_0 & \in & [150,160] \\
X_3 - X_1 & \in & [130,140] \\
X_3 - X_2 & \in & [110,120]
\end{array}$$

4 A Simple Fragment

We investigate a restricted class, called Simple TCLP, in which there are no free-variables (i.e. only token and data variables) and restricted such that functions and atoms specify at most a single token argument. This section applies results obtained by [3, 4] to Simple TCLP which accept binary temporal constraints on token terms.

4.1 Token Succession

The central notion that enables us to obtain our results is the *token succession*. Intuitively, instead of indexing facts with time points which are linearly ordered we index facts by tokens which need not have any temporal ordering. Unary functions (used in Datalog$_{nS}$ as successor functions) are used here to obtain successors of tokens. These functions define the succession ordering of tokens which is orthogonal to their temporal ordering.

Definition 2 : Given a set of unary functions $f_1, \ldots, f_k$ mapping token symbols to token symbols, the domain $K(f_1, \ldots, f_k)$ of token terms is as follows:

$$tt_{\underbrace{0,\ldots,0}_{k\ times}} \in K(f_1, \ldots, f_k) \quad and$$

$$tt_{i_1,\ldots,i_k} \in K(f_1, \ldots, f_k) \implies$$
$$f_j(tt_{i_1,\ldots,i_k}) = tt_{i_1,\ldots,i_j+1,\ldots,i_k} \in K(f_1, \ldots, f_k)$$

In other words, $tt_{i_1,\ldots,i_k}$ is obtained by applying, for all $j \in [1,k]$, f_j on $tt_{0,\ldots,0}$, i_j times.

For Simple TCLP the set of ground token terms is $K(f_1, \ldots, f_k)$.

4.2 Restrictions

We make a number of restrictions and assumptions which originate in Datalog$_{nS}$. A TCLP is said to be *simple* if it complies with the following restrictions:

- The domains of non-token variables are finite.
- Rules do not contain ground terms. Such terms can be eliminated by introducing additional predicates.
- Rule bodies are not empty.
- Equalities are eliminated by replacing variables in a class of equated variables with a single representative.
- *Rules are range restricted.* After elimination of equalities, a variable is limited if it appears in a literal in the body of the rule.
- *Rules are normal.* A TCLP rule r is *semi-normal* if every atom in r contains at most 1 token variable, namely every predicate $P(tt_1, \ldots, tt_m, d_1, \ldots, d_k)$ is such that $m = 1$. A semi-normal rule r is normal if every non-ground functional term in r is of depth at most 1. This restriction can be lifted by the syntactic structure described below.

- Functions must specify a single argument which is a token variable, but the number of variables of other sorts is not restricted.

4.3 Finite Representation

The general notion of finite representation of infinite relations relies on the notion of *homomorphic* mapping H. A finite representation of a logic program P is a homomorphism from the least Herbrand M_P onto a finite structure M_P^*. The existence of such an homomorphism guarantees that the finite structure is equivalent to the least Herbrand model.

Theorem 2 : *For every Simple TCLP P there exists a homomorphism mapping M_P to a finite structure, namely its least model M_P is finitely representable.*

We illustrate this feature by presenting instances from two seemingly similar classes of programs: One is a class of CLP programs which have no finite representation. The other is a class of TCLP programs which admits a finite representation.

Consider the following simple program, given in [4], for which there is no finite representation:

$$\begin{array}{l} eq(0,0). \\ eq(T+1,T+1) \leftarrow eq(T,T) \end{array}$$

The reason no finite representation exists is that no homomorphism exists. Assume the contrary, then as shown in [4] the existence of a homomorphism implies the existence of a period l such that for every predicate $P(T_1,T_2) \Leftrightarrow P(T_1+l,T_2) \Leftrightarrow P(T_1,T_2+l) \Leftrightarrow P(T_1+l,T_2+l)$. Thus the truth value of $eq(T,T)$ and of $eq(T+l,T)$ must be the same for every grounding of T, which is a contradiction.

Fortunately, TCLP programs using the predicate $Eq(tt_1,tt_2)$ admit finite representations because $Eq(tt_1,tt_2)$ differs from $eq(tt_1,tt_2)$:

$$\begin{array}{lcl} Eq(tt_i,tt_j) & \leftarrow & (begin(tt_i)-begin(tt_j) \in [0,0]), \\ & & (end(tt_i)-end(tt_j) \in [0,0]). \end{array}$$

The truth value of $Eq(tt_i,tt_{i+l})$ may be false because there is no restriction in the language that requires that $begin(tt_i)=begin(tt_{i+l})$ or $end(tt_i)=end(tt_{i+l})$. Thus, no contradiction is derived.

The following TCLP program, given in [4], uses a full fledged binary token function and thus does not admit a finite representation:

$$\begin{array}{l} p(tt_0). \\ p(g(tt,tt)) \leftarrow p(tt). \end{array}$$

4.4 Decidability

In general, it is not clear whether the existence of a finite representation implies decidability. For Simple TCLP, as for Datalog$_{nS}$, due to their periodic nature, bottom up evaluation with a finite depth bound is complete.

Given a program $P=(Z \cup D)$ where Z is a set of rules and D is a set of ground facts, answering a query Q amounts to deciding consistency of the formula which consists of $Z \cup D \cup \neg Q$. In the following we assume that Q is grounded. In this case, the answer to the query is 'Yes' iff the formula $Z \cup D \cup \neg Q$ is unsatisfiable.

Define the following parameters:

- k is the maximal arity of predicates in Z and D;
- d is the number of different data constants in D;
- c is the maximum depth of a ground token term in D (if $c=0$ there is no such term);
- h is the depth of the single functional term in Q (if $h=0$ there is no such term);
- t_i is the number of token predicates of arity i.

Theorem 3 : *m-bounded bottom-up evaluation of a yes/no (closed) query is sound and complete for $m=max(c,h)+2^s$, $s=\sum_{i=0}^{k} t_i d^{i-1}$.*

5 Modeling a Changing World

Often it seems that modeling the world poses difficulties caused by limited expressiveness of the language used. This issue was addressed *implicitly* in [11][1] when discussing representation of incomplete descriptions of events. Here, Simple TCLP appear to be too restricted. We show that Simple TCLP are sufficiently expressive for AI applications. We propose a syntactic structure that allows to obtain the benefits of reification.

First, we describe some general problems through an example and thereafter we provide with a solution. Consider the statement "John and Fred were roommates, but now, John owes Fred money, hates him and threatens to kill him. John unloaded his gun 10 minutes ago, but later he loaded it and now the gun is pointing at Fred". Consider the queries:

1. "What is the relationship between John and Fred now ?". The answer is "John owes money, hates and threatens Fred".
2. "What is the status of the gun now ?". The answer is "the gun is loaded and pointing at Fred."
3. "When did the gun get loaded ?". The answer is "John loaded the gun between 10 minutes ago and now.

A possible way to represent (in first order logic) the above temporal information is to use the following predicates: *Loaded*(Gun,t), *PointedAt*(X,Y,t),

[1]In section 8 of that paper.

$Owe(X,Y,t)$, $Hate(X,Y,t)$ and $Threaten(X,Y,t)$ where t is a temporal qualification[2].

Clearly, if we are to represent the queries described above in first order logic, we need a different set of predicates. This is because in first order logic it is not possible to quantify over predicate symbols, as seems to be required for representing the above three queries.

5.1 Temporal Data Model

We define the token and database structures of we use to represent the world. In the next section we show that these structure can be described by Simple TCLPs.

Definition 3 : [tokens]
A *data value* is a pair $\langle a, v\rangle$, where a is a symbol describing an attribute and v is a symbol describing a constant which is the value assigned to this attribute. There are two types of *time tokens*: instant and durative. An *instant time token* is a triplet $\langle n_{tt}, t, \mathcal{V}\rangle$ where n_{tt} is a symbol specifying its name, t is a rational constant specifying the time point associated with tt and $\mathcal{V}$ is the set of data values. A *durative time token* tt is a quadruple $\langle n_{tt}, t_1, t_2, \mathcal{V}\rangle$ where n_{tt} is a symbol specifying its name, t_1 and t_2 are rational constants specifying the the beginning and end time points of the interval associated with tt and $\mathcal{V}$ is the set of data values.

Example 3 : To specify that 'the gun was loaded at 9:00 and unloaded at 9:10 we use two *instant time tokens* whose set $\mathcal{V}$ specifies a single data value:

$$\begin{aligned} tt_1 &= \langle N1,\ 9{:}00,\ \{\langle Loaded,\ true\rangle\}\rangle\,, \\ tt_1 &= \langle N2,\ 9{:}10,\ \{\langle Loaded,\ false\rangle\}\rangle \end{aligned}$$

We model the world using a (possibly infinite) set of Time Tokens organized as a (possibly infinite) relational database. A *class*, described by a relation, is a collection of *objects, events* or *actions* which have the same set of attributes. A similar approach was proposed in [11].

Definition 4 : [databases]
A tuple is an instant or a durative time token whose set of data values $\mathcal{V}$ is finite. A *token relation* is a (possibly infinite) set of time tokens and has three distinguished attributes called *ID*, *Begin* and *End*, that specify unique token names and the beginning and end time points of tokens respectively. If a tuple specifies an instant time token the values assigned to the attributes *Begin* and *End* are equal. A *token database* is a finite set of relations.

Example 4 : We might have a relation 'person' describing people, with the attributes 'name', 'sex', 'eye-color', 'hear-color' and 'salary'. In this relation, each tuple describes a person by specifying his/her name and the values of some of the attributes. To specify that John is a male and list his salary for the year 1980 we could use the relations:

Person					
ID	*Begin*	*End*	*Name*	*Sex*	*SalaryPointer*
N0	1/1/80	12/31/80	*John*	*Male*	*P1*

John's 1980 *Salary*				
ID	*Begin*	*End*	*SalaryPointer*	*Salary*
N1	1/1/80	1/31/80	*P1*	\$2000
N2	2/1/80	2/28/80	*P1*	\$2200
⋮	⋮	⋮	⋮	⋮
N12	12/1/80	12/31/80	*P1*	\$3230

Note that the token N0 specifies a time interval which is the concatenation of the intervals specified by the tokens N1...N12.

The action of loading a gun can be described by a relation which may specify the attributes 'actor', 'motivation' and other attributes of interest. A tuple of that relation might assign 'Actor=John' and 'Motivation=Kill' to specify that John performed a gun loading action in order to kill someone rather than to participate in a shooting range.

5.2 Relation, Member and Attribute Predicates

Using the methodology described above, the world can be modeled by a relational database defined over a finite set of relation and attribute symbols. To describe this database using Simple TCLP we need only three predicates: "**Relation**", "**Member**" and "**Attribute**", and three basic types of variables: relation, attribute and token names. The predicate "Relation" has two arguments, $Relation(r, s)$ where r is a relation name and s is a variable which specifies a set of attribute names[3]; this predicate evaluates to *true* iff the scheme of the relation r subsumes s. The predicate "Member" has two arguments, $Member(t, r)$ where r is a relation name and t is a token name; this predicate evaluates to *true* iff the tuple t is a member of the relation r. The predicate "Attribute" has three arguments $Attribute(t, a, v)$ where t is a token name, a is an attribute name and v is a constant which is the value of a; this predicate evaluates to *true* iff the attribute a of tuple t evaluates to v.

The predicates Relation, Member and Attribute specify at most a single token argument and thus can be used in Simple TCLP programs.

5.3 Sample Queries

We discuss the three queries presented above. The first query requires to compute the set of relations that

[2]or a state, as in situation calculus.

[3]We abuse the notation: the domain of this variable (of a new sort) is the set of all the possible subsets of attributes.

hold *now* between John and Fred:

$$\{r_x \mid \exists tt_x\ Member(tt_x, r_x) \wedge Attribute(tt_x, Person1, John) \wedge Attribute(tt_x, Person2, Fred) \wedge During(now, tt_x)\}$$

$$= \{\ Owes, Hates, Threatens\ \}.$$

The answer to the second query is given by the set

$$\{(a_x, v_x) \mid \exists tt_x\ Member(tt_x, Gun) \wedge Attribute(tt_x, a_x, v_x) \wedge During(now, tt_x)\}$$

$$= \{\ (Loded, true),\ (Pointed, Fred)\ \}$$

namely we quantify over attribute (or fluent) names.

Finally, the answer to the third query is given by the set $\{tt \mid Member(tt, Load)\}$.

6 Conclusion

General Logic Programs were augmented with a class of temporal constraints supported by Temporal Constraint Networks. Performence benefits are obtained in the presence of disjunctions using known constraint propagation algorithms. A decidable fragment which admits a finite representation was identified. Some of the restrictions introduced by the decidable fragment are overcome by a syntactic structure which provides with the benefits of reification while avoiding the technical complications of reification. In particular, the proposed syntactic structure allows quantification on temporal occurrences and relation symbols.

References

[1] J. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26:832–843, 1983.

[2] F. Bacchus, J. Tenenberg, and J. Koomen. A non-reified temporal logic. In *Proc. KR'89*, pages 2–10, 1989.

[3] J. Chomicki. Depth-bounded bottom-up evaluation of logic programs. *Journal of Logic Programming*, 18:68–81, 1995.

[4] J. Chomicki. Finite representation of infinite query answers. *ACM Transaction on Database Systems*, (to appear), 1996.

[5] A. Galton. A critical examination of Allen's theory of action and time. *Artificial Intelligence*, 42:159–188, 1990.

[6] A. Galton. Reified temporal theories and how to unreify them. In *Proc. IJCAI'91*, pages 1177–1182, 1991.

[7] A. Gerevini, L. Schubert, and S. SCHAEFFER. Temporal reasoning in timegraph I-II. *SIGART bulletin*, 4(3):T1–T4, 1993.

[8] J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, (to appear), 1996.

[9] H. Kautz and P. Ladkin. Integrating metric and qualitative temporal reasoning. In *Proc. AAAI'91*, pages 241–246, 1991.

[10] M. Koubarakis. *Foundations of Temporal Constraint Databases.* PhD thesis, National Technical University of Athens, Athens, Greece, 1994.

[11] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 3, 1986.

[12] D. McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.

[13] I. Meiri. Combining qualitative and quantitative constraints in temporal reasoning. In *Proc. AAAI'91*, 1991.

[14] R.Dechter, I.Meiri, and J.Pearl. Temporal constraint networks. In *Proc. KR'89*, pages 83–93, 1989.

[15] E. Schwalb and R. Dechter. Processing disjunctions in temporal constraint networks. *Artificial Intelligence*, (to appear) 1997.

[16] Y. Shoham. *Reasoning about Change: Time and Causation from the Standpoint of Artificial Intelligence.* The MIT Press, 1988.

[17] P. van Beek. Reasoning about qualitative temporal information. *Artificial Intelligence*, 58:297–326, 1992.

[18] P. van Hentenrick. *Constraint Satisfaction in Logic Programming.* MIT Press, 1989.

[19] L. Vila and H. Reichgelt. The token reification approach to temporal reasoning. *Artificial Intelligence*, (to appear) 1996.