A TEMPORAL EXTENSION OF PROLOG

TOMAS HRYCEJ

▷ Temporal Prolog, a temporal logic extension of PROLOG, is presented. The primary criterion for the model selection has been its natural embedment into the logic programming paradigm. Under strong efficiency constraints, a first-order "reified" logic has been taken as a basis for the implementation. Allen's temporal constraint algorithm has been extended for treatment of retractable constraints. Their embedment into Temporal Prolog can be viewed as an instance of the Constraint Logic Programming paradigm. An example inspired by K. Forbus's Qualitative Process Theory illustrates how qualitative simulation and related tasks can be formulated in Temporal Prolog in a transparent and declarative way. ◁

1. INTRODUCTION

Temporal aspects of world description are of crucial importance in AI areas that are concerned with representation and inferences about dynamically changing world, particularly planning (e.g., Allen and Koomen [5], Dean [11], McDermott [37], and Vere [52], qualitative simulation (e.g., Forbus [16]), cognitive modeling (e.g., Allen [3]), natural language semantics (e.g., Allen [4]), or databases (e.g., Dean [11] or Dean and McDermott [12]).

This is the reason why much attention has been paid to temporal aspects of world description in recent years. These efforts resulted in the implementation of several working temporal logic systems.

This paper presents Temporal Prolog, an extension of PROLOG capable of handling temporally referenced logical statements and temporal constraints. It is concerned with implementation aspects of both temporal constraints and temporal logic.

Temporal Prolog can be viewed as a synthesis of temporal logic and of the Constraint Logic Programming paradigm, in which temporal constraints are formulated.

Address correspondence to Tomas Hrycej, Daimler-Benz AG, Research Center Ulm, Wilhelm-Runge-Str. 11, D-7900, Ulm, Germany.

Received October 1987; accepted March 1992.

Since Temporal Prolog is a programming language level tool, efficiency issues are of crucial importance. Some of its features have been introduced from pragmatic viewpoints, rather than from viewpoints of logical purity and completeness.

Primary objectives of Temporal Prolog development are listed in Section 2. In Section 3, several alternative approaches to temporal logic are overviewed and the logic of Temporal Prolog is developed.

Language elements extending PROLOG are given in Section 4.

The implementation of axioms of Section 3, in particular the constraining character of the rules and variable instantiation are the topics of Section 5.

Section 6 describes the implementation of the temporal constraint propagation algorithm and its interface to the logic, which can be viewed as an instance of the Constraint Logic Programming paradigm. The emphasis of this description is on its nonstandard features, in particular retractability and efficiency.

Section 7 briefly discusses applications of Temporal Prolog.

2. PRIMARY OBJECTIVES

Rather than develop a novel theoretical temporal logic system, the primary goals underlying the Temporal Prolog project are:

(1) Selecting a temporal logic model based on an easily comprehensible and intuitively clear concept of temporal referencing.
(2) Embedding the model in a natural way into the logic programming paradigm and programming style. Logic programming is characterized by (i) a declarative formulation of statements (facts and rules) in predicate formalism and (ii) solving problems by posing declarative queries which are resolved automatically by an interpreter. Our temporal reasoner should be integrated into this model as seamlessly as possible. This objective is similar to that of the query system of temporal database of Dean and McDermott [12].
(3) Formulating efficient inference algorithms. For a system incorporated into a common PROLOG interpreter, this requirement can be formulated more concretely as, "there should be no backtracking alternatives for a single temporal situation." This requirement has a strong influence on the implementation of temporal logic axioms (Sections 3 and 5).
(4) Making the model easy to implement on top of the PROLOG interpreters available at present. This goal may sound more "commercial" than is meant. The basic idea is that the temporal logic should be a relatively autonomous system that can be implemented on top of an available PROLOG interpreter, so that the built-in PROLOG inference mechanism is left unmodified. The advantage of this approach is that efficient and bug-free commercial interpreters can be used even if they are not available in source form.

3. LOGIC OF TEMPORAL PROLOG

3.1. Existing Models

As stated above, a criterion for the choice of temporal-logic model has been that it provide an easily comprehensible and intuitively clear concept of temporal referencing. In this section, existing temporal models will be, without claiming completeness, briefly reviewed.

As stated by Turner [48], there are basically two alternative approaches to handling time by logic.

(1) First-order logic can be directly used to formulate statements with some of the propositional symbols corresponding to time positions (points or intervals), for example, *boiling*(*water*, t) meaning "water is boiling at interval t."
(2) First-order logic can be extended, for example, by modal operators, for treatment of temporal modes.

3.1.1. Modal Temporal Logics. The modal logic approach seems to be the one preferred by logicians. The classical temporal logic (e.g., Rescher and Urquhart [42]) uses four modal operators:

FA meaning "A is true at some future time;"

PA meaning "A was true at some past time;"

GA meaning "A will be true at all future times;" and

HA meaning "A has always been true in the past."

The semantics of the classical temporal logic is given by the function h: $\mathrm{T} \times \mathrm{A} \rightarrow \{1,0\}$ with T a nonempty set of ordered time points and A, the set of atomic sentences. The semantics of nonatomic sentences can be inferred by the following rules (see [48]):

$h(t, A \text{ and } B) = 1$ iff $h(t, A)$ and $h(t, B)$;

$h(t, \neg A) = 1$ iff $h(t, A) = 0$;

$h(t, \mathrm{FA}) = 1$ iff $h(\exists t)\ (t < t'$ and $h(t', A) = 1)$;

$h(t, \mathrm{PA}) = 1$ iff $h(\exists t)\ (t > t'$ and $h(t', A) = 1)$.

Unfortunately, until now, no efficient theorem-proving algorithm has been found for arbitrary modal logics. However, there are algorithms for some restricted classes of temporal logics.

One of them is the resolution procedure of Abadi and Manna [1] for the Propositional Temporal Logic (PTL) of Manna and Pnueli [35]. The procedure has been generalized by Abadi and Manna [2] for the First-Order Temporal Logic (FTL).

PTL as well as FTL use the following modal operators:

$\bigcirc\ u$ meaning "u is true in the next state;"

$\Box\ u$ meaning "u is always true (from now on);"

$\Diamond\ u$ meaning "u is eventually true;"

uUv meaning "u is true until v is true;" and

uPv meaning "u precedes v."

The resolution algorithm for FTL extends the resolution for PTL by skolemization and unification rules. While the PTL-resolution is shown to be complete, it is not the case for FTL (for more details, see [1, 2]).

PTL has been used for specification and synthesis of communicating processes (see Manna and Wolper [36]). Another operational system for parallel program modeling is the language TEMPURA of Hale and Moszkowski [18].

3.1.2. Logics Using First-Order Predicate Calculus. The first of the approaches quoted at the beginning of this section uses first-order predicate calculus to capture temporal aspects. Logical statements are "indexed" by the time at which they are true. There are several different views of this model:

- First, they can be viewed as first-order logics in which each n-ary predicate is extended to an $(n + 1)$-ary prediate, the $(n + 1)$st argument being the time in which the predicate is supposed to be true. For example, *boiling*(*water*) would be extended to *boiling*(*water*, t) (see above).
- The second view is that of "reified logics" (see Shoham [45] or Reichgelt [41]). This is expressed by the predicate $HOLDS(P, t)$ with P an "object-level" predicate formula and t a time point or interval in which P is true.
- An interesting view is proposed by Reichgelt [41]. He sees the reified logics as a first-order formalization of the semantics of the modal temporal logic. This becomes clear if we compare the predicate $HOLDS(P, t)$ (and Allen's axioms of the next section) with the above definition of the semantic function $h(t, P)$.

The reified view seems to be the most flexible and the most powerful one. Its semantics has been studied, e.g., by Reichgelt [41] and Haugh [19].

The most important AI-oriented examples of this paradigm are the theoretical models of Allen [3, 4], and McDermott [37]. Since some parts of Allen's model have been adopted by Temporal Prolog, it is described in more detail in Section 3.2. The differences between both are listed in Section 3.3. The way Allen's temporal constraint propagation algorithm has been extended for Temporal Prolog is given in Section 6.

The implementations of models in this group differ in the formalism for constraints between the time positions (points or intervals). While Allen's model defines elementary relations directly on intervals, other systems of functionality comparable with that of Temporal Prolog (in particular the Temporal Database of Dean and McDermott [12] and Temporal Unification of PROMPT of Penberthy [40]) use relations between interval endpoints. Interval constraints are superior to endpoint constraints in expressiveness. For example, the disjointness of two intervals (crucial for the negation definition in Temporal Prolog) cannot be expressed in the endpoint formalism. (For other aspects of comparison of Temporal Prolog and Temporal Database, see Section 5.3.)

There have also been several attempts to formalize inferences on nonconvex intervals, i.e., intervals with "gaps," like "every Monday," etc. (see, Ladkin [32, 33] or Leban et al. [34]). While such models seem to be useful if used with an extensive metric information, like calendar dates, their applicability to qualitative temporal relations is probably limited.

For AI tools like Temporal Prolog, first-order-based models have the following advantages if compared with the modal logic models:

(1) They can use efficient first-order theorem provers (Goal 3 of Section 2).
(2) They can be implemented on top of PROLOG (Goal 2 of Section 2).

(3) For AI applications, knowledge representation is of crucial importance. For example, in planning domain, actions are typically represented by preconditions, which must be satisfied in some interval P, and influences, which are effective in one or more intervals I_j. The action itself takes place in an interval A. The durations of intervals P, I_j, and A are frequently unknown, and there are some constraints on relations between intervals P, I_j, and A, like that P must precede (or contain) A, etc. It is not obvious how such descriptions of actions can be translated to modal logics like PTL.

Reichgelt [41] points out that modal logics are more expressive in some cases. In particular, they are capable of capturing "changing ontology," that is, the fact that some objects may exist only at certain times. This problem can be solved by making explicit statements about the temporal extent of the existence of a particular object, like *HOLDS(human(john), johns_lifetime)* saying that john exists during a certain interval, his lifetime. This is a certain analogy to DENOTES-clauses of Reichgelt's model TR. The qualitative simulator of Section 7 makes extensive use of this representation.

For these reasons, the first-order, nonmodal approach has been adopted for Temporal Prolog.

3.2. *Allen's Temporal Logic*

Allen [4] has formulated several axioms for an interval-based temporal logic (notation slightly modified, "IN(S,T)" meaning "S is a subinterval of T"):

Axiom H.2: $\text{HOLDS}(P, T) \leftrightarrow (\forall S.\text{IN}(S, T) \rightarrow (\exists R.\text{IN}(R, S) \,\&\, \text{HOLDS}(P, R)))$.

Axiom H.3: $\text{HOLDS}(P \,\&\, Q, T) \leftrightarrow \text{HOLDS}(P, T) \,\&\, \text{HOLDS}(Q, T)$.

Axiom H.4: $\text{HOLDS}(\neg P, T) \leftrightarrow (\forall S.\text{IN}(S, T) \rightarrow \neg \text{HOLDS}(P, S))$.

For disjunction to have the property $\text{HOLDS}(P \vee Q, T) \leftrightarrow \text{HOLDS}(\neg(\neg P \,\&\, \neg Q), T)$, it is defined by

Axiom H.7: $\text{HOLDS}(P \vee Q, T) \leftrightarrow$
$(\forall S.\text{IN}(S, T) \rightarrow (\exists R.\text{IN}(R, S) \,\&\, (\text{HOLDS}(P, R) \vee \text{HOLDS}(Q, R))))$.

These axioms correspond to the following informal principles:

- If A holds in an interval S, it also holds in any subinterval T of S.
- If both A and B hold in T, their conjunction also holds in T.
- Negation of A holds in T, if there is no subinterval of T in which A holds.
- Disjunction is defined as a negation of conjunction of negations.

Intervals are in temporal relationships with other intervals. These temporal relations are formulated in the well-known formalism, which is only briefly summarized in this section. For more details on the underlying concepts, see [3].

There are 13 distinct elementary relations between two intervals: ⟨(before),⟩ (after), *m* (meets), *mi* (met by), *o* (overlaps), *oi* (overlapped by), *d* (during), *di* (contains), *s* (starts), *si*(started by), *f* (finishes), *fi* (finished by) and = (equals). These elementary relations ar exhaustive and exclusive, i.e., exactly one of them describes a fully specified qualitative relation between two intervals.

However, the information available about an interval relation may be incomplete. So we may only know that the exact relation is an element of a set of elementary relations. For example, we know that the relation between the intervals I and J is "during" or "starts" or "finishes" or "equal" (not knowing which of these four elementary relations is actually valid), which circumscribes the commonsense expression "interval I is contained in interval J." Such an ambiguous relationship can be represented by the set $\{d, s, f, =\}$, or generally by a subset of the set of 13 elementary relations. Special cases are (1) an unrestricted relation, which would be represented by the whole set of all 13 elementary relations, and (2) a completely specified relation, which is represented by a set containing a single elementary relation. The exhaustiveness of the set of 13 elementary relations implies that an empty relation is a logical contradiction.

Allen [3] proposes a constraint propagation algorithm which uses the "transitivity table," a set of $13^2 = 169$ inference rules of the type "if the relation between I and J is R and the relation between J and K is S, then the relation between I and K is constrained to T," I, J and K being intervals, and R, S and T interval relations represented in the above formalism. If a relationship between two intervals is constrained, all consequences for relations between other intervals (i.e., a transitive closure) are calculated.

The above temporal constraint model can be, after certain modifications (see Section 6), adopted by Temporal Prolog. Unfortunately, it is not the case for the above temporal axioms. While their underlying principles are intuitively clear, the above formalism brings about formidable implementation problems. The axiom for negation, and consequently, also the disjunction definition make use of quantifiers over intervals. Allen gives no details about implementation and reasoning or proof mechanisms in his logics. An implementation seems to make reasoning about intervals on an abstract, predicate level necessary, which is probably much more difficult to implement efficiently than a simple propositional model working only with concrete, instantiated intervals.

This is the reason why an alternative axiom set will be searched for.

3.3. *The Temporal-Prolog Approach*

The goal of this section is to formulate a set of axioms with intervals as conceptual primitives. These axioms in their final form are given in Section 3.3.2. Since it is easier to formulate the intuitions behind these axioms in a time-point-oriented model, a simple logic based on time points (and intervals as sets of time points) will be first presented in Section 3.3.1. The results of the time-point analysis will be incorporated into the axioms of Section 3.3.2. Then, in the rest of this paper, time intervals will be considered as primitives.

3.3.1. Time-Point Logic. As mentioned in Section 3.1, there are several alternative views of the first-order treatment of temporal informations. The notation of the "reified" view is probably the most natural and will be adopted throughout this paper. Let us assert $holds(A, t)$ if formula A (or an "object level" predicate A) is true in the time point t. Let us first treat the formulas A of $holds(A, t)$ as propositional formulas. The extension to predicates will be addressed in Section 3.3.2.

The following axioms for logical connectives are introduced (the signs &, V and ¬, if appearing in the first argument of *holds*, are to be interpreted as functional symbols—infix for & and V—rather than connectives):

$$\text{Axiom P.1: holds(A, t) \& holds(B, t) = holds(A \& B, t).} \tag{1}$$

$$\text{Axiom P.2: holds(A, t)} \vee \text{holds(B, t) = holds(A} \vee \text{B, t).} \tag{2}$$

$$\text{Axiom P.3: } \neg\,\text{holds(A, t) = holds(}\neg\text{A, t).} \tag{3}$$

If a formula A is true in each element t of a set of time points T, we can write

$$\underset{t\in T}{\&}\ \text{holds}(A,t) \tag{4}$$

The following relations can be easily inferred for expressions of the type (4). Since $a \& b \rightarrow a$, we can write

$$\Big(\underset{t\in T}{\&}\ \text{holds}(A,t)\Big)\&S\subset T\rightarrow\Big(\underset{t\in S}{\&}\ \text{holds}(A,t)\Big). \tag{5}$$

From (1) and (2), $a \& b \rightarrow a$ and $a \rightarrow a \vee b$, we get

$$\Big(\underset{t\in T}{\&}\ \text{holds}(A,t)\Big)\&\Big(\underset{t\in T}{\&}\ \text{holds}(B,t)\Big)\rightarrow\Big(\underset{t\in T}{\&}\ \text{holds}(A\&B,t)\Big) \tag{6}$$

$$\Big(\underset{t\in T}{\&}\ \text{holds}(A,t)\Big)\vee\Big(\underset{t\in T}{\&}\ \text{holds}(B,t)\Big)\rightarrow\Big(\underset{t\in T}{\&}\ \text{holds}(A\vee B,t)\Big). \tag{7}$$

$$\Big(\underset{r\in R}{\&}\ \text{holds}(A,r)\Big)\&\Big(\underset{s\in S}{\&}\ \text{holds}(B,s)\Big)\rightarrow\Big(\underset{t\in R\cap S}{\&}\ \text{holds}(A\&B,t)\Big) \tag{8}$$

$$\Big(\underset{r\in R}{\&}\ \text{holds}(A,r)\Big)\&\Big(\underset{s\in S}{\&}\ \text{holds}(B,s)\Big)\rightarrow\Big(\underset{t\in R\cup S}{\&}\ \text{holds}(A\vee B,t)\Big). \tag{9}$$

From (3) and the relation $a \& \neg a \rightarrow$ *false*, we get

$$\Big(\underset{r\in R}{\&}\ \text{holds}(A,r)\Big)\&\Big(\underset{s\in S}{\&}\ \text{holds}(\neg A,s)\Big)\rightarrow R\cap S=\{\ \}. \tag{10}$$

Relations (8) and (9) can be seen as constructive definitions of conjunction and disjunction, respectively, on expressions of type (4). Since the members of R ∩ S and R ∪ S are, for given sets R and S, the only time-points for which A&B, or A ∨ B, respectively, can be inferred, these relations can be seen as complete definitions of conjunction and disjunction, respectively. The relation (10) is a (nonconstructive) definition of negation. For a complete construction definition of negation. It would be necessary to construct a set which is the complement of the union R(A) of all sets in which A holds:

$$\Big(\underset{r\in R(A)}{\&}\ \text{holds}(A,r)\Big)\&S(A)=\{t|t\notin R(A)\}\rightarrow\Big(\underset{t\in S(A)}{\&}\ \text{holds}(\neg A,t)\Big). \tag{11}$$

To make a definition of intervals as sets of time points possible, the following properties of time points are postulated:

(1) Time points are totally ordered.
(2) Time points are discrete, with some sufficient density. (This property is motivated by some difficulties concerning interval endpoints if time points are dense—see Allen [3].)
(3) The set of all time points is infinite.

We can now define intervals in terms of time points. For reasons given in Section 3.1, we consider only convex intervals.

Definition 3.1. An interval is a convex set of time points, i.e., if $r, s \in T$, then for all t such that

$r < t < s$, $t \in T$, too.

Since time points are totally ordered and only convex intervals are considered, we can define the endpoints of an interval T.

Definition 3.2. The beginning of interval T is the time point $b(T) \in T$ such that $b(T) \leq T$ for all $t \in T$.

Definition 3.3. The end of interval T is the time point $e(T) \in T$ such that $e(T) \geq t$ for all $t \in T$.

The next step is to formulate relations defining logical connectives for expressions of the type (4), i.e., for statements holding for a whole interval.

Since we are only concerned with qualitative temporal relations, i.e., information about partial ordering, overlapping or containment of intervals, we will never exactly know which time points are elements of which interval. This has an important implication: we will not be able to "compute" intersections or unions of intervals exactly. The only thing we can do is to impose some appropriate constraints on the relations between the intervals and to query these relations. So our approach will be that of minimal commitment: we shall try to construct a (symbolic) interval and make only as many commitments about its relations to other intervals as necessary to guarantee the inference (e.g., inferences like (5) to (10)).

We shall now investigate how the logical connectives conjunction, disjunction and negation can be treated in such a minimal commitment manner.

An important requirement on the axioms is efficient implementation (Goal 3, Section 2). It can be formulated more technically in the following way. For a given set of intervals and temporal relations, the proofs should be possible without additional backtracking, i.e., only one PROLOG clause should match (if only one would match for the nontemporal part of the problem). Otherwise, a combinatorial explosion caused by temporal axioms would take place even in those cases in which the problem itself is of linear complexity (see the discussion of conjunction treatment below).

CONJUNCTION. There are two candidates for the treatment of conjunction:

(1) the relation (8) and
(2) the relation

$$\left(\mathop{\&}_{r\in R} \text{holds}(A,r)\right) \& \left(\mathop{\&}_{s\in S} \text{holds}(B,s)\right) \& T \subset R \& T \subset S \rightarrow \left(\mathop{\&}_{t\in T} \text{holds}(A\&B,t)\right) \qquad (12)$$

which results from (5) and (6).

The advantage of the first formulation is that it constructively describes the complete interval in which the conjunction A&B holds, i.e., the intersection of R and S.

If the ordering of intervals, i.e., the ordering of their endpoints, is fully specified, the intersection can be easily computed. Since the endpoints of interval $T = R \cap S$

TABLE 1. Alternative orderings for conjunction.

Relation b(R):b(S)	Relation e(R):e(S)	Relation R:S	b(T) equal to	e(T) equal to	Relation T:R	Relation T:S
$\leq$	$\geq$	{di, si, fi, =}	b(S)	e(S)	{d, s, f, =}	{=}
$\leq$	$<$	{o, s}	b(S)	e(R)	{s, =}	{f}
$>$	$\geq$	{oi, f}	b(R)	e(S)	{f, =}	{s}
$>$	$<$	{d}	b(R)	e(R)	{=}	{d}

result from the endpoints of R and S by the relations

$$b(T) = \max(b(R), b(S)) \qquad e(T) = \min(e(R), e(S)), \tag{13}$$

the intersection is defined by the relations T: R and T: S of Table 1.

Unfortunately, if the ordering of R and S is not fully specified, backtracking over some or all of the four alternatives of Table 1 would be necessary. For example, if the relation between R and S were R—{o, oi} → S (R overlaps S or is overlapped by S), we would have to split the case into two subcases: (1) R—{o} → S with constraints T—{f} → R and T—{s} → S; and (2) R—{oi} → S with constraints T—{f} → S and T—{s} → R.

The definition by (12) does not suffer from this problem. For any relation between R and S that does not exclude any kind of overlapping, an interval T such that it is subinterval of both R and S can be defined in a single clause by constraining both relations to {d, s, f, =}. Keeping in mind that (12) can be inferred from (6) and (5), it is (6) which has been adopted as a conjunction axiom.

Disjunction. Like for conjunction, there are two candidates for the treatment of disjunction:

(1) the relation (9) and
(2) the relation (7).

A more constructive circumscription of (7) is

$$\left[\left(\underset{r\in R}{\&}\text{holds}(A,r)\right) \vee \left(\underset{s\in S}{\&}\text{holds}(B,s)\right)\right] \& (T = S \vee T = R)$$
$$\rightarrow \left(\underset{t\in T}{\&}\text{holds}(A \vee B,t)\right). \tag{14}$$

So the second formulation would obviously make backtracking over T = R ∨ T = S necessary. However, even a nontemporal disjunction A ∨ B causes backtracking in PROLOG.

Paradoxically, the first formulation is, at first glance, a nonbacktracking one (although there is always backtracking for a nontemporal disjunction). However, backtracking may arise within the interval union definition for ambiguous interval relations (like R—{o, oi} → S above).

Like for conjunction, the endpoints of interval T = R ∪ S result from the endpoints of R and S by the relations

$$b(T) = \min(b(R), b(S)) \qquad e(T) = \max(e(R), e(S)). \tag{15}$$

The cases corresponding to different orderings of endpoints of R and S are listed in Table 2.

TABLE 2. Alternative orderings for disjunction.

Relation b(R):b(S)	Relation e(R):e(S)	Relation R:S	b(T) equal to	e(T) equal to	Relation T:R	Relation T:S
$\leq$	$\geq$	{di, si, fi, =}	b(S)	e(S)	{=}	{di, si, fi, =}
$\leq$	$<$	{o, s}	b(S)	e(R)	{si}	{fi, =}
$>$	$\geq$	{oi, f}	b(R)	e(S)	{fi}	{si, =}
$>$	$<$	{d}	b(R)	e(R)	{di}	{=}

So backtracking over one to four alternatives may occur (while the disjunction definition according to (14) would always backtrack over exactly two alternatives).

The superiority of the first formulation is obvious from Figure 1. Suppose A holds in AT, B holds in BT and C holds in CT with AT overlapping BT, AT overlapping CT and CT overlapping BT. The expression (A $\vee$ B)&C should intuitively hold in CT. This cannot be inferred using the second formulation.

So the first formulation will be used as disjunction axiom. On the other side, there are some problems concerning variable instantiation (see Section 5.2). This is why the first formulation may still be useful in some cases.

NEGATION. The relation (10) is sufficient for the treatment of negation in the following sense. If an interval S is constructed and it is asserted to be disjoint (i.e., to have an empty intersection) with all intervals in which A holds, then it is guaranteed that the negation of A holds in S.

There is a serious problem concerning completeness. The set of points in which a negation holds is generally nonconvex. Even if A holds in a single convex interval R, the negation of A holds (1) in the convex set of points before R and (2) the convex set of points after R. If A holds in multiple convex intervals R_i, there are correspondingly more convex intervals for the negation. Their number depends not only on the number of intervals R_i but also on their relations, i.e., on their overlapping. The nonconvex set S(A) of (11) can be written as a union

$$S(A) = \bigcup_j S_j \qquad (16)$$

with S_j convex intervals such that there is no convex interval U in which $\neg$A holds such that S_j is a proper subset of U.

This situation is illustrated on Figure 2. If A holds in convex intervals R_1, R_2 and R_3, $\neg$A holds in convex intervals S_1, S_2 and S_3.

The best solution seems to be to define a symbolic interval S which is not committed to any of the convex intervals S_j, but can be further constrained to be equal to one of them.

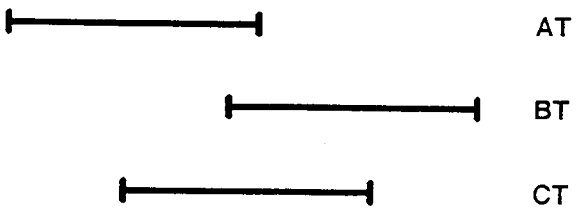

FIGURE 1. Overlapping intervals for the expression $(A \vee B)\&C$.

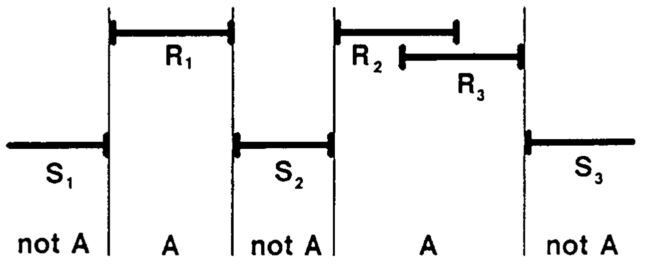

FIGURE 2. Intervals in which *A* or *not A* holds.

The assertion for intervals R and S to be disjoint can be decomposed into two cases:

$b(R) \geq e(S)$, i.e., $R—\{>, mi\} \rightarrow S$

and

$$b(S) \geq e(R), \text{ i.e., } R—\{<, m\} \rightarrow S. \tag{17}$$

Equalities are covering the case in which S fills a gap one of whose boundaries is defined by an endpoint of R, like S_2 of Figure 2 filling a gap, whose left boundary is defined by R_1.

Since the ordering may be incompletely specified, the disjointness is expressed by constraining the relation between R and S to $\{<, >, m, mi\}$. This solution is unproblematic from the backtracking viewpoint.

NOTE. Because of discreteness of time points, the relation "R meets S", i.e., $R—\{m\} \rightarrow S$, means that the latest point of R is the immediate neighbor of the earliest point of S.

3.3.2. Interval-Based Axioms of Temporal Prolog. Let us now formulate the axioms for time intervals. The predicate HOLDS(P, T) (written in capitals to distinguish it from its time-point counterpart) means "the statement (e.g., a Prolog clause) P holds (i.e., is true) in interval T." Since temporally unlimited statements coexist with those with temporal scope in Temporal Prolog, they are denoted by HOLDS(P), meaning P holds "universally," without limitation to a temporal interval.

Axiom I.1: HOLDS(A, S) & subinterval(T, S) $\rightarrow$ HOLDS(A, T). If A holds in an interval S, it also holds in any subinterval T of S.

Axiom I.2: HOLDS(A) $\rightarrow$ ($\forall$T) HOLDS(A, T). If A holds without temporal limitation, it also holds in any time interval.

Axiom I.3: HOLDS(A, T) & HOLDS(B, T) $\rightarrow$ HOLDS(A & B, T). If both A and B hold in T, their conjunction also holds in T.

Axiom I.4: HOLDS(A, U) & HOLDS(B, V) & union(U, V, T) $\rightarrow$ HOLDS(A $\vee$ B, T). A weaker alternative to Axiom I.4 (see Section 3.3.1) is the following Axiom I.4a.

Axiom I.4a: HOLDS(A, T) $\vee$ HOLDS(B, T) $\rightarrow$ HOLDS(A $\vee$ B, T). If at least one of A and B holds in T, their disjunction also holds in T.

Axiom I.5: HOLDS(A, S) & HOLDS($\neg$A, T) $\rightarrow$ disjoint(S, T). If A holds in S and (not A) holds in T, then S and T are disjoint intervals.

In the interval algebra, the relation subinterval(R, S) is expressed by R—{d, s, f, =} → S and the disjointness of R and S by R—{<, >, m, mi} → S. Interval union is constructed as a disjunction of the four alternatives of Table 2.

Axiom I.5 is related to Dean and McDermott's contradiction handling (Section 4.2 of [12]). The advantage of the present, interval-oriented approach is that disjointness of two intervals can be simply asserted (by {<, >, m, mi} relation)—even in case there is no ordering between the intervals or their endpoints—while the point-oriented approach of [12] requires a commitment to a certain ordering. Comparing the above axioms with those of Allen (Section 3.2), we can observe that Axioms I.1 and I.3 have direct counterparts in Allen's system. On the other side, there are the following differences:

(1) Negation is defined in terms of disjoint intervals, i.e., intervals in which A and negation of A hold are disjoint.
(2) Disjunction is defined "symmetrically" to conjunction.
(3) All axioms have the form of implications rather than equivalences, so that their transformation into the Horn-clause form is more straightforward. For example, the validity of a conjunction can be inferred from the validity of its arguments, but not inversely. The latter inference is never needed since the Horn-clause form does not permit assertions of the form A&B—they would be partitioned into two clauses A and B, but then the proof of the latter inference would be trivial.

With the present axiom set, the objective formulated in Section 3.2—avoiding quantifiers over intervals—has been achieved.

COMPLETENESS. In a first-order formulation of temporal logic, completeness is given by the completeness of the resolution algorithm (or its PROLOG implementation).

However, we have to keep in mind that this completeness concept is relative to the first-order formulations of (1) temporal logic axioms and (2) temporal constraint propagation. For example, if the axioms fail to define the interval for a conjunction of two formulas soundly and completely (e.g., with regard to the above time-point model), a correct resolution will certainly not be able to make up for this. So, some concept of completeness with regard to temporal logic axioms and temporal constraints must be defined.

The most straightforward completeness definition would be by relations (8), (9) and (11). This criterion would guarantee constructing maximal intervals for all logical connectives. Unfortunately, our committment to convex intervals and to an efficient implementation has led to axioms which do not satisfy this strong definition. However, a slightly weaker completeness criterion is satisfied. This criterion is given by the following definition.

Definition 3.4. An axiom set is a complete least-commitment definition of logical connectives, if for each set CL of clauses HOLDS(A, S) and HOLDS(A) and a query HOLDS(B, TV) with B—a formula consisting of propositional symbols and connectives &, $\vee$ and $\neg$, and TV—an interval variable TV can be consistently instantiated to a (constructed) convex interval T with the following properties:

(1) HOLDS(B, T) is sound in the sense of time-point axioms (1) through (3) and

(2) relations of T to other intervals are, for each complete interval endpoint ordering, supersets of the relations given by complete definitions, i.e., by intersection for conjunction (8), union for disjuntion (9) and complement (11) decomposed into maximal convex intervals by (16) for negation.

This definition substitutes the formualtion of maximal intervals (given, e.g., by the intersection of R and S in (8)) by the requirements of soundness and "potential maximality," i.e., interval relations which can be, for each total ordering of interval endpoints, further constrained so that the interval is maximal. For example, for the first case of Table 3.1, the relation of T to both R and S are $\{d, s, f, =\}$ (by Axioms I.1 and I.3) while the maximal interval T would have the relation $\{d, s, f, =\}$ to R and $\{=\}$ to S. The relation potentially allows T to be smaller than maximal, but the maximal case, $\{=\}$, is contained in $\{d, s, f, =\}$.

We can make the following, more formal statement about the above set of Axioms I.1 to I.5.

Proposition 3.1. Axioms I.1 to I.5 are a complete least-commitment definition of logical connectives.

PROOF. Axiom I.2 can be viewed as a transformation of predicate HOLDS(A) into predicate HOLDS(A, Inf), *Inf* being an "infinite" interval such that each other interval is a subinterval of *Inf*. So we will consider only predicate HOLDS(A, T). □

For each formula B such that HOLDS(B, T) (or HOLDS(B)) is directly contained in the set CL of Definition 3.4, the completeness is immediately proved.

CONJUNCTION. For each formula B = C&D, we get, by Axioms I.1 and I.3

$$\text{HOLDS}(C, R) \,\&\, \text{HOLDS}(D, S) \,\&$$
$$\&\, \text{subinterval}(T, R) \,\&\, \text{subinterval}(T, S) \rightarrow \text{HOLDS}(C\&D, T). \qquad (18)$$

Since the newly constructed interval T is a subinterval of both R and S, both C and D hold in T. HOLDS(C&D, T), and thus also the soundness for conjunction, can then be proved by (6).

As can be seen from Table 1, for all total orderings of endpoints of R and S and corresponding relatons of T with R and S, the relation $\{d, s, f, =\}$ is a superset of all these relatons. So the axioms are a complete least-commitment definition of conjunction.

DISJUNCTION. The proof is trivial since the interval union of Axiom I.4 is a direct reformulation of the complete disjunction definition (9).

NEGATION. The soundness follows immediately from (10).

For each formula $B = \neg C$, the interval T is such that it is disjoint with any interval R_i in which C holds. "Set complement" of all intervals in which C holds can be, according to (16), expressed as a union of convex intervals T_j. For a given endpoint ordering, each such interval T_j is related to each interval S_i by exactly one of the relations $<, >, m, mi$. T of Axiom I.5 stands for any of the intervals T_j. Axiom I.5 guarantees the disjointness of T with any of the intervals S_i, by constraining this relation to $\{<, >, m, mi\}$, which is a superset of each of the above relations. So Axiom I.5 is a complete least-commitment definition of negation. □

If Axiom I.4a were substituted for Axiom I.4, the axiom set would not be a complete least-commitment definition. However, the "potentially maximal" interval T would be decomposed into multiple subintervals, generated by backtracking. The union of these parts would be the interval T itself.

The second aspect of temporal completeness, the completeness of temporal-constraint propagation, is addressed in Section 6.1.

Treatment of Predicates. So far, we have investigated how logical connectives &, $\vee$ and $\neg$ can be correctly treated. The extension to predicates is straightforward. Formulas A of HOLDS(A,T) are allowed to contain uninstantiated variables and Skolem functions, and the correct resolution is performed by the unification procedure of PROLOG in exactly the same way as for "timeless" predicates. This will be clear from the implementation of Axioms I.1 to I.5, given in Section 5.2. Some nontrivial questions concerning variable unification are also discussed in Section 5.2.

On the other side, the first argument of HOLDS must be instantiated. In other words, no quantification over predicate formulas A is allowed. The case of universal quantification over T, i.e., HOLDS(A,T) with T uninstantiated, corresponds to HOLDS(A).

4. LANGUAGE EXTENSION

There are two types of language elements in addition to those of standard PROLOG: (1) temporal references and (2) temporal constraints.

Temporal References. If a statement S (a fact or a rule) holds exactly during the time interval T, S *in* T (T being instantiated to a symbolic interval identifier) is asserted.

For queries, predicates P *dur* T and P *mkdur* T are used. (For exact definitions, see Section 5.) Their semantics is "can it be inferred that predicate P holds in interval T?" In contrast to the predicate P *in* T, these queries encompass also the cases in which P holds in a superinterval of T or even without temporal limitation.

All three predicates correspond to the predicate HOLDS of previous sections: P *in* T for assertions, P *dur* T and P *mkdur* T for queries.

Temporal Constraints. Temporal constraints are formulated directly in Allen's interval formalism (see Section 3.2).

A temporal constraint is asserted by the backtrackable built-in predicate *constrain_rel(T, S, R)*. (The details of the backtrackable algorithm are given in Section 6.) By this predicate, the relation between intervals T and S is constrained to R (the set R is expressed in the PROLOG list notation, all three parameters must be fully instantiated). On backtracking, this constraint, including all its consequences, is retracted. The predicate fails if contradiction (an empty relation between some two intervals) results.

Actual interval relations can be queried by the predicate *get_rel*(T, S, R). It instantiates the variable R to the relation between (instantiated) intervals T and S.

NOTE. If necessary, qualitative relations could be extended by additional quantitative relations without modification of the logic rules given below. Such an extension is, e.g., the Extended Temporal Prolog of Hrycej [27].

5. IMPLEMENTATION OF AXIOMS

This section presents the implementation of the axioms of Section 3.3.2 in PROLOG. Before this, it is useful to explain the concept of constraining and nonconstraining rules.

5.1. Constraining Rules

As stated in the previous section, there are two query predicates: P *dur* T and P *mkdur* T. They represent two different query "philosophies:" nonconstraining and constraining queries. The exact semantics of P *dur* T is "*Can it be inferred, given the existing temporal relations, that P holds during T*?" while that of P *mkdur* T is "*Can the existing temporal relations be consistently constrained so that P holds during T*?"

Although the first definition seems to better match the theorem-proving character of PROLOG, it suffers from some deficiencies:

(1) It makes frequently too strong requirements on the existing relations if some statement is to be proved. In particular, a negation of a statement holds only in an interval which happens to have been explicitly asserted to be disjoint with the interval in which the statement holds—a rather uncommon case in practice.

(2) Answering a query, it is often necessary to construct a new interval and assert some constraints on its relations to other intervals. For example, if a conjunction (A, B) *mkdur* T is to be proved, an internal T is constructed which is set to appropriate relations to intervals R and S in which A and B hold, respectively. It is difficult to define the relations between R and T on the one side and S and T on the other side such that the relation between R and S remains untouched after constraint propagation.

(3) With the nonconstraining approach, what is not provable is not proved to be false (even under closed-world assumption) because a relation between two symbolic intervals is allowed to be a set (not a single element) of several of 13 elementary relations. The consequence of this is that, e.g., if A holds in T and the relation between S and T is unspecified (i.e., any elementary relation is possible). A cannot be proven either to hold in S or not to hold in S.

(4) Since ambiguities in temporal relations (i.e., relations consisting of a set of multiple elementary relations) often represent underconstrained situations rather than real ambiguities, we are frequently interested in possible relations, rather than provable ones. For example, we may be in search for additional time constraints to reach some goal. This is obviously the case in planning. Constructing a plan, it is important to know whether an action is consistent with other actions. In other cases, we may wish to generate potentially possible situations, e.g., for qualitative simulation. We get such

possible situations by constraining off some of elementary time relations. The constraining query formulation may then better match our intentions.

We can argue in the following way that even the constraining approach is conform with the theorem-proving character of PROLOG. We can consider a statement as provable if existing time relations can be constrained to their own subsets so that the statement can be proved. These subsets are, in fact, instantiations of a more general (or less determined) time relation. For example, relation {d, s, f} could be instantiated to {d, s}, {d, f}, {s, f}, {d}, {s}, {f}. Such a procedure would correspond to the way solutions are generated when interpreting a logic program: querying *human*(*X*), we may get the answer "yes" with instantiation X = john (while the answer may be "no" if X had to remain uninstantiated).

Since a relation can contain up to 13 elementary relations, it is obviously computationally nonsensical to generate blindly all such subsets. A good alternative is to generate only some of them in a goal-driven manner. For example, if we wish to prove a conjunction of A and B (A in S, B in T), we can try to constrain the relation between S and T to make the construction of a common subinterval i(S, T) of both intervals possible. If this additional constraint does not lead to a contradiction, "A&B in the interval i(S, T)" is proved.

Although both the nonconstraining and the constraining approaches have been implemented in Temporal Prolog (by predicates *dur* and *mkdur*, respectively), only the latter is described in this paper. For implementation details of nonconstraining rules, see Hrycej [25].

5.2. Implementation

Because of the implementary character of this paper, logical rules will be stated directly in PROLOG notation. Variables are denoted by uppercase, constants and functors by lowercase. Temporal predicates *in*, *dur* and *mkdur* are defined as infix operators.

CONJUNCTION. A conjunction of A and B holds in interval W (with identifier instantiated to i(U, V)) if W is a subinterval of both intervals U and V in which A and B, respectively, hold (Axiom I.3). This is ensured by the predicate *constrain_to_intersect* below, or more precisely, by its last clause. The first three clauses are treating trivial subcases of this. The first clauses refers to the case of equality between U and V, while the second and the third clauses apply to the case of U being a subinterval of V, or vice versa. The predicate *subset*(*R*, *S*) is true if the list R represents a subset of the list S.

```
(A,B) mkdur W:-A mkdur U,B mkdur V,constrain_to_intersect(U,V,W).
constrain_to_intersect(U,U,U).
constrain_to_intersect(U,V,U):-U\=V, get_rel(U,V,R),
    subset(R,[d,s,f,=]).
constrain_to_intersect(U,V,V):-U\=V, get_rel(U,V,R), R\=[=],
    subset(R,[di,si,fi,=]).
constrain_to_intersect(U,V,W):-
  U\=V,
  get_rel(U,V,R),
    not subset(R, [d, s, f, =]),
    not subset(R, [di, si, fi, =]),
    W = i(U, V),
    constrain_rel(W, U, [d, s, f, =]),
    constrain_rel(W, V, [d, s, f, =]).
```

Example 5.1. Suppose the following clauses are given:

```
a in r.
b in s.
```

Then, the query (a, b) *mkdur* T makes T instantiate to $i(r, s)$ with relations i(r, s)—{d, s, f, =} → r and i(r, s)—{d, s, f, =} → s. By constraint propagation, the relation between r and s is constrained to r—{o, oi, d, di, s, si, f, fi, =} → s, which is the most general formulation of overlapping in the Allen's interval formalism.

If the relation between s and t were a priori constrained to disjointness, i.e., to r —{<, >, m, mi} → s, (a, b) *mkdur* T would fail.

If the above clauses were modified to

```
a in r.
b.
```

i.e., if b were "timeless," T would be instantiated simply to r.

DISJUNCTION. To implement Axiom I.4, a union interval of U and V denoted by u(U, V) has to be constructed. The clauses of *constrain_to_overlap* correspond to the different cases of overlapping (see Section 3.3.1 and Table 2).

```
(A; B) mkdur W:-A mkdur U, B mkdur V, constrain_to_overlap(U, V, W).
constrain_to_overlap(U, U, U).
constrain_to_overlap(U, V, U):-U \= V, constrain_rel(U, V, [di, si, fi, =]).
constrain_to_overlap(U, V, V):-U \= V, constrain_rel(U, V, [d]).
constrain_to_overlap(U, V, W):-
  U \= V,
  W = u(U, V),
  constrain_rel(U, V, [o, s]),
  constrain_rel(W, U, [si]),
  constrain_rel(W, V, [fi, =]).
constrain_to_overlap(U, V, W):-
  U \= V,
  W = u(U, V),
  constrain_rel(U, V, [oi, f]),
  constrain_rel(W, U, [fi]),
  constrain_rel(W, V, [si, =]).
```

Example 5.2. Suppose the following clauses are given:

```
a(x) in r.
b(x) in s.
b(y) in t.
```

Suppose further the relation between r and s is a priori constrained to, say, overlapping (r—{o} → s). Then, in the query $(a(X); b(X))$ *mkdur* T, the third clause of *constrain_to_overlap* will match and variable T will be instantiated to the constructed interval $u(r, s)$.

If the relation between r and s were not constrained at all before the query, the four last clauses of *constrain_to_overlap* would match, and four corresponding solutions would be generated.

Example 5.2 illustrates an important point concerning variable instantiation. For Axiom I.4 to be applicable, a common instantiation for both disjunction arguments must be found. This would not be satisfied, e.g., by $a(x)$ and $b(y)$. However, the disjunction may also refer to different instantiations. The above query $(a(X); b(X))$ *mkdur* T would then have to be encoded in the "pure" clausal form, i.e., by an additional predicate

```
c(X):-a(X).
c(X):-b(X).
```

The query $c(X)$ *mkdur* T would then result in two different instantiations: X = x and X = y.

So the disjunction by a set of clauses on the one side and by the operator ";" in the clause body on the other side, have different semantics if uninstantiated variables are present.

A less complete (in the sense of Section 3.3.2) but substantially simpler solution is to implement Axiom I.4a by the following two clauses.

```
(A;B) mkdur T:-A mkdur T.
(A;B) mkdur T:-B mkdur T.
```

NEGATION. To ensure that the nonconstructive statement formulated by Axiom I.5 is always satisfied, the constructed interval T in which (*not* A) holds is asserted to be disjoint with all intervals S in which A holds.

This is trivial as long as all variables of A are instantiated. The case of one or more uninstantiated variables in A is more difficult. Since

$$(\forall x)\neg a(x) = \neg((\exists x)a(x)), \qquad (19)$$

the interval T must be disjoint with all intervals in which any instantiation of A holds. However, the variables of A must remain uninstantiated after the negation call, as can be seen from

$$(\forall x)(\neg a(x) \& b(x)) = \neg((\exists x)a(x)) \& (\forall x)b(x), \qquad (20)$$

quantifier inversion takes place only for negative conjuncts.

The easiest way to implement this is by widespread built-in predicate *findall*(X, Y, Z) which finds all instantiations of variable X (here the interval S) in predicate call Y (here A *mkdur* S) and instantiates Z with the list of all such instantiations, X remains uninstantiated after the call of *findall*, which is consistent with our requirements. Unfortunately, *findall* is not backtrackable. So, the constraints caused by the constraining query A *mkdur* S would not be retracted if backtracking occurred. This makes the use of the predicate *backtrack_point* necessary, which retracts all constraints asserted after it if backtracking takes place.

```
(not A) mkdur T:-
  T = n(NL),
  backtrack_point,
  findall(S, A mkdur S, NL),
  make_disjoint(NL, T).

make_disjoint([S|L], T):-constrain_rel(S, T, [ < , > , m, mi]), make_disjoint(L, T).
make_disjoint([ ], _).
```

Example 5.3. Suppose the following clauses are given:

```
a(x) in r.
a(y) in s.
```

Then, the query $(not\ a(X))\ mkdur\ T$ makes T instantiate to n([r, s]), with relations n([r, s])—{ < , > , m, mi} → r and n([r, s])—{ < , > , m, mi} → s.

The sequential unification algorithm of PROLOG imposes a constraint on the formulation of clauses. Suppose the query $(not\ a(X), b(X))\ mkdur\ T$ is to be evaluated. If $b(X)$ with uninstantiated X holds in some interval, this can be written as (5.2.2). If $b(X)$ holds only with X instantiated to say k, the query corresponds directly to

$$(\neg a(k) \& b(k)). \tag{21}$$

and not to

$$\neg((\exists x) a(x)) \& b(k), \tag{22}$$

which would be the case when evaluating the query $(not\ a(X), b(X))\ mkdur\ T$ by the above implementation of Axiom I.5.

This makes clear that the $(not\ A)$ has to be evaluated with variable instantiations consistent with those of other conjuncts. Since PROLOG interpreters instantiate sequentially, in the left-to-right order, a (not fully instantiated) expression of the type $(not\ A)$ must be placed after (i.e., to the right of) all conjuncts which may bind some of its variables. So the above query has to be reformulated as $(b(X),\ not\ a(X))\ mkdur\ T$.

NOTE. Like PROLOG, Temporal Prolog admits only positive predicates, i.e., $(not\ A)$ cannot be explicitly asserted. Instead of being directly asserted, the validity of a negative statement is inferred. The negation of Temporal Prolog is a logical extrapolation of PROLOG negation. While the latter is defined (in most commercial PROLOG systems) as $(not\ A)$ is true if A cannot be proved ("negation by failure," which is a form of closed-world assumption—see Hogger [21]), $(not\ A)$ in Temporal Prolog is true when A cannot be proved (i.e., at the time in which A cannot be proved).

RESOLUTION. In addition to the treatment of logical connectives, we have to define how resolution is to be performed for temporal statements. Substituting a Horn clause H ← B for C and its body B for D in (18), we get

$$\begin{aligned}&\mathrm{HOLDS}(\mathrm{H} \leftarrow \mathrm{B}, \mathrm{R}) \,\&\, \mathrm{HOLDS}(\mathrm{B}, \mathrm{S}) \,\& \\ &\quad \&\, \mathrm{subinterval}(\mathrm{T}, \mathrm{R}) \,\&\, \mathrm{subinterval}(\mathrm{T}, \mathrm{S}) \rightarrow \mathrm{HOLDS}(\mathrm{H}, \mathrm{T}).\end{aligned} \tag{23}$$

This relation can also be viewed as a "temporal modus ponens."

So the resolution of a predicate call is reduced to the resolution of the body of a clause matching the call.

The predicate *clause_mkdur* retrieves clauses in the above form H ← B. Its four clauses perform this, respectively, for (1) built-in predicates (which must be declared by the predicate *built_in*); (2) "timeless" predicates of the type HOLDS(A), using the widespread built-in predicate *clause* which instantiates H to the head and B to the body of a clause; (3) temporal facts (i.e., bodyless clauses); and (4) general temporal clauses.

```
true mkdur T.
X mkdur W:-clause_mkdur(X, Body, U), Body mkdur V,
constrain_to_intersect(U, V, W).

clause_mkdur(H, true, T):-built_in(H), call(H).
clause_mkdur(H, B, T):-clause(H, B).
clause_mkdur(H, true, T):-H in T.
clause_mkdur(H, B, T):-(H:-B) in T.
```

Example 5.4. Suppose the following clauses are given:

```
(h(X):-b(X)) in r.
b(x) in s.
```

Then, the query *h(X) mkdur T* would result in the following instantiations: X = x and T = i(r, s).

5.3. Comparison with the Temporal Database of Dean and McDermott

The temporal database system TMM of Dean and McDermott [12] uses an ingenious system of protections and reason maintenance for modeling persistence. A statement valid in interval [B, E] is considered as "true throughout an interval [BEG, END]" if (1) the upper bound of B is before BEG and (2) the upper bound of E is after END (while the lower bound of E may be before END). In other words, the beginning point of the interval has to be guaranteed while its end point has merely to be admissible (i.e., not inconsistent). The reason for this asymmetry seems to be in the planning domain for which the tool is intended. Typically, actions planned have effects whose beginning is known to be immediately after the action or simultaneous with the actions' beginning, while their termination (i.e., persistence) cannot be assessed. So the effects persist "until something changes." The constraining rules of this section make such an asymmetry obsolete: interval relations are constrained to meet query conditions and are immediately tested for contradiction. This is true for both beginnings and ends of intervals (a differentiated treatment of beginnings and ends would be difficult since intervals are conceptual primitives). So, for example, an action may be implicitly "shifted" (by constraining its relation to other time tokens) if it is necessary to resolve a contradiction or to satisfy a further constraint.

Additionally, imposing constraints simultaneously with the proofs of queries makes reason maintenance of persistances also obsolete. If a query can be answered positively by introducing the constraint-terminating persistence of a statement (i.e., constraining the endpoint of the corresponding interval), this constraint is definitely asserted. Should this persistence constraint later cause a contradiction, the whole problem-solving branch is retracted, including the constraints caused by the query. So the overall consistence is simply ensured by the backtracking mechanism of PROLOG.

The same effect is obtained in TMM by asserting a protection (instead of a constraint) for assumed persistence and management of protections by a reason-maintenance system. This complicated system seems to be useful for planning applications—it makes "patching" the plan possible if contradictions appear after an action has already been accepted. In planning, there are two levels of constraints: (1) those resulting from some planning decisions and (2) those resulting from the physical situation of the world. It is clear that the first group of constraints has, in a sense, a "lower priority" than the second—it can be manipulated by revising the decision, while the second cannot. However, outside the planning domain, there is no apparent reason to maintain such a two-level treatment. For example, in qualitative simulations (see Section 7), all constraints are one-level—a contradiction corresponds to a nonphysical solution, and it is backtracking (and not patching) which is the appropriate reaction to it. One more reason for adopting the simpler (and more robust) constraining approach in Temporal Prolog is that the programmer would have to be made responsible for distinction and appropriate treatment of both levels.

6. TEMPORAL-CONSTRAINT PROPAGATION

The interval algebra of J. Allen [3] can be formulated in logic (see [48]) by predicates of the type

$$\text{Meets}(i1,i2)\,\&\,\text{During}(i2,i3) \rightarrow (\text{Overlaps}(i1,i3) \vee \text{During}(i1,i3) \vee \text{Meets}(i1,i3)).$$

There are two reasons why such a formulation is difficult to implement directly.

(1) The rules of the above type are not Horn clauses.
(2) The backward-chaining character of PROLOG would make the computation of transitive closure (a forward-chaining procedure) very inefficient.

Since the most natural formulation of the interval algebra is by constraints, a promising possibility is an implementation in the paradigm Constraint Logic Programming (CLP—see Jaffar, Lassez and Maher [28]).

Constraint Logic Programming is a natural extrapolation of the fact that PROLOG unification can be seen as a constraint expressing equality between two predicate variables. This principles can be extended in a straightforward way to equality theories between numeric variables (see Goguen and Meseguer [17], Jaffar, Lassez and Maher [28], or van Emden and Yukawa [50]). Several systems have been developed for handling certain classes of constraints, like real-value linear constraints, in the CLP(R) system of Jaffar and Michaylov [29] or PROLOG III of Colmerauer [9], integer constraints, e.g., in the forward-checking procedure of van Hentenryck and Dincbas [51] or in the quantifier elimination of the TRILOGY language of Voda [55].

Temporal constraints of Temporal Prolog represent another such constraint class. The temporal constraint propagation algorithm is conceptually embedded into PROLOG in a similar way as how linear algebraic constraints are embedded in CLP language (see, Jaffar and Michaylov [29]) but, in contrast to CLP, it requires no modification of the PROLOG inference machine— it is implemented on top of PROLOG (Goal 4, Section 2) and communications via predicate *constrain_rel* (see Section 4) with a constraint module implementing the procedural formulation of Allen's algorithm [3]. For more about this interface, see Section 6.3.

The implementation of an efficient and backtrackable temporal-constraint propagation algorithm is the topic of this section.

6.1. Basic Algorithm

The basic algorithm for a complete constraint network is due to Allen [3]. In a slightly modified notation, it consists of the following procedures:

Algorithm 6.1.

```
    constrain_rel(i, j, NewRel)
      update_rel(i, j, NewRel);
      process_queue(constraint, constraint_propagation);
with
    update_rel(r1, r2, NewRel)
      H ← NewRel;
      if not H = R(r1, r2) then add_to_queue(constraint, r1, r2);
      R(r1, r2) ← H;
    constraint_propagation(i, j)
      For each k such that triangle(i, j, k) do
      begin
        update_rel(k, j, transitions(R(k, i), R(i, j)));
        update_rel(i, k, transitions(R(i, j), R(j, k)));
      end
```

Auxiliary functions are defined as follows:

- The function *transitions*($R1, R2$) corresponds to Allen's *constraints*($R1, R2$) —it is an implementation of the transitivity table.
- The function *process_queue*(*queue_id*, *queue_proc*) calls for each item (i, j) in the queue *queue_id* the procedure *queue_proc*(i, j).
- The functionality of *add_to_queue* is obvious.

NOTE. For extensive applications, space and computing time requirements growing with the square of the number of intervals may be unacceptable (quadratic growth for storage, cubic for computing time). Fortunately, the relation network contains redundant relations which can be easily reconstructed using transitivity of certain elementary relations, for example precedence ($<$, m-relations) or containment (d-, s-, f-relations). For extensive discussion of transitivity in temporal relation, see Hrycej [22, 23, 24].

COMPLETENESS OF CONSTRAINT PROPAGATION. As stated by Vilain and Kautz [54], Allen's polynomial time-constraint propagation algorithm is incomplete while the complete algorithm is NP-hard. The most dangerous possible consequence is that some contradiction will not be discovered, i.e., that there is no solution satisfying all temporal constraints. So, for example, an inconsistent plan can be generated.

Valdes-Perez [49] has proposed a complete algorithm based on intelligent backtracking. Its worst-case complexity is exponential, but, according to Valdes-Perez, substantial efficiency gain can be expected in typical cases.

However, even this algorithm is too slow for tools like Temporal Prolog which are intended to process up to hundreds of intervals and constraints. Even worse, the algorithm for Temporal Prolog must be made backtrackable (see the next section), which is an additional computational burden. Under such conditions, Allen's polynomial time algorithm is "slow enough."

If completeness is an important requirement for a given application (although dropping this requirement is one of the pragmatic proposals made by both Vilain and Kautz and Valdes-Perez and has been accepted in probably the most current applications of interval algebra), a possible solution is to constrain the relation language to some subset such that the polynomial-time algorithm is complete.

The broadest of such subsets described until now is the one corresponding to a time-point algebra, whose completeness has been shown by Vilain and Kautz [54]. The time-point algebra is an analogy of the interval algebra except for its primitive being a time point and the 13 elementary interval relations being substituted by three point relations: $\{<, >, =\}$. Like in the interval algebra, the relation between two points is a nonempty subset of this set. There are eight such subsets: $\{<\}$, $\{>\}$, $\{=\}$, $\{<, =\}$, $\{>, =\}$, $\{<, >\}$, $\{<, >, =\}$. (A class of relations closely related to this are the "generalized windows" of Rit [43].) The interval algebra constrained to interval relations that can be circumscribed by a conjunction of such point relations between interval endpoints is complete since its transitivity operations correspond exactly to the point-based transitivity operations of Vilain and Kautz.

An example of such a relation is the relation A—{d, s, f, =} → B which can be expressed as

$$\text{begin}(A)\text{—}\{>, =\} \rightarrow \text{begin}(B)\,\&\,\text{end}(A)\text{—}\{<, =\} \rightarrow \text{end}(B).$$

On the other side, the relation A—{m, s} → B does not belong to this class—its endpoint circumscription is disjunctive:

$$\begin{aligned}&\text{end}(A)\text{—}\{=\} \rightarrow \text{begin}(B)\\ &\quad \vee (\mathit{begin}(A)\text{—}\{=\} \rightarrow \mathit{begin}(B)\,\&\,\mathit{end}(A)\text{—}\{<\} \rightarrow \mathit{end}(B)).\end{aligned}$$

Let us denote the class of interval relations that can be circumscribed by a conjunction of endpoint relations as "endpoint-conjunctive."

Temporal Prolog axioms (see Section 5) make use of the following relations: {d, s, f, =}, expressing the subinterval relation, {s, =} and {f, =}, defining an interval union, and $\{<, >, \text{m}, \text{mi}\}$, expressing interval disjointness. Except for disjointness, all these relations are endpoint-conjunctive. For a broad class of applications (like the qualitative simulation of Section 7), the class consisting of endpoint-conjunctive relations and disjointness is sufficient. On the other hand, omitting disjointness would be a substantial constraint, in particular, it would not be possible to express

- the negation according to Axiom I.5;
- the disjointness of intervals for contradictory facts;
- limited resources (like a single machine for performing multiple operations), etc.

The goal conflict between expressiveness, completeness and efficiency can be hardly be resolved. The only thing to be done is to point out that using negation and endpoint-disjunctive relations in Temporal Prolog may potentially lead to inconsistent solutions (while confining ourselves to endpoint-conjunctive relations would guarantee completeness).

However, it must be said that inconsistencies caused by disjointness alone (i.e., without other endpoint-disjunctive relations) are very scarce. The reason for this is that pairs of disjointness relations do not combine to any further constraint, i.e., A —$\{<, >, m, mi\} \rightarrow$ B & B—$\{<, >, m, mi\} \rightarrow$ C implies no constraint on the relation between A and C via the transitivity table. A four-node network of Figure 3 has been tested for all combinations of endpoint-disjunctive relations R1, R2, R3 and R4. Out of 160,000 possible combinations, a single one with R1 = R2 = R3 = R4 = $\{o, oi, d, di, s, si, f, fi, =\}$ has led to an (undetected) inconsistency. So for most real applications, the inconsistency risk is acceptable.

The reward for accepting reduced expressiveness or potential incompleteness are acceptable computing times (see Section 6.4). If the completeness has very high priority for some application, the algorithm of Valdes-Perez can be substituted for Allen's.

6.2. Nonmonotonic Extension of the Algorithm

The above temporal constraint algorithm is monotonic—it does not allow retraction of asserted constraints. Therefore, the predicate *constraint_rel* of Section 4 using this algorithm would not be backtrackable. A straightforward technique for solving this problem in a PROLOG implementation is to keep the whole network of temporal relations as an additional argument in all predicates (or, if the network is modified within the predicate, two arguments: an input network and an output

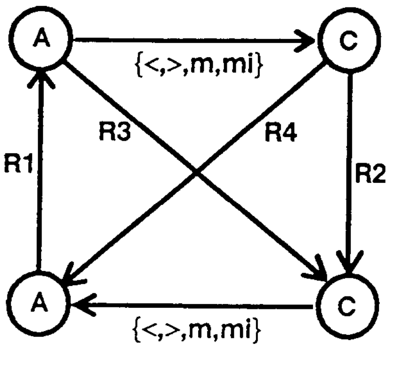

FIGURE 3. An interval network with two disjoint interval pairs.

network). However, this solution is extremely stack-intensive: in a toy example with only 10 intervals the 2 MByte stack overflowed! Thus, it is necessary to formulate an efficient nonmonotonic algorithm, which is the objective of this section.

Another common way to implement retractability of inferences is to label data in some way with their antecedents [15] or external assumptions [13] from which they have been inferred. To introduce nonmonotonicity into our temporal constraint system, we have to identify (1) elementary pieces of data and (2) justifications.

The way gone by Vilain [53] was to regard the whole transitivity table inference as elementary, so that potentially multiple (old and new) inferences for a single pair of intervals have to be stored. So in Vilain's model, whole transitivity-table-based inferences are explicitly stored for truth-maintenance operations. Such an representation is uneconomical in the sense that the number of such stored inferences per interval pair monotonically increases. The inference database grows at the same pace as the total number of constraints.

In this section, a more concise representation of dependencies is presented.

Justification Structure of Interval Relations. In our system, an alternative view of elementary datum is implemented. For each of the 13 elementary relations and each interval pair, the list of intervals, the "exclusion list," is stored, by way of which the elementary relation is excluded. For example, $R(I, J) = \{d, o\}$ $\&R(J, K) = \{mi\}$ constrains the relations between I and K to $\{d, di, fi, o\}$, i.e., it justifies the exclusion of <, >, oi, m, mi, s, si, f, =. So K is inserted into the exclusion list of each of the given elementary relations. The elementary relation with an empty exclusion list is regarded as possible. If a constraint is externally justified, the external constraint identifier enters the list instead of an interval identifier.

The elementary piece of data can be formulated as "exclusion of a single elementary relation between intervals I and J." For example, the semantics of I—[<, m] → J is "between the intervals I and J, the relation *before* or the relation *meets* are possible." In other words, "the relations *before* and *meets* are not excluded while all other relations are excluded." There may be several justifications for this elementary piece of data; if there are none, the relation is "possible."

Justifications are provided by the transitivity table, or better said by *transitions*. For example, I—[d, o] → J—[mi] → K results in I—[d, di, fi, o] → K, i.e., it justifies the exclusion of {<, >, m, mi, s, si, f, =}. In terms of our elementary pieces of data, the conjunctive exclusion of {<, >, di, oi, m, mi, s, si, f, fi, =} between I and J and of {<, >, d, di, o, oi, m, s, si, f, fi, =} between J and K justifies the exclusion of all elementary relations between I and K from {<, >, oi, m, mi, s, si, f, =}. This is the conjunctive form of justifications, typical for most truth-maintenance systems.

However, for external assumptions to be retractable, it is sufficient to store the interval identifier that justifies an exclusion. For example, if excluded relations between A and B and those between B and C exclude A *before* C, B is stored as a justification for the exclusion of *before* between A and C. This justification is revised after every modification of the relation between A and B. An elementary relation, the justification set of which is empty, is believed to be "possible."

The Algorithm. The basic algorithm of Section 6.1 has now only to be modified for handling exclusion lists. There are two basic system functions: constraint insertion and constraint retraction. Since the constraint propagation part of the algorithm is similar for both functions, an additional two-valued argument *Op* (with values *ins* and *del*) is introduced. For an external assumption (a set of one or more constraints) to be retractable, it must be given a unique identifier, which serves as an external justification (*Just*). Algorithm 6.1 undergoes the following modifications.

Algorithm 6.2.

```
    $update\_rel(ins, r1, r2, NewRel, Just)$
        $J(r1,r2) \leftarrow modify\_justification(J(r1,r2), NewRel, Just)$;
        $H \leftarrow possible\_rels(J(r1,r2))$;
        $if$ not $H = R(r1,r2)$ $then$ $add\_to\_queue(constraint, r1, r2)$;
        $R(r1,r2) \leftarrow H$;
    $update\_rel(del, r1, r2, NewRel, Just)$
        $J(r1,r2) \leftarrow modify\_justification(J(r1,r2), NewRel, Just)$;
        $H \leftarrow possible\_rels(J(r1,r2))$;
        $if$ not $H = R(r1,r2)$ $then$ $add\_to\_queue(constraint, r1, r2)$;
        $R(r1,r2) \leftarrow H$;
```

Additional *auxiliary functions*:

- The function *modify_justification*(*Justifications, NewCon, IdJust*) returns *Justifications* modified by the additional constraint *NewCon*. The exclusions of all elementary relations not contained in *NewCon* are additionally justified by the event identifier *IdJust*. If some justifications by this identifier are already present, they are substituted for the new ones.
- The function *possible_rels*(*Just*) returns the set of basic relations which are not excluded by *Just*, i.e., those with empty justification set.

SIMPLIFIED ALGORITHM FOR CHRONOLOGICAL BACKTRACKING. If we confine ourselves to chronological backtracking, the algorithm may be substantially simplified. Instead of an exclusion list for each elementary relation, it is sufficient to keep the identifier of the first external assumption that led to the exclusion of this elementary relation. For example, if the relation between I and J is $\{<, m\}$ and it has been constrained to $\{<\}$ after the introduction of external assumption *Ald*, identifer *Ald* is stored as a justification for exclusion of "m." This enables us to keep the whole information about interval relations for N intervals in a three-dimensional $N*N*13$-array, whose element $m(i, j, r)$ contains the chronologically first external assumption identifier which excludes the elementary relation r between intervals i and j. To retract an assumption, all matrix elements containing its identifier are zeroed. Chronological backtracking guarantees that assumptions are retracted in an order reverse to their assertion.

The space complexity of the algorithms of this section is $O(N^2)$, i.e., it is completely independent from the number of constraints. This is a substantial improvement over Vilain's algorithm whose complexity is proportional to the number of constraints.

6.3. *Interface between Temporal Logic and Temporal Constraints*

The communication between PROLOG and the constraint propagation system is materialized by two predicates: *constrain_rel*(I, J, R, A) and *retract_rel*(I, J, R, A) with I, J intervals, R the temporal relation between them and A a constraint identifier. The former predicate asserts a constraint and performs subsequent constraint propagation. The latter retracts a constraint including its formerly propagated consequences.

However, not the above pair of predicates, but a single backtrackable predicate is needed. This is done by the following clause (*gen_integer*(*Int*) generates successive integers):

```
constrain_rel(I,J,R):-gen_integer(A),(constrain_rel(I,J,R,A);

                                      retract_rel(I,J,R,A),fail).
```

During its first call, this clause generates the constraint identifier A and asserts the constraint. On backtracking, the clause is called the second time. The assumption identified by A is retracted and the whole clause fails.

An important advantage of this idea is its easy integrability into any PROLOG interpreter (Goal 4, Section 2). The expense for it is that no cut may be used. Otherwise, the second, retracting call of *constrain_rel* could not be guaranteed. In fact, a weaker condition is sufficient: if a cut has been used, no *constrain_rel* may be called between the neck symbol ":-" of the clause and the cut.

6.4. *Actual Implementation*

For efficiency reasons, the chronological backtracking algorithm of Section 6.2 has been implemented in C and linked into a commercially available PROLOG interpreter. Using an array representation instead of a relation database, the computing times have improved more than 1,000 times compared to the original implementation in PROLOG.

Although no systematic complexity investigation has been made so far, a feeling of the order of magnitude of computing times can be obtained from running the qualitative simulation example of Section 7 which imposes 64 constraints on 23 distinct intervals. On a PCS Cadmus 9900 UNIX-Workstation, using InterFace Prolog System, the complete simulation took 15.38 sec. Out of this time, 6.48 sec were spent by temporal constraint propagation, so computing the transitive closure for an additional constraint took 0.101 sec on average.

To show (in a very simplified manner) how the computing time may develop for larger problems, the intervals and temporal constraints of the qualitative simulation example have simply been duplicated, i.e., for each interval, a new interval (distinct from the original one) has been defined and all constraints have been imposed on the set of such new intervals. This procedure is at least partially realistic since the transitive closure is then computed over the set of both original and duplicated intervals, and it has the advantage that the constraint sets are directly comparable. By repeated duplication, examples of different sizes have been generated. The results are given in Table 3.

TABLE 3. Computing times of temporal-constraint propagation.

Number of Intervals	Number of Constraints	Computing Time for all Constraints (in seconds)	Computing Time per Constraint (in seconds)
23	64	6.48	0.101
46	128	18.34	0.143
92	256	59.52	0.233
184	512	227.80	0.445

7. APPLICATIONS

As stated in Section 1, the key applications of Temporal Prolog are problems concerning description of dynamic situations.

One such application is qualitative simulation, or "qualitative physics." Qualitative physics has its origin in the efforts to overcome the inability of present expert systems to reason about structure and function of technical devices, or, more exactly, to infer their behavior from their structure. Temporal relations cover some crucial aspects of qualitative reasoning:

- Causal relations incorporate the sequentiality of cause and effect.
- Complex preconditions for certain activities, like physical processes. Presume simultaneous occurrence of multiple elementary conditions.
- Persistence of object properties is constrained to the period of the object's existence, etc.

There have been several attempts to formulate general theoretical frameworks and working systems for qualitative reasoning about physical systems (see Davis [10], deKleer and Brown [14], Forbus [16], Kuipers [30, 31], Voss [56], Weld [57] and Williams [58, 59].

There is also a tradition of formulating qualitative descriptions in the logic programming paradigm, e.g., by Barrow [6], Clocksin [7], Heintze, Michaylov, and Stuckey [20], and Pashtan [39].

However, although all of these models are concerned with modeling behavior along a time line, few of them consider time explicitly.

This is exactly the advantage of using Temporal Prolog for problems of qualitative physics. As a demonstration of its capabilities, a simple qualitative simulator has been implemented. It has been inspired by the Qualitative Process Theory (QPT) of Forbus [16]. The most salient difference from the QPT is that *temporal relations have been made explicit.* Behavior is described by temporal relations between the states of individual physical quantities instead of by a sequence of global states. The conceptual units of behavior in this model are physical processes. For example, suppose four generic processes are defined: (1) heat flow from a heat source to a destination; (2) boiling if a fluid has reached its boiling temperature; (3) growing pressure if the amount of a gas in a closed can is growing; and (4) explosion after the pressure necessary for a closed can to burst has been attained. A generic process is described by the predicate *process*, e.g.,

```
process(heat_flow(D,S)):-
   (obj(S),heat_source(S),obj(D),D\=S,at (D,S),
     less(temp,[D],temp_s(S))) mkdur T,
   assert_infl(inc(temp,[D],+),T).
```

the expression of the second line expressing the precondition under which the process is launched and the third line performing an assertion of process influences, i.e., increasing the temperature of the heat-flow destination object. Domain-specific concepts and relations are described by clauses like

```
at(F,T):-obj(C),can(C),fluid(F),inside(F,C), under(T,C).
```

The initial state is given by

- object definitions (together with intervals in which objects exist, or "changing ontologies"—see Section 3.1), like

  ```
  obj(water) in ow.
  ```

- object properties;
- initial values of physical quantities and their increments, like

  ```
  val(temp,[water],between(t_melt,t_boil))  in tw.
  inc(temp,[water], =)                      in itw.
  ```

- initial temporal constraints, etc.

The simulation is performed by alternating two steps: (1) initializing as many processes as possible and (2) performing qualitative state changes, and it terminates if no qualitative state change can be made.

Suppose the initial state consists of the following objects: water in a closed can and a flame under the can. In the can, there is already a certain amount of steam, but the pressure has not yet reached the bursting point of the can. The temperature of the water is lower than the temperature of the flame, so that heat flow can take place.

With this initial state, the simulation results in process ordering according to Figure 4. The existence of the object "can" persists during the processes *heat_flow*, *boiling*, *growing_pressure*, and it is terminated by the process *explosion*. Information of this type can be acquired by the query predicate *get_rel*(I, J, R), I and J being intervals in which some processes are asserted to be active. We can also construct complete histories of parameter values by finding all clauses of the type *val(Quantity, Objects, Value) in T* and querying their order or their temporal relations to other histories.

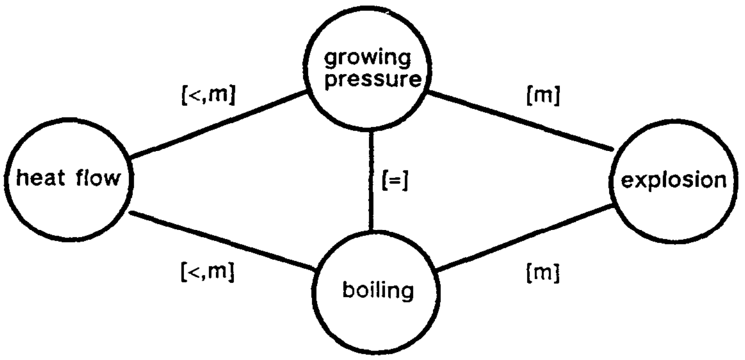

FIGURE 4. Ordering of processes.

A complete account of the qualitative simulator is given in [26]. For another application of Temporal Prolog for reasoning about processes, see Noekel [38]. A further class of problems with a strong temporal aspect is planning. While early planners like WARPLAN of Warren (described in [8]) required a commitment to a certain linear ordering, more recent systems, like NOAH (Sacerdoti [44]), NONLIN (Tate [46]), and O-PLAN (Tate [47]), DEVISER (Vere [52]), introduced partial ordering—some pairs of actions are ordered, others are not. Temporal Prolog (in a manner similar to the planner of Allen and Koomen [5]) allows even more general temporal relations, e.g., explicit disjointness, without further commitment to a certain ordering. So less backtracking is necessary, and planning becomes more efficient.

8. CONCLUSION

Temporal Prolog, a temporal extension of PROLOG, is a pragmatic implementation of a reified temporal logic. It provides natural and easily comprehensible means for constraining validity of general predicates, i.e., logical facts and rules, to a certain time interval. It conforms to logic programming concepts and programming style. Both its temporal logic inference engine and its temporal constraint propagator are reasonably efficient without substantial lost of expressive power in comparison, e.g., with Allen's theoretical model.

Two alternatives for implementing logic axioms are incorporated into Temporal Prolog: constraining rules and nonconstraining rules. Constraining queries are a simple and robust alternative to the TMM model of Dean and McDermott which is based on a planning-specific concept of protection and which makes reason maintenance necessary to ensure consistency. Both are obsolete in Temporal Prolog—consistency is ensured by the backtracking mechanism of PROLOG. The whole system can be easily implemented in any commercially available PROLOG interpreter. If an interface to a procedural language is available (which is the case for most commercial products), efficiency of temporal-constraint propagation can be substantially improved.

Key potential applications of Temporal Prolog seem to be in planning, qualitative physics and other applications characterized by temporally limited facts and explicit qualitative temporal constraints. Problems from these domains can be formulated in Temporal Prolog in a transparent and declarative way. Various temporal constraints, like nonoverlapping of contradictory states or persistence of process influences, can be directly represented. So Temporal Prolog provides an appropriate environment for the development of tools for planning and qualitative reasoning.

This research has been supported in part by the Bundesminister für Forschung und Technologie, grant No. ITW 8506 C9 (TEX-B research project).

I would like to thank Rachel Jones for carefully reading and correcting the text.

REFERENCES

1. Abadi, M., and Manna, Z., Nonclausal Temporal Deduction, in: R. Parikh (ed.), *Logic of Programs*, Springer-Verlag, 1985.

2. Abadi, M., and Manna, Z., A Timely Resolution, in: *Proceedings of the Symposium on Logic in Computer Science,* Cambridge, U.K., 1986.
3. Allen, J. F., Maintaining Knowledge about Temporal Intervals, *Commun. ACM* 26(11):832–843 (1983).
4. Allen, J. F., Towards a General Theory of Action and Time, *Artif. Intell.* 23:123–154 (1984).
5. Allen, J. F., and Koomen, J. A., Planning Using a Temporal World Model, in: *Proceedings of the Eighth International Conference on Artificial Intelligence*, Karlsruhe, Germany, 1983, pp. 741–747.
6. Barrow, H. G., VERIFY: A Program for Proving Correctness of Digital Hardware Designs, *Artif. Intell.* 24:437–491 (1984).
7. Clocksin, W. F., Logic Programming and Digital Circuit Analysis, *J. Logic Programming* 4:59–82 (1987).
8. Coelho, H., Cotta, J. C., and Pereira, L. M., *How to Solve It with PROLOG,* Laboratorio Nacional de Engenharia Civil, Lisbon, Portugal, 1980.
9. Colmerauer, A., Introduction to PROLOG III, in: *Proceedings of the 4th Annual ESPRIT Conference*, Brussels, Belgium, 1987.
10. Davis, R., Diagnostic Reasoning Based on Structure and Behavior, *Artif. Intell.* 24:347–410 (1984).
11. Dean, T. L., Large-Scale Temporal Data Bases for Planning in Complex Domains, in: *Proceedings of the Tenth International Conference on Artificial Intelligence*, Milano, Italy, 1987, pp. 860–866.
12. Dean, T. L., McDermott, D. V., Temporal Data Base Management, *Artif. Intell.* 32:1–55 (1987).
13. deKleer, J., An Assumption-Based TMS, *Artif. Intell.* 28:127–162 (1986).
14. deKleer, J., and Brown, J. S., A Qualitative Physics Based on Confluences, *Artif. Intell.* 24:7–83 (1984).
15. Doyle, J., A Truth Maintenance System, *Artif. Intell.* 12:231–272 (1979).
16. Forbus, K. D., Qualitative Process Theory, *Artif. Intell.* 24:85–168 (1984).
17. Goguen, J. A., Meseguer, J., Equality, Types, Modules, and (Why Not?) Generics for Logic Programming, *J. Logic Programming* 2:179–210 (1984).
18. Hale, R., and Moszkowski, B., Parallel Programming in Temporal Logic, in: *Proceedings Conference on Parallel Architectures and Languages Europe*, Eindhoven, Netherlands, 1987.
19. Haugh, B. A., Non-Standard Semantics for the Method of Temporal Arguments, in: *Proceedings of the Tenth International Conference on Artificial Intelligence*, Milano, Italy, 1987, pp. 449–455.
20. Heintze, N., Michaylov, S., and Stuckey, P., CLP(R) and Some Electrical Engineering Problems, in: J.-L. Lassez (ed.), *Logic Programming: Proceedings of the Fourth International Conference*, MIT Press, Cambridge, Mass., 1987.
21. Hogger, C. J., *Introduction to Logic Programming*, Academic Press, New York, 1984.
22. Hrycej, T., An Efficient Algorithm for Reasoning about Time Intervals, in: *Proceedings of the Conference "Expertensysteme '87,"* Nurnberg, Germany, 1987.
23. Hrycej, T., Transitivity in Relations between Time Intervals, in: Heyer, Krems, Goerz (eds.): *Wissensarten und ihre Darstellung*, Springer-Verlag, Berlin, 1988.
24. Hrycej, T., A Transitivity-Based Algorithm for Temporal Constraint Propagation, in: Früchtenicht, H. W. et al. (eds.), *Technische Expertensysteme: Wissensrepräsentation und Schlussfolgerungsverfahren*, R. Oldenbourg Verlag, 1988.
25. Hrycej, T., Temporal Prolog, in: *Proceedings of the European Conference on Artificial Intelligence '88*, Munich, Germany, 1988.
26. Hrycej, T., Qualitative Simulation with Temporal Prolog, Technical Report No. TEX-B-42, PCS GmbH, Munich, Germany, 1988.
27. Hrycej, T., Extended Temporal Prolog: Quantitative Temporal Constraints, submitted to the Eleventh International Conference on Artificial Intelligence, Detroit, Mich., 1989.

28. Jaffar, J., Lassez, J.-L., and Maher, M. J., A Theory of Complete Logic Programs with Equality, *J. Logic Programming* 3:211–223 (1984).
29. Jaffar, J., and Michaylov, S.,, Methodology and Implementation of a CLP system, in: J-L. Lassez (ed.), *Logic Programming: Proceedings of the Fourth International Conference*, MIT Press, Cambridge, Mass., 1987.
30. Kuipers, B., Commonsense Reasoning about Causality: Deriving Behaviour from Structure, *Artif. Intell.* 24:169–203 (1984).
31. Kuipers, B., The Limits of Qualitative Reasoning, in: *Proceedings of the Ninth International Conference on Artificial Intelligence*, Los Angeles, Calif., 1985.
32. Ladkin, P., Time Representation: A Taxonomy of Interval Relations, in: *Proceedings of the Sixth National Conference on Artificial Intelligence*, Philadelphia, Penn., 1986, pp. 367–371.
33. Ladkin, P., The Completeness of a Natural System for Reasoning with Time Intervals, in: *Proceedings of the Seventh National Conference on Artificial Intelligence*, Seattle, Wash., 1987, pp. 462–467.
34. Leban, B., McDonald, D. D., and Forster, D. R., A Representation for Collections of Temporal Intervals, in: *Proceedings of the Sixth National Conference on Artificial Intelligence*, Philadelphia, Penn., 1986, pp. 360–366.
35. Manna, Z., and Pnueli, A., Verification of Concurrent Programs: The Temporal Framework, in: R. S. Boyer, and J. S. Moore (eds.), *The Correctness Problem in Computer Science*, Academic Press, New York, 1982, pp. 215–273.
36. Manna, Z., and Wolper P., Synthesis of Communicating Processes from Temporal Logic Specifications, *ACM Trans. Programming Lang. Syst.* 6(1):68–93 (1984).
37. McDermott, D., A Temporal Logic for Reasoning about Processes and Plans, *Cognitive Science* 6(2):101–155 (1982).
38. Noekel, K., Temporal Matching: Recognizing Dynamic Situations from Discrete Measurements, submitted to the Eleventh International Joint Conference on Artificial Intelligence, Detroit, Mich., 1989.
39. Pashtan, A., A Prolog Implementation of an Instruction-Level Processor Simulator, *Software—Practice and Experience* 17(5):309–318 (1987).
40. Penberthy, J. S., Temporal Unification and the Temporal Partial Order, in: *Proceedings of the Fifth IEEE Conference on Artificial Intelligence Applications*, San Diego, Calif., 1987.
41. Reichgelt, H., Semantics for Reified Temporal Logic, in: J. Hallam, and C. Mellish (eds.), *Advances in Artificial Intelligence*, Wiley, 1987.
42. Rescher, J., and Urquhart, A., Temporal Logic, Springer-Verlag, 1971.
43. Rit, J-F., Propagating Temporal Constraints for Scheduling, in: *Proceedings of the Sixth National Conference on Artificial Intelligence*, Philadelphia, Penn., 1986, pp. 383–388.
44. Sacerdoti, E. S., *A Structure for Plans and Behaviour*, Elsevier North-Holland, New York, 1977.
45. Shoham, Y., Reified Temporal Logics: Semantical and Ontological Considerations, in: *Proceedings of the European Conference on Artificial Intelligence '86*, Brighton, U.K., 1986.
46. Tate, A., Generating Project Networks, in: *Proceedings of the Fifth International Conference on Artificial Intelligence*, 1977, pp. 888–893.
47. Tate, A., Goal Structure, Holding Periods and "Clouds," Technical Report No. AIAI-TR-17, University of Edinburgh, U.K., 1986.
48. Turner, R., *Logics for Artificial Intelligence*, Ellis Horwood, Chichester, 1984.
49. Valdes-Perez, R. E., The Satisfiability of Temporal Constraint Networks, in: *Proceedings of the Seventh National Conference on Artificial Intelligence*, Seattle, Wash., 1987, 256–260.
50. van Emden, M. H. and Yukawa, K., Logic Programming with Equations, *J. Logic Programming* 4:265–288 (1987).
51. van Hentenryck, P., and Dincbas, M., Forward Checking in Logic Programming, in: J.-L. Lassez (ed.), *Logic Programming: Proceedings of the Fourth International Conference*, MIT Press, Cambridge, Mass., 1987.

52. Vere, S. A., Planning in Time: Windows and Durations for Activities and Goals, *IEEE Trans. Pattern Analysis and Mach. Intell.* 5(3):246–267 (1983).
53. Vilain, M. B., A System for Reasoning about Time, in: *Proceeding of the Second National Conference on Artificial Intelligence*, Pittsburgh, Penn., 1982, pp. 197–201.
54. Vilain, M. B., and Kautz, H., Constraint Propagation Algorithms for Temporal Reasoning, in: *Proceedings of the Sixth National Conference on Artificial Intelligence*, Philadelphia, Penn., 1986, pp. 377–382.
55. Voda, P., TRILOGY System Manual, Complete Logic Systems, Inc., Vancouver, Canada, 1987.
56. Voss, H., Representing and Analyzing Time and Causality in HIQUAL models, Memo SEKI-85-07, Fachbereich Informatik, Universität Kaiserslautern, Germany, 1986.
57. Weld, D. S., The use of Aggregation in Causal Simulation, *Artif. Intell.* 30:1–34 (1986).
58. Williams, B. C., Qualitative Analysis of MOS Circuits, *Artif. Intell.* 24:281–340 (1984).
59. Williams, B. C., Doing Time: Putting Qualitative Reasoning on a Firmer Ground, in: *Proceedings of the Sixth National Conference on Artificial Intelligence*, Philadelphia, Penn., 1986.