Artificial Intelligence and Language Processing

David Waltz Editor

Maintaining Knowledge about Temporal Intervals

JAMES F. ALLEN *The University of Rochester*

James F. Allen's main interests are in artificial intelligence in particular natural language processing and the representation of knowledge.

Author's Present Address: James F. Allen, Computer Science Department, University of Rochester, Rochester, NY 14627.

The research described in this paper was supported in part by the National Science Foundation under Grants IST-80-12418 and IST-82-10564, and in part by the Office of Naval Research under Grant N00014-80-C-0197.

0001-0782/83/1100-0832 75¢

***ABSTRACT:** An interval-based temporal logic is introduced, together with a computationally effective reasoning algorithm based on constraint propagation. This system is notable in offering a delicate balance between expressive power and the efficiency of its deductive engine. A notion of reference intervals is introduced which captures the temporal hierarchy implicit in many domains, and which can be used to precisely control the amount of deduction performed automatically by the system. Examples are provided for a database containing historical data, a database used for modeling processes and process interaction, and a database for an interactive system where the present moment is continually being updated.*

1. INTRODUCTION

The problem of representing temporal knowledge and temporal reasoning arises in a wide range of disciplines, including computer science, philosophy, psychology, and linguistics. In computer science, it is a core problem of information systems, program verification, artificial intelligence, and other areas involving process modeling. (For a recent survey of work in temporal representation, see the special sections in the April 1982 issues of the ACM SIGART and SIGMOD Newsletters.)

Information systems, for example, must deal with the problem of outdated data. One approach to this is simply to delete outdated data; however, this eliminates the possibility of accessing any information except that which involves facts that are presently true. In order to consider queries such as, "Which employees worked for us last year and made over $15,000," we need to represent temporal information. In some applications, such as keeping medical records, the time course of events becomes a critical part of the data.

In artificial intelligence, models of problem solving require sophisticated world models that can capture change. In planning the activities of a robot, for instance, one must model the effects of the robot's actions on the world to ensure that a plan will be effective. In natural language processing, researchers are concerned with extracting and capturing temporal and tense information in sentences. This knowledge is necessary to be able to answer queries about the sentences later. Further progress in these areas requires more powerful representations of temporal knowledge than have previously been available.

This paper addresses the problem from the perspective of artificial intelligence. It describes a temporal representation that takes the notion of a *temporal interval* as primitive. It then describes a method of representing the relationships between temporal intervals in a hierarchical manner using constraint propagation techniques. By using *reference intervals*,

the amount of computation involved when adding a fact can be controlled in a predictable manner. This representation is designed explicitly to deal with the problem that much of our temporal knowledge is relative, and hence cannot be described by a date (or even a "fuzzy" date).

We start with a survey of current techniques for modeling time, and point out various problems that need to be addressed. After a discussion of the relative merits of interval-based systems versus point-based systems in Section 3, a simple interval-based deduction technique based on constraint propagation is introduced in Section 4. This scheme is then augmented in Section 5 with reference intervals, and examples in three different domains are presented. In the final sections of the paper, extensions to the basic system are proposed in some detail. These would extend the representation to include reasoning about the duration of intervals, reasoning about dates when they are available, and reasoning about the future given knowledge of what is true at the present.

The system as described in Section 5 has been implemented and is being used in a variety of research projects which are briefly described in Section 6. Of the extensions, the duration reasoner is fully implemented and incorporated into the system, whereas the date reasoner has been designed but not implemented.

2. BACKGROUND

Before we consider some previous approaches to temporal representation, let us summarize some important characteristics that are relevant to our work:

- The representation should allow significant imprecision. Much temporal knowledge is strictly relative (e.g., A is before B) and has little relation to absolute dates.
- The representation should allow uncertainty of information. Often, the exact relationship between two times is not known, but some contraints on how they could be related are known.
- The representation should allow one to vary the grain of reasoning. For example, when modeling knowledge of history, one may only need to consider time in terms of days, or even years. When modeling knowledge of computer design, one may need to consider times on the order of nanoseconds or less.
- The model should support *persistence*. It should facilitate default reasoning of the type, "If I parked my car in lot A this morning, it should still be there now," even though proof is not possible (the car may have been towed or stolen).

This does not exhaust all the issues, and others will come up as they become relevant. It provides us with a starting criteria, however, for examining previous approaches. Previous work can be divided roughly into four categories: state space approaches, date line systems, before/after chaining, and formal models.

State space approaches (e.g., [7, 17]) provide a crude sense of time that is useful in simple problem-solving tasks. A state is a description of the world (i.e., a database of facts) at an instantaneous point in time. Actions are modeled in such systems as functions mapping between states. For example, if an action occurs that causes P to become true and causes fact Q to be no longer true, its effect is simulated by simply adding fact P to the current state and deleting fact Q. If the previous states are retained, we have a representation of time as a series of databases describing the world in successive states. In general, however, it is too expensive to maintain all the previous states, so most systems only maintain the present state. While this technique is useful in some applications, it does not address many of the issues that concern us. Note that such systems do provide a notion of persistence, however. Once a fact is asserted, it remains true until it is explicitly deleted.

In datebase systems (e.g., [4, 5, 12, 13]), each fact is indexed by a *date*. A date is a representation of a time such that the temporal ordering between two dates can be computed by fairly simple operations. For example, we could use the integers as dates, and then temporal ordering could be computed using a simple numeric comparison. Of course, more complicated schemes based on calendar dates and times are typically more useful. Because of the nice computational properties, this is the approach of choice if one can assign dates for every event. Unfortunately, in the applications we are considering, this is not a valid assumption. Many events simply cannot be assigned a precise date. There are methods of generalizing this scheme to include ranges of dates in which the event must occur, but even this scheme cannot capture some relative temporal information. For instance, the fact that two events, A and B, did not happen at the same time cannot be represented using fuzzy dates for A and B. Either we must decide that A was before B, or B was before A, or we must assign date ranges that allow A and B to overlap. This problem becomes even more severe if we are dealing with time intervals rather than time points. We then need fuzzy date ranges for both ends of the interval plus a range for the minimum and maximum duration of the interval.

The next scheme is to represent temporal information using before/after chains. This approach allows us to capture relative temporal information quite directly. This technique has been used successfully in many systems (e.g., [4, 13]). As the amount of temporal information grows, however, it suffers from either difficult search problems (searching long chains) or space problems (if all possible relationships are precomputed). This problem can be alleviated somewhat by using a notion of reference intervals [13], which will be discussed in detail later. Note that a fact such as "events A and B are disjoint" cannot be captured in such systems unless disjunctions can be represented. The approach discussed in this paper can be viewed as an extension of this type of approach that overcomes many of its difficulties.

Finally, there is a wide range of work in formal models of time. The work in philosophy is excellently summarized in a textbook by Rescher and Urquhart [16]. Notable formal models in artificial intelligence include the situation calculus [14], which motivates much of the state space based work in problem solving, and the more recent work by McDermott [15]. In the situation calculus, knowledge is represented as a series of situations, each being a description of the world at an instantaneous point of time. Actions and events are functions from one situation to another. This theory is viable only in domains where only one event can occur at a time. Also, there is no concept of an event taking time; the transformation between the situations cannot be reasoned about or decomposed. The situation calculus has the reverse notion of persistence: a fact that is true at one instance needs to be explicitly reproven to be true at succeeding instants.

Most of the work in philosophy, and both the situation calculus and the work by McDermott, are essentially point-based theories. Time intervals can be constructed out of points, but points are the foundation of the reasoning system. This approach will be challenged in the upcoming section.

One other formal approach, currently under development, that is compatible with an interval-based temporal representa-

tion is found in the Naive Physics work of Hayes [10, 11]. He proposes the notion of a *history*, which is a contiguous block of space-time upon which reasoning can be organized. By viewing each temporal interval as one dimension of a history, this work can be seen as describing a reasoning mechanism for the temporal component of Naive Physics.

3. TIME POINTS VS. TIME INTERVALS

In English, we can refer to times as points or as intervals. Thus we can say the sentences:

We found the letter at twelve noon.
We found the letter yesterday.

In the first, "at twelve noon" appears to refer to a precise point in time at which the finding event occurred (or was occurring). In the second, "yesterday" refers to an interval in which the finding event occurred.

Of course, these two examples both refer to a date system where we are capable of some temporal precision. In general, though, the references to temporal relations in English are both implicit and vague. In particular, the majority of temporal references are implicitly introduced by tense and by the description of how events are related to other events. Thus we have

We found the letter while John was away.
We found the letter after we made the decision.

These sentences introduce temporal relations between the times (intervals) at which the events occurred. In the first sentence, the temporal connective "while" indicates that the time when the find event occurred is during the time when John was away. The tense indicates that John being away occurred in the past (i.e., before now).

Although some events appear to be instantaneous (e.g., one might argue that the event "finding the letter" is instantaneous), it also appears that such events could be decomposed if we examine them more closely. For example, the "finding the letter" might be composed of "looking at spot X where the letter was" and "realizing that it was the letter you were looking at." Similarly, we might further decompose the "realizing that it was the letter" into a series of inferences that the agent made. There seems to be a strong intuition that, given an event, we can always "turn up the magnification" and look at its structure. This has certainly been the experience so far in physics. Since the only times we consider will be times of events, it appears that we can always decompose times into subparts. Thus the formal notion of a time point, which would not be decomposable, is not useful. An informal notion of time points as very small intervals, however, can be useful and will be discussed later.

There are examples which provide counterintuitive results if we allow zero-width time points. For instance, consider the situation where a light is turned on. To describe the world changing we need to have an interval of time during which the light was off, followed by an interval during which it was on. The question arises as to whether these intervals are open or closed. If they are open, then there exists a time (point) between the two where the light is neither on nor off. Such a situation would provide serious semantic difficulties in a temporal logic. On the other hand, if intervals are closed, then there is a time point at which the light is both on and off. This presents even more semantic difficulties than the former case. One solution to this would be to adopt a convention that intervals are closed in their lower end and open on their upper end. The intervals could then meet as required, but each interval would have only one endpoint. The artificiality of this solution merely emphasizes that a model of time based on points on the real line does not correspond to our intuitive notion of time. As a consequence, we shall develop a representation that takes temporal intervals as primitive.

If we allowed time points, intervals could be represented by modeling their endpoints (e.g., [4]) as follows: Assuming a model consisting of a fully ordered set of points of time, an interval is an ordered pair of points with the first point less than the second. We then can define the relations in Figure 1 between intervals, assuming for any interval t, the lesser endpoint is denoted by t− and the greater by t+.

We could implement intervals with this approach, even given the above argument about time points, as long as we assume for an interval t that t− < t+, and each assertion made is in a form corresponding to one of the interval relations. There are reasons why this is still inconvenient, however. In particular, the representation is too uniform and does not facilitate structuring the knowledge in a way which is convenient for typical temporal reasoning tasks. To see this, consider the importance of the *during* relation. Temporal knowledge is often of the form

event E′ occurred during event E.

A key fact used in testing whether some condition P holds during an interval t is that if t is during an interval T, and P holds during T, then P holds during t. Thus *during* relationships can be used to define a hierarchy of intervals in which propositions can be "inherited."

Furthermore, such a *during* hierarchy allows reasoning processes to be localized so that irrelevant facts are never considered. For instance, if one is concerned with what is true "today," one need consider only those intervals that are *during* "today," or above "today" in the *during* hierarchy. If a fact is indexed by an interval wholly contained by an interval representing "yesterday," then it cannot affect what is true now. It is not clear how to take advantage of these properties using the point-based representation above.

4. MAINTAINING TEMPORAL RELATIONS

4.1. The Basic Algorithm

The inference technique described in this section is an attempt to characterize the inferences about time that appear to be made automatically or effortlessly during a dialogue, story comprehension, or simple problem-solving. Thus it should provide us with enough temporal reasoning to participate in these tasks. It does not, however, need to be able to account for arbitrarily complex chains of reasoning that could be done, say, when solving a puzzle involving time.

We saw above five relations that can hold between intervals. Further subdividing the *during* relation, however, pro-

Interval Relation	Equivalent Relations on Endpoints
t < s	t+ < s-
t = s	(t- = s-) & (t+ = s+)
t overlaps s	(t- < s-) & (t+ > s-) & (t+ < s+)
t meets s	t+ = s-
t during s	((t- > s-) & (t+ =< s+)) or ((t- >= s-) & (t+ < s+))

FIGURE 1. Interval Relation Defined by Endpoints.

vides a better computational model.[1] Considering the inverses of these relations, there are a total of thirteen ways in which an ordered pair of intervals can be related. These are shown in Figure 2.

Sometimes it is convenient to collapse the three *during* relations (d, s, f) into one relationship called *dur*, and the three *containment* relations (di, si, fi) into one relationship called *con*. After a quick inspection, it is easy to see that these thirteen relationships can be used to express any relationship that can hold between two intervals.

The relationships between intervals are maintained in a network where the nodes represent individual intervals. Each arc is labeled to indicate the possible relationship between the two intervals represented by its nodes. In cases where there is uncertainty about the relationship, all possible cases are entered on the arc. Note that since the thirteen possible relationships are mutually exclusive, there is no ambiguity in this notation. Figure 3 contains some examples of the notation. Throughout, let N_i be the node representing interval i. Notice that the third set of conditions describes disjoint intervals.

Throughout this paper, both the above notations will be used for the sake of readability. In general, if the arc asserts more than one possible relationship, the network form will be used, and in the case where only one relationship is possible, the relation form will be used.

For the present, we shall assume that the network always maintains complete information about how its intervals could be related. When a new interval relation is entered, all consequences are computed. This is done by computing the transitive closure of the temporal relations as follows: the new fact adds a constraint about how its two intervals could be related, which may in turn introduce new constraints between other intervals through the transitivity rules governing the temporal relationships. For instance, if the fact that i is *during* j is added, and j is *before* k, then it is inferred that *i* must be *before* k. This new fact is then added to the network in an identical fashion, possibly introducing further constraints on the relationship between other intervals. The transitivity relations are summarized in Figure 4.

The precise algorithm is as follows: assume for any temporal relation names r1 and r2 that T(r1, r2) is the entry in the transitivity table in Figure 4. Let R1 and R2 be arc labels, assume the usual set operations ($\cap$ for intersection, $\cup$ for union, $\subset$ for proper subset), and let ϵ be the empty set. Then *constraints (R1, R2)* is the transitivity function for lists of relation names (i.e., arc labels), and is defined by:

```
Constraints (R1, R2)
  C ← ε;
  For each r1 in R1
    For each r2 in R2
      C ← C ∪ T(r1, r2);
  Return C;
```

Assume we have a queue data structure named *ToDo* with the appropriate queue operations defined. For any two intervals i, j, let N(i, j) be the relations on the arc between i and j in the network, and let R(i, j) be the new relation between *i* and j to be added to the network. Then we have the following algorithm for updating the temporal network:

```
To Add R(i, j)
  Add ⟨i, j⟩ to queue ToDo;
  While ToDo is not empty do
begin
  Get next ⟨i, j⟩ from queue ToDo;
  N(i, j) ← R(i, j);
  For each node k such that Comparable(k, j) do
  begin
    R(k, j) ← N(k, j) ∩ Constraints(N(k, i), R(i, j))
    If R(k, i) ⊂ N(k, i)
      then add ⟨k, i⟩ to ToDo;
  end
  For each node k such that Comparable(i, k) do
  begin
    R(i, k) ← N(i, k) ∩ Constraints(R(i, j), N(j, k))
    If R(i, k) ⊂ N(k, i)
      then add ⟨i, k⟩ to ToDo;
  end
end;
```

We have used the predicate *Comparable(i, j)* above, which will be defined in Section 5. For the present, we can assume it always returns true for any pair of nodes.

Relation	Symbol	Symbol for Inverse	Pictoral Example
X *before* Y	<	>	XXX YYY
X *equal* Y	=	=	XXX YYY
X *meets* Y	m	mi	XXXYYY
X *overlaps* Y	o	oi	XXX YYY
X *during* Y	d	di	XXX YYYYYY
X starts Y	s	si	XXX YYYYY
X finishes Y	f	fi	XXX YYYYY

FIGURE 2. The Thirteen Possible Relationships.

Relation	Network Representation
1. i *during* j	N_i --(d)→ N_j
2. i *during* j or i *before* j or j *during* i	N_i --(< d di)→ N_j
3. (i < j) or (i > j) or i *meets* j or j *meets* i	N_i --(< > m mi)→ N_j

FIGURE 3. Representing Knowledge of Temporal Relations in a Network.

4.2. An Example

Consider a simple example of this algorithm in operation. Assume we are given the facts:

S *overlaps or meets* L

S is *before, meets, is met by,* or *after* R.

[1] This fact was pointed out to me by Marc Vilain and was first utilized in his system [18].

B r2 C / A r1 B	<	>	d	di	o	oi	m	mi	s	si	f	fi
"before" <	<	no info	< o m d s	<	<	< o m d s	<	< o m d s	<	<	< o m d s	<
"after" >	no info	>	> oi mi d f	>	> oi mi d f	>	> oi mi d f	>	> oi mi d f	>	>	>
"during" d	<	>	d	no info	< o m d s	> oi mi d f	<	>	d	> oi mi d f	d	< o m d s
"contains" di	< o m di fi	> oi di mi si	o oi dur con =	di	o di fi	oi di si	o di fi	oi di si	di fi o	di	di si oi	di
"overlaps" o	<	> oi di mi si	o d s	< o m di fi	< o m	o oi dur con =	<	oi di si	o	di fi o	d s o	< o m
"over-lapped-by" oi	< o m di fi	>	oi d f	> oi mi di si	o oi dur con =	> oi mi	o di fi	>	oi d f	oi > mi	oi	oi di si
"meets" m	<	> oi mi di si	o d s	<	<	o d s	<	f fi =	m	m	d s o	<
"met-by" mi	< o m di fi	>	oi d f	>	oi d f	>	s si =	>	d f oi	>	mi	mi
"starts" s	<	>	d	< o m di fi	< o m	oi d f	<	mi	s	s si =	d	< m o
"started by" si	< o m di fi	>	oi d f	di	o di fi	oi	o di fi	mi	s si =	si	oi	di
"finishes" f	<	>	d	> oi mi di si	o d s	> oi mi	m	>	d	> oi mi	f	f fi =
"finished-by" fi	<	> oi mi di si	o d s	di	o	oi di si	m	si oi di	o	di	f fi =	fi

FIGURE 4. The Transitivity Table for the Twelve Temporal Relations (omitting "=").

These facts might be derived from a story such as the following:

> John was not in the room when I touched the switch to turn on the light.

where we let S be the time of touching the switch, L be the time the light was on, and R be the time that John was in the room. The network storing this information is

$$R \leftarrow\text{-}(< m\,mi >)\text{- -}\ S\ \text{- -}(om)\rightarrow L.$$

When the second fact is added, the algorithm computes a constraint between L and R (via S) by calling the function *Constraints* with its two arguments, *R1* and *R2*, set to {oimi} and {(mmi)}, respectively. Note that we obtained the inverse of the arc from S to L simply by taking the inverse of each label. *Constraints* uses the transitivity table for each pair of labels and returns the union of all the answers. Since

$$\begin{aligned} T(oi, <) &= (< o\,m\,di\,fi) \\ T(oi, m) &= (o\,di\,fi) \\ T(oi, mi) &= (>) \\ T(oi, >) &= (>) \\ T(mi, <) &= (< o\,m\,di\,fi) \\ T(mi, m) &= (s\,si =) \\ T(mi, mi) &= (>) \\ T(mi, >) &= (>) \end{aligned}$$

we compute (< > o m di s si fi =) as the constraint between L and R and thus obtain the network

$$\begin{array}{l} R \leftarrow\text{-}(< m\,mi >)\text{- -}\ S\ \text{- -}(om)\rightarrow\ L \\ \uparrow \qquad\qquad\qquad\qquad\qquad\qquad\quad | \\ \text{- - -}(< > o\,oi\,m\,di\,s\,si\,fi =)\text{- - - - -} \end{array}$$

Let us consider what happens now when we add the fact

L *overlaps, starts,* or is *during* R

This fact might arise from a continuation of the above story such as

> But John was in the room later while the light went out

Taking the intersection of this constraint with the previously known constraint between L and R to eliminate any impossible relationships gives

$$L\ \text{- -}(os)\rightarrow R$$

To add this constraint, we need to propagate its effects through the network. A new constraint between S and R can be calculated using the path:

$$S\ \text{- -}(om)\rightarrow L\ \text{- -}(os)\rightarrow R$$

From the transitivity tables, we find:

$$\begin{aligned} T(o, o) &= (< o\,m) \\ T(o, s) &= (o) \\ T(m, o) &= (<) \\ T(m, s) &= (m) \end{aligned}$$

Thus the inferred constraint between S and R is

$$S\ \text{- -}(< o\,m)\rightarrow R.$$

Intersecting this with our previous constraint between S and R yields

$$S\ \text{- -}(< m)\rightarrow R.$$

With respect to the example story, this is equivalent to inferring that John entered the room (*i.e.*, R started) either after I touched the switch or at the same time that I finished touching the switch. Thus the new network is:

$$\begin{array}{l} R \leftarrow\text{-}(< m)\text{- -}\ S\ \text{- -}(om)\rightarrow\ L \\ \uparrow \qquad\qquad\qquad\qquad\quad | \\ \text{- - - - - - - - - -}(o\ s)\text{- - - - - - - -} \end{array}$$

Of couse, if there were other nodes in the network, there would be other constraints derived from this new information. Thus, if we added a new interval D, say with the constraint D - -(d)→ S, we would infer the following new relationships as well:

$$D\ \text{- -}(<)\rightarrow R$$

$$D\ \text{- -}(< o\ m\ d\ s)\rightarrow L.$$

4.3. Analysis

A nice property of this algorithm is that it only continues to operate as long as it is producing new further constrained relationships between intervals. Since there are at most thirteen possible relationships that could hold between two intervals, there are at most thirteen steps that could modify this relationship. Thus for a fixed number of nodes N, the upper limit on the number of modifications that can be made, irrespective of how many constraints are added to the network, is 13 × the number of binary relations between N nodes, which is:

$$13 \times \frac{(N-1)(N-2)}{2}$$

Thus, in practice, if we add approximately the same number of constraints as we have nodes, the average amount of work for each addition is essentially linear (i.e., N additions take $O(N^2)$ time; one addition on average takes $O(N)$ time).

The major problem with this algorithm is the space requirement. It requires $O(N^2)$ space for N temporal intervals. Methods for controlling the propagation, saving time and space, will be discussed in the next section.

It should be noted that this algorithm, while it does not generate inconsistencies, does not detect all inconsistencies in its input. In fact, it only guarantees consistency between three node subnetworks. There are networks that can be added which appear consistent by viewing any three nodes, but for which there is no consistent overall labeling. The network shown in Figure 5 is consistent if we consider any three nodes; however, there is no overall labeling of the network.[2] To see this, if we assign the relationship between A and C, which could be f or fi according to this network, to either f alone, or fi alone, we would arrive at an inconsistency. In other words, there is no consistent labeling with A - -(f)→ C, or with A - -(fi)→ C, even though the algorithm accepts A - -(f fi)→ C.

To ensure total consistency, one would have to consider constraints between three arcs, between four arcs, etc. While this can be done using techniques outlined in Freuder [9], the computational complexity of the algorithm is exponential. In practice, we have not encountered problems from this deficiency in our applications of the model. We can verify the consistency of any subnetwork, if desired, by a simple backtracking search through the alternative arc labelings until we

[2] This network is due to Henry Kautz, personal communication.

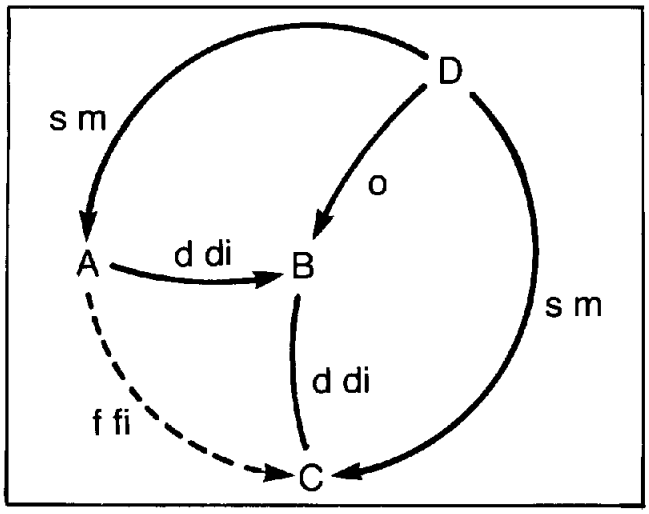

FIGURE 5. An Inconsistent Labeling.

arrive at a labeling for the whole subnetwork in which every arc has only one label.

5. CONTROLLING PROPAGATION: REFERENCE INTERVALS

In order to reduce the space requirements of the representation without greatly affecting the inferential power of the mechanism, we introduce *reference intervals*. Formally, a reference interval is simply another interval in the system, but it is endowed with a special property that affects the computation. Reference intervals are used to group together clusters of intervals for which the temporal constraints between each pair of intervals in the cluster is fully computed. Such a cluster is related to the rest of the intervals in the system only indirectly via the reference interval.

5.1. Using Reference Intervals

Every interval may designate one or more reference intervals (i.e., node clusters to which it belongs). These will be listed in parentheses after the interval name. Thus the node names

I1(R1)

I2(R1, R2)

describe an interval named I1 that has a reference interval R1, and an interval named I2 that has two reference intervals R1 and R2. Since I2 has two reference intervals, it will be fully connected to two clusters. An illustration of the connectedness of such a network is formed in Figure 6.

The algorithm to add relations using reference intervals is identical to the previous addition algorithm except that the comparability condition is no longer universally true. For any node N, let *Refs*(N) return the set of reference intervals for N. For any two nodes K and J, *Comparable(K, J)* is true if

1) *Refs*(K) ∩ *Refs*(J) is not null, that is, they share a reference interval; or
2) K ε *Refs*(J); or
3) J ε *Refs*(K).

Since reference intervals are simply intervals themselves, they may in turn have their own reference intervals, possibly defining a hierarchy of clusters. In most of the useful applications that we have seen, these hierarchies are typically tree-like, as depicted in Figure 7.

If two intervals are not explicitly related in the network, a relationship can be retrieved by finding a path between them through the reference intervals by searching up the reference hierarchy until a path (or all paths) between the two nodes are found. Then, by simply applying the transitivity relationships along the path, a relationship between the two nodes can be inferred. If one is careful about structuring the reference hierarchy, this can be done with little loss of information from the original complete propagation scheme.

To find a relationship between two nodes I and J, where N(i, j) represents the network relation between nodes i and j as in Section 4.1, we use the algorithm:

```
If N(I, J) exists
then return N(I, J)
else do
  Paths := Find-Paths(I, J)
  For each path in Paths do
    R := R ∩ Constrain-along-path(path)
  return R;
end;
```

The function *Find-Paths* does a straightforward graph search for a path between the two nodes with the restriction that each step of the path must be between a node and one of its reference intervals except for the one case where a direct connection is found. Thus, a path is of the general form

$$n_1, n_2, \ldots, n_k, n_{k+1}, \ldots, n_m$$

where all of the following hold:

- for all i from 1 to k − 1, n_{i+1} is a reference interval for n_i;
- n_k and n_{k+1} are connected explicitly;
- for all i from k + 1 to m − 1, n_i is a reference interval for n_{i+1};

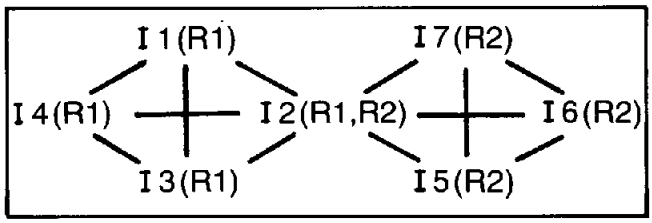

FIGURE 6. The Connectness of a Network with Two Reference Intervals.

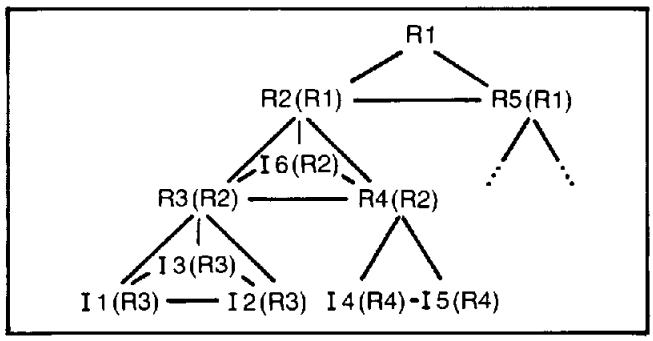

FIGURE 7. A Tree-Like Hierarchy Based on Reference Intervals.

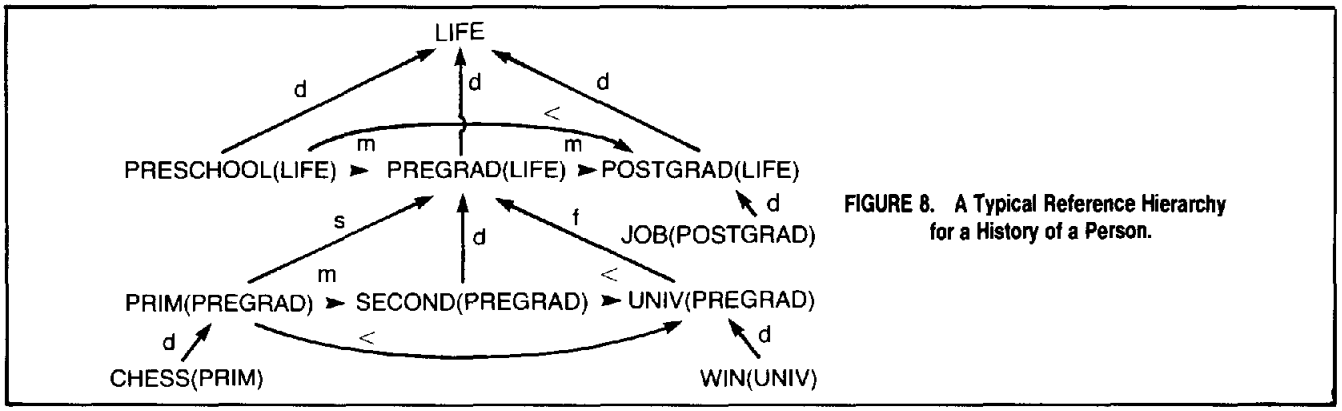

FIGURE 8. A Typical Reference Hierarchy for a History of a Person.

The function *Constrain-along-path* simply takes a path and computes the transitivity constraints along it. Thus if a path consisted of the nodes $n_1, n_2, n_3, \ldots, n_m$, we compute the relation between n_1 and n_m as follows:

$$R := N(n_1, n_2)$$

$$R := \text{Constraints}(R, N(n_2, n_3))$$

$$R := \text{Constraints}(R, N(n_3, n_4))$$

$$\ldots$$

$$R := \text{Constraints}(R, N(n_{m-1}, n_m))$$

where *Constraints* was defined in Section 4.1.

5.2. Examples

There are no restrictions imposed by the system on the use of reference intervals. Their organization is left up to the system designer. Certain principles of organization, however, are particularly useful in designing systems that remain efficient in retrieval, and yet capture the required knowledge. The most obvious of these is a consequence of the path search algorithm in the previous section: the more tree-like the reference hierarchy, the more efficient the retrieval process. The others considered in this section exploit characteristics of the temporal knowledge being stored.

With domains that capture historical information, it is best to choose the reference intervals to correspond to key events that naturally divide the facts in the domain. Thus, if modeling facts about the history of a particular person, key events might be their birth, their first going to school, their graduation from university, etc. Kahn and Gorry [13] introduced such a notion of reference events in their system. Other times in their system were explicitly related to these reference events (i.e., points). In our system, the intervals between such key events would become the reference intervals. Other time intervals would be stored in the cluster(s) identified by the reference intervals that contain them. Thus, we could have a series of reference intervals for the time from birth to starting school (PRESCHOOL), during school (PREGRAD), and after graduation (POSTGRAD). In addition, certain reference intervals could be further decomposed. For example, PREGRAD could be divided into primary and secondary school (PRIM and SECOND) and the time at university (UNIV). The times of the rest of the events would be stored with respect to this reference hierarchy. Figure 8 depicts this set of facts including its reference hierarchy, plus intervals such as the time spent learning chess (CHESS), the time the person won the state lottery (WIN), and the time of the first job (JOB). If an event extended over two reference intervals, then it would be stored with respect to both. For example, if learning to play chess occurred during primary and secondary school, the interval CHESS would have two reference intervals, namely, PRIM and SECOND.

We can now trace the retrieval algorithm for this set of facts. Let us find the relationship between CHESS and WIN. There is no explicit relationship between the intervals, so we must search up the reference hierarchy. Only one path is found, namely:

CHESS(PRIM) - -(d)→ PRIM(PREGRAD) - -(<)→

UNIV(PREGRAD) - -(di)→ WIN(UNIV)

Applying the transitivity relations along the first path, we infer first that

CHESS *before* UNIV

and then

CHESS *before* WIN.

The fact that CHESS is *before* JOB can be inferred similarly from the path

CHESS - -(d)→ PRIM - -(s)→ PREGRAD - -(m)→

POSTGRAD - -(di)→ JOB.

Consider another domain, namely, that of representing information about processes or actions. Such knowledge is required for problem-solving systems that are used to guide the activity of a robot. Each process can be described as a partial sequence of subprocesses. Such a decomposition is not described in absolute temporal terms (i.e., using dates), but by the subprocess's relation to its containing process. Thus a natural reference hierarchy can be constructed mirroring the process hierarchy. For example, consider a process P consisting of a sequence of steps P1, P2, and P3 and another process Q consisting of subprocesses Q1 and Q2 occurring in any order, but not at the same time. Furthermore, let Q2 be decomposed into two subprocesses Q21 and Q22, each occurring simultaneously. To simulate a world in which process P begins before Q begins, we can construct the reference hierarchy in Figure 9. With this organization we can infer relationships between subprocesses of Q and subprocesses of P in the same manner as above. As long as the decomposition of

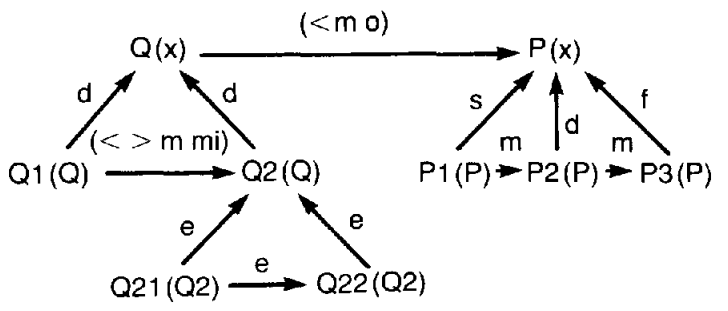

FIGURE 9. A Reference Hierarchy Mirroring a Process Hierarchy.

processes or actions can be done independently (such as in the NOAH system [17]), this organization will capture all the relevant temporal knowledge.

More interesting cases arise when there may be interactions among subprocesses. For instance, we might want to add that Q1 must occur before Q21. Note that, in adding Q1 *before* Q21, we can infer a new relationship between Q1 and Q2 from the path

$$Q1(Q) \text{ - -}(<)\rightarrow Q21(Q2) \text{ - -}(e)\rightarrow Q2(Q)$$

because Q1 and Q2 share the reference interval Q. It does not matter that Q21 does not share a reference interval with Q1. In more complicated cases, we will find relationships between subprocesses such that an important relationship between the processes containing the subprocesses will not be inferred because they do not share a reference interval. For instance, if we learn that Q2 overlaps P1, adding this will not cause the relationship between Q and P to be constrained to simply the *overlaps* relation even though that would be a consequence in the system without reference intervals. There is no path consisting of two arcs from Q to P that is affected by adding Q2 *overlaps* P1.

To allow this inference, we need to reorganize the reference hierarchy. For example, we could, when adding a relation between two noncompatible nodes, expand one of the node's reference intervals with the other node's reference intervals. In this scheme, to add Q2 *overlaps* P1, we would first add P to Q2's reference interval list. Then adding the relation will allow the appropriate changes. In particular, among others, we would infer that

$$Q2(Q, P) \text{ - -}(o)\rightarrow P(X)$$

from the path

$$Q2(Q, P) \text{ - -}(o)\rightarrow P1(P) \text{ - -}(s)\rightarrow P(X),$$

and then infer

$$Q(X) \text{ - -}(o)\rightarrow P(X)$$

from the path

$$Q(X) \text{ - -}(di)\rightarrow Q2(Q, P) \text{ - -}(o)\rightarrow P(X)$$

and the previous constraints between Q and P. The final state of the processes after these two additions is summarized in Figure 10.

Manipulating the reference hierarchies as in this example can be effective if used sparingly. With overuse, such tricks tend to "flatten out" the reference hierarchy as more intervals become explicitly related. In domains where such interactions are rare compared with the pure decompositional interactions, it can be very effective.

5.3. Representing the Present Moment

The technique of reference interval hierarchies provides a simple solution to the problem of representing the present moment. In many applications, such as natural language processing and process modeling, the present is constantly moving into the future. Thus a representation of *NOW* must allow for frequent updating without involving large-scale reorganization of the database each time.

Suppose we have a database in which all assertions are indexed by the temporal interval over which they hold. As time passes, we are interested in monitoring what is true at the present time, as well as in the past and future. The method suggested here is to represent *NOW* as a variable that, at any specific time, is bound to an interval in the database. To update *NOW*, we simply reassign the variable to a new interval that is *after* the previous interval representing the present moment. The key observation is that while the present is continually changing, most of the world description is remaining the same. We can exploit this fact by using refer-

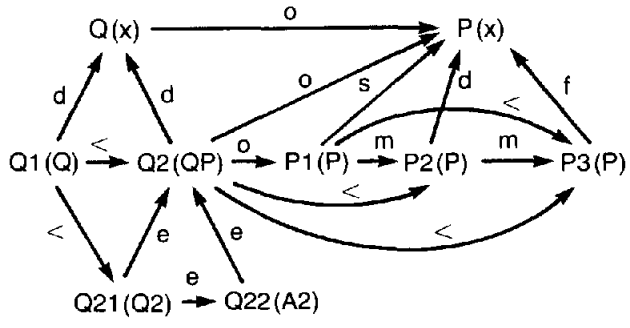

FIGURE 10. The Process Knowledge After Two Updates.

ence intervals to control the inferences resulting from updating *NOW*.

For example, let *NOW* be interval N1, which is *during* its reference interval R1. An example state of the database would be

N1(R1) *during* R1

R1 *before* I1, R1 *after* I2, R1 *during* I3

From this we can infer easily that the present (i.e., N1) is during I3, before I1, and after I2. If *NOW* then is updated (slightly), N2 can be defined as the new *NOW* using the same reference interval by adding the facts

N2(R1) *during* R1, N2(R1) *after* N1(R1)

Thus, *NOW* has been updated but most of the relations in the database have been unaffected, for the effects of N2 will only propagate to intervals referenced by R1. The reference interval R1 has "protected" the rest of the database from a minor change in the present moment.

Of course, eventually *NOW* will cease to be during R1 and a new reference interval will be needed. This will involve a more major update to the database, but the amount of work can be reduced if R1 itself has a reference interval that "protects" much of the database from it.

Thus we need a hierarchy of reference intervals, each containing the present moment. This hierarchy could be designed to mirror the set of English terms that can be used to refer to the present. For example, in English we can refer to the exact moment of an utterance (e.g., at a race, the starter may say "Go now!"), as well as to larger intervals such as "this morning," "today," and "this year." We can also refer to more event-oriented intervals such as "during this lecture" and "while at this bar." These are the types of intervals that should be maintained in the hierarchy representing the present. Furthermore, these intervals typically have well defined starting and termination points. Thus it is reasonable to assume that the temporal database will receive explicit notification when one of them ceases to contain the present. This allows the following important assumption:

> When updating the *NOW* interval, unless otherwise stated, its relationship to its reference interval(s) remains constant.

When one of the reference intervals in the hierarchy ceases to contain the present moment, a new reference interval is selected. (This new interval should usually be provided by the user.) This update is done in the identical fashion as described above with *NOW*. In particular, the relationship with the higher-level reference interval remains constant. A new *NOW* interval, below the new reference interval in the hierarchy, must be introduced. For example, the beginning of a new day would make much of the old hierarchy part of the past (i.e., "yesterday").

While many intervals will be generated by this succession of intervals for *NOW*, many of them can be garbage collected when the reference intervals are updated. In particular, any interval that is not used to index any events or facts may be removed from the database. In a system modeling a natural language dialogue, a large number of these intervals would be used only to index the time of an utterance: These generally can be deleted without harm.

6. DISCUSSION

The temporal representation described is notable in that it is both expressive and computationally feasible. In particular, it does not insist that all events occur in a known fixed order, as in the state space approach, and its allows disjunctive knowledge, such as that event A occurred either before or after event B, not expressible in date-based systems or simple systems based on before/after chaining. It is not as expressive as a full temporal logic (such as that of McDermott [15]), but these systems do not currently have viable implementations.

This balance between expressive power and computational efficiency is achieved by the restricted form of disjunctions allowed in the system. One can only assert disjunctive information about the relationship of two intervals. In other words, we can assert that A is *before* or *meets* B, but not that (A *meets* B) or (C *before* D). This limited form of disjunction is ideal for the constraint propagation algorithm.

The system has been implemented and is being used in a variety of applications. Both FRANZ LISP [8] and INTERLISP versions are running on a VAX 11/780 under UNIX. The system presently also includes the duration reasoner described below. It is currently being used in research in representing actions, events, and interactions that arise in natural language dialogues [1]. We are also using the representation as a world model for research in automatic problem-solving systems [3]. Such systems have long been constrained by their inadequate temporal models.

Vilain [18] has implemented a version of this system which, at the cost of greater space requirements, can perform consistency maintenance. In other words, in his system, when an inconsistency is found, the set of facts that caused the inconsistency can be identified. This system also explicitly allows time points in the representation and has a larger transitivity table, including all interval/point and point/point interactions. This violates the semantics of the interval representation, and so has not been adopted in our present system.

Let us consider why we would like time points, however. They seem to be referred to in English. We can, for instance, talk about the beginning and ending of events. There is no reason to assume, however, that these "endpoints" are truly zero-width points rather than intervals small enough so that they appear to be instantaneous. What this suggests is that there might be a minimum duration ε, such that all intervals of duration less than ε would be viewed as points. This would simplify our reasoning about such times for we would not have to worry about the possibility of two such intervals overlapping. It would be assumed either that these small intervals are equal or that one is before the other.

But this minimum size cannot be fixed in advance. A historian, for instance, may be happy to consider days as points, whereas the computer engineer, when reasoning about a logic circuit, would consider a day to be an eternity. Thus the interval size, where it is appropriate to simplify reasoning by assuming point-like times, varies with the reasoning task.

7. FUTURE RESEARCH AND EXTENSIONS

There are many areas in which this system is being extended. In particular, an interface to a *duration* reasoner has been incorporated into the system, and a system for reasoning about dates will be implemented in the near future. Finally, we are investigating reasoning systems that depend on the notion of persistence.

7.1. Duration Reasoning

We have designed a duration reasoning system based on the same principles as the interval relation reasoner described above. In particular, it is designed to allow relative information (e.g., interval A took longer than interval B) as well as representing uncertainty. The reasoner is again based on con-

straint propagation and a notion of reference durations can be defined.

Very briefly, the duration relationship between two intervals is expressed by outlining a range that includes the multiplicative factor which the duration of the first would be multiplied by to get the duration of the second. For example, the fact that the duration of A is less than the duration of B, expressed as *dur*(A) < *dur*(B), is represented by the relation A - -(0(1))→ B. In other words, *dur*(A) >= 0**dur*(B) and *dur*(A) < 1**dur*(B). The parentheses about the factor 1 indicate an open endpoint; thus the durations of A and B could not be equal. Both the upper and lower duration limits may be open or closed.

Duration information is encoded in a network orthogonal to the relationship network. Propagation across two duration restrictions is accomplished simply by multiplying the respective upper and lower duration limits. For example, if we have the facts

$$\text{dur(A)} <= \text{dur(B)}$$

$$\text{dur(C)} <= \text{dur(B)}$$

$$\text{dur(B)} < 2*\text{dur(C)}$$

which in network form would be

$$\text{A - -(01)→ B - -(1(2))→ C}$$

we derive the relation

$$\text{A - -(0(2))→ C.}$$

The duration reasoner and the interval reasoner are not independent of each other, however. They constrain each other by rules such as the following:

$$\text{If I - -(dsf)→ J then dur(I) < dur(J).}$$

Using this rule, constraints introduced in one network may introduce constraints in the other. In many examples, the networks may exchange information back and forth multiple times before the propagation terminates.

Reference durations correspond to the notion of scales, or common units. Constraints do not propagate through a reference duration. Thus, if the duration HOUR is a reference duration, and we add that dur(A) is between 1 and 2 hours, and dur(B) is less than one half an hour, no relation between dur(A) and dur(B) will be inferred. It will be derived at retrieval time via the reference duration HOUR. Further details on the duration reasoner can be found in [2].

7.2. Date Lines

Having considered the maintenance of relative temporal information in detail, we now consider how to exploit date information when available. Let a *date line* be any representation consisting of a fully ordered set of values taken to correspond to times. A date line corresponding to a simple calendar could be defined as follows:

- values: ordered triples of integers, representing year, month (1–12), and day (1–31) (for example, (50 3 25) represents March 25, 1950)
- comparison operation: orders triples in the obvious manner (for example, (50 3 25) < (75 1 1))

With date lines, the comparison operation between two times on the same date line is relatively inexpensive compared to searching the network of temporal relations.

Date line information could be incorporated into the present system by allowing any interval in the network to have date line information associated with it which identifies the dating system and dates associated with its start and end. The name of the date line is necessary to identify the operations for comparing values. A new interval, added with date line information specified, may affect the relationship to its reference interval and to the other intervals in its "reference class." For example, if two intervals are dated by the same date line, and have date values specified, those values can be used to calculate the exact relation between the intervals. If this relation is more specific than the information stored in the relational network, the network is updated and its effects propagated as usual.

When retrieving a relationship between two intervals dated by the same date line, the date information should be considered first before applying the usual network retrieval mechanism. Sometimes, however, the date line information will not be specific enough to pinpoint a specific relationship, and a network search will still be necessary. It may occur that one of the intervals being considered is dated but the other is not. In this case, the date information may be used only if a relationship can be found between the nondated interval and another interval dated by the same date line. In general, this may be too expensive to consider. A specific case that could be very useful, however, occurs when a reference interval involved in the search is dated by the appropriate date line. We can then compare the two dated intervals to obtain a relationship, which can be propagated back to the nondated interval.

A useful date line for dialogue systems is the time-of-day line. A reasonable implementation of this might have the basic duration of one minute, and have values consisting of an hour-minute pair. If the system were given access to a clock, this date line could be used extensively in the *NOW* hierarchy. Of course, the relative time database is still required to store the facts that are acquired during the dialogue as facts typically hold for much longer than the time that they are being talked about.

If the system does not have such easy access to an internal clock, it may still get time-of-day information occasionally during a dialogue. In this case, some of the *NOW* intervals will map onto the time-of-day line, while others will only be related to it by some relation (e.g., after 10 o'clock). In such a scheme, a new reference interval for the *NOW* interval would be created each time a precise time-of-day value was identified. For example, if the system learns that it is presently 10 o'clock, it can create an "after 10 o'clock" reference interval in which the *NOW* intervals will be contained until the next specific time is acquired. Whether such a technique is feasible requires further search.

7.3. Persistence of Intervals

The last requirement described in the introduction was that the representation should facilitate plausible inferences of the form "if fact P is true now, it will remain true until noticed otherwise." Most of the issues concerning this fall outside the range of this paper, as this system only knows about time intervals. However, a simple trick using this representation makes inferences of the above form easy to implement.

Typically, when a new fact is learned, its exact extent in time is not known. For instance, when I parked by car in the parking lot this morning I knew its location. Sitting at my desk now, I assume it is still there, though I have no proof of that fact. In general, I assume it will remain where it is until I pick it up. Thus, although I do not know the extent of the interval in which my car is parked, I want to be able to assume that this fact holds later in the day. The temporal representation is already based on the observation that most time intervals do not have precisely defined limits. If we

allow the user to specify that some intervals should be assumed to extend as far as possible given the constraints, then we can use such intervals to index facts that are assumed to persist until discovered otherwise.

Thus, if we let a fact P be indexed by a *persistent interval* Tp, then testing P later during an interval t will succeed (by assumption) if it is possible that t is *during* Tp. Checking whether relationships between intervals are possible is easy, since the representation explicitly maintains this information.

For example, let Tp represent the interval in which my car is in the parking lot. I know that Tp is *met by* Tarrive, where Tarrive is the time that I arrived at school today. Then, if *NOW* is represented as interval Tnow, where Tnow *after* Tarrive, we can test if my car is on the parking lot. Since it is there during Tp, we are interested in whether it is possible that Tnow is *during* Tp. The known constraints allow us to infer the following:

Tp *met by* Tarrive, Tarrive *before* Tnow

=> Tp - -(< odim)→ Tnow

Thus it is possible that Tnow is *during* Tp, since it is possible that Tp contains ("di") Tnow. So the test succeeds.

Of course, if it is later learned that the car was found to be missing during time interval Tmiss, then Tp is constrained to be *before* Tmiss (even though it is still persistent). If Tnow is then after or during Tmiss, then it is not possible any longer that Tnow is during Tp.

Managing a system such as this is a difficult problem that requires some form of truth maintenance (e.g., see [6]). These issues, however, are independent of the temporal representation. All that is shown here is that the necessary temporal calculations are easily done within this framework.

An interesting technique suggested by the above may simplify much of the computation required for truth maintenance for this type of assumption. In particular, let us assume that P holds during interval Tp, where Tp is a persistent interval. Furthermore, assume that for any time, P implies Q. Then if we test P at time t, and find it is true by assumption, so we can infer Q during time t. However, if we index Q by Tp rather than by t, then we still can use the fact that Q is true during t (by assumption), but if we ever discover further constraints on Tp that then eliminate the possibility that t is *during* Tp, then both P *and* Q will cease to be true (by assumption) during t. Thus, by indexing all the consequences of P by the same interval, Tp, we can revise our beliefs about P and all its consequences simply by adding constraints about Tp. While this idea obviously requires further investigation, it appears that it may allow a large class of assumption-based belief revision to be performed easily.

8. SUMMARY

We have described a system for reasoning about temporal intervals that is both expressive and computationally effective. The representation captures the temporal hierarchy implicit in many domains by using a hierarchy of reference intervals, which precisely control the amount of deduction performed automatically by the system. This approach is partially useful in domains where temporal information is imprecise and relative, and techniques such as dating are not possible.

Acknowledgments. Many people have provided significant help during the design and development of this work. In particular, I would like to thank Marc Vilain and Henry Kautz for work on developing and extending the system. I have also received many valuable criticisms from Alan Frisch, Pat Hayes, Hans Koomen, Drew McDermott, Candy Sidner, and the referees. Finally, I would like to thank Peggy Meeker for preparing the manuscript, and Irene Allen and Henry Kautz for the valuable editorial criticism on the final draft.

REFERENCES

1. Allen, J. F., Frisch, A. M., and Litman, D. J. ARGOT: The Rochester dialogue system, Proc. Nat. Conf. on Artificial Intelligence, AAAI 82, Pittsburgh, Pa., Aug. 1982.
2. Allen, J. F., and Kautz, H. A. "A model of naive temporal reasoning," to appear in J. R. Hobbs and R. Moore (Ed)., *Contributions in Artificial Intelligence, Vol. 1*, Ablex Pub. Co., Norwood, N.J., 1983.
3. Allen, J. F., and Koomen, J. A. Planning using a temporal world model. Submitted to 8th Int. Joint Conf. Artificial Intelligence, Aug. 1983.
4. Bruce, B. C. A model for temporal references and its application in a question answering program. *Artificial Intelligence 3* (1972), 1–25.
5. Bubenko, J. A., Jr. Information modeling in the context of system development. Proc. IFIP Congress 80, Oct. 1980, North-Holland, Amsterdam.
6. Doyle, J. A truth maintenance system. *Artificial Intelligence 12*, 3, (Nov. 1979), 231–272.
7. Fikes, R. E., and Nilsson, N. J. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence 2*, (1971), 189–205.
8. Foderaro, J. K. The *FRANZ LISP Manual*. Dept. of Computer Science, U. of California, Berkeley, 1980.
9. Freuder, E. C. A sufficient condition for backtrack-free search. *J. ACM 29*, 1 (Jan. 1982), 24–32.
10. Hayes, P. J. The Naive Physics manifesto. In *Expert Systems*, D. Michie (Ed.), Edinburgh U. Press, 1979.
11. Hayes, P. J. Naive Physics I: Ontology for liquids. Working Paper 63, Institut pour les Etudes Semantiques et Cognitives, Geneva, 1978.
12. Hendrix, G. G. Modeling simultaneous actions and continuous processes. *Artificial Intelligence 4*, 3 (1973), 145–180.
13. Kahn, K. M., and Gorry, A. G. Mechanizing temporal knowledge. *Artificial Intelligence 9*, 2 (1977), 87–108.
14. McCarthy, J., and Hayes, P. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence 4*, Edinburgh U. Press, 1969.
15. McDermott, D. A temporal logic for reasoning about processes and plans. *Cognitive Science 6*, (1982), 101–155.
16. Rescher, N., and Urquhart, A. *Temporal Logic*. Springer-Verlag, New York, 1971.
17. Sacerdoti, E. D. *A Structure for Plans and Behavior*. Elsevier North-Holland, New York, 1977.
18. Vilain, M. A system for reasoning about time. Proc. AAAI 82, Pittsburgh, Pa., Aug. 1982.

CR Categories and Subject Descriptors: I.2.4 [**Knowledge Representation Formalisms and Methods**]: Representations—*time, temporal representation*; I.2.3 [**Deduction and Theorem Proving**]: Deduction—*constraint propagation, temporal reasoning*; H.3.3 [**Information Search and Retrieval**]: Clustering, Retrieval Methods

General Terms: Algorithms

Additional Key Words and Phrases: temporal interval, interval reasoning, interval representation.

Received 12/81; revised 3/83; accepted 5/83