Constraint Propagation Algorithms for Temporal Reasoning

Marc Vilain
BBN LABORATORIES
10 MOULTON ST.
CAMBRIDGE, MA 02238

Henry Kautz
UNIVERSITY OF ROCHESTER
COMPUTER SCIENCE DEPT.
ROCHESTER, NY 14627

Abstract: *This paper considers computational aspects of several temporal representation languages. It investigates an interval-based representation, and a point-based one. Computing the consequences of temporal assertions is shown to be computationally intractable in the interval-based representation, but not in the point-based one. However, a fragment of the interval language can be expressed using the point language and benefits from the tractability of the latter.*[1]

The representation of time has been a recurring concern of Artificial Intelligence researchers. Many representation schemes have been proposed for temporal reasoning; of these, one of the most attractive is James Allen's algebra of temporal intervals [Allen 83]. This representation scheme is particularly appealing for its simplicity and for its ease of implementation with constraint propagation algorithms.

Reasoners based on this algebra have been put to use in several ways. For example, the planning system of Allen and Koomen [1983] relies heavily on the temporal algebra to perform reasoning about the ordering of actions. Elegant approaches such as this one may be compromised, however, by computational characteristics of the interval algebra. This paper concerns itself with these computational aspects of Allen's algebra, and of a simpler algebra of time points.

Our perspective here is primarily computation-theoretic. We approach the problem of temporal representation by asking questions of complexity and tractability. In this light, this paper examines Allen's interval algebra, and the simpler algebra of time points.

The bulk of the paper establishes some formal results about the temporal algebras. In brief these results are:

- Determining consistency of statements in the interval algebra is NP-hard, as is determining all consequences of these statements. Allen's polynomial-time constraint propagation algorithm is sound but not complete for these tasks.
- In contrast, constraint propagation is sound and complete for computing consistency and consequences of assertions in the time point algebra. It operates in $O(n^3)$ time and $O(n^2)$ space.
- A restricted form of the interval algebra can be formulated in terms of the time point algebra. Constraint propagation is sound and complete for this fragment.

Throughout the paper, we consider how these formal results affect practical Artificial Intelligence programs.

The Interval Algebra

Allen's interval algebra has been described in detail in [Allen 83]. In brief, the elements of the algebra are *relations* that may exist between intervals of time. Because the algebra allows for indefiniteness in temporal relations, it admits many possible relations between intervals (2^{13} in fact). But all of these relations can be expressed as *vectors* of definite *simple relations*, of which there are only thirteen.[2] The thirteen simple relations, whose definitions appear in Figure 1, precisely characterize the relative starting and ending points of two temporal intervals. If the relation between two intervals is completely defined, then it can be exactly described with a simple relation. Alternatively, vectors of simple relations introduce indefiniteness in the description of how two temporal intervals relate. Vectors are interpreted as the disjunction of their constituent simple relations.

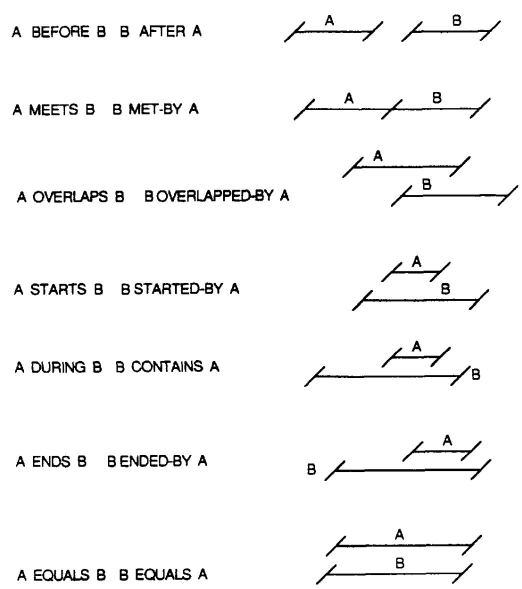


Figure 1: Simple relations in the interval algebra

Two examples will serve to clarify these distinctions (please refer to figure 2). Consider the simple relations *BEFORE* and *AFTER*: they hold between two intervals that strictly follow each other, without overlapping or meeting. The two differ by the order of their

[1]This research was supported in part by the Defense Advanced Research Projects Agency, under contracts N00014-85-C-0079 and N-00014-77-C-0378.

[2]In fact, these thirteen simple relations can be in turn expressed in terms of universally and existentially quantified expressions involving only one truly primitive relation. For details, see [Allen & Hayes 85].

arguments: today John ate his breakfast *BEFORE* he ate his lunch, and he ate his lunch *AFTER* he ate his breakfast. To illustrate relation vectors, consider the vector *(BEFORE MEETS OVERLAPS)*. It holds between two intervals whose starting points strictly precede each other, and whose ending points strictly precede each other. The relation between the ending point of the first interval and the starting point of the second is left ambiguous. For instance, say this morning John started reading the paper before starting breakfast, and he finished the paper before his last sip of coffee. If we didn't know whether he was done with the paper before starting his coffee, at the same time as he started it, or after, we would then have:

PAPER *(BEFORE MEETS OVERLAPS)* COFFEE

Returning to our formal discussion, we note that the interval algebra is principally defined in terms of vectors. Although simple relations are an integral part of the formalism, they figure primarily as a convenient way of notating vector relations. The mathematical operations defined over the algebra are given in terms of vectors; in a reasoner built on the temporal algebra, all user assertions are made with vectors.

Two operations, an addition and a multiplication, are defined over vectors in the interval algebra. Given two different vectors describing the relation between the same pair of intervals, the addition operation "intersects" these vectors to provide the least restrictive relation that the two vectors together admit. The need to add two vectors arises from situations where one has several independent measures of the relation of two intervals. These measures are combined by summing the relation vectors for the measures. For example, say the relation between intervals *A* and *B* has been derived by two valid measures as being both

V_1 = *(BEFORE MEETS OVERLAPS)*
V_2 = *(OVERLAPS STARTS DURING)*.

To find the relation between *A* and *B*, that is implied by V_1 and V_2, the two vectors are summed:

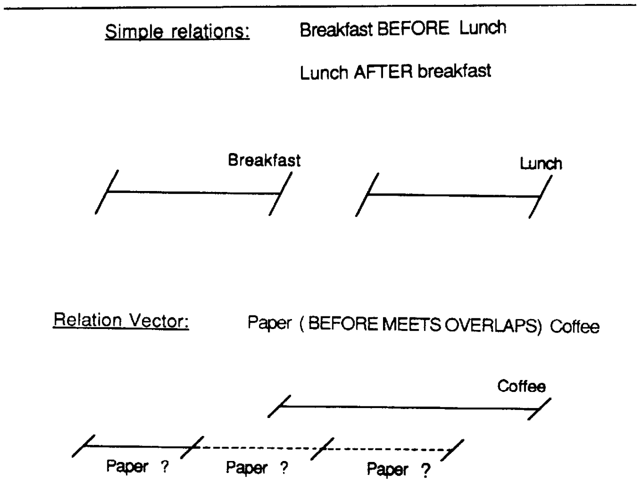


Figure 2: Examples of simple relations and relation vectors

$V_1 + V_2$ = *(OVERLAPS)*.

Algorithmically, the sum of two vectors is computed by finding their common constituent simple relations.

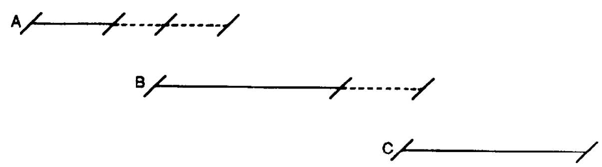
Multiplication is defined between pairs of vectors that relate three intervals *A*, *B*, and *C*. More precisely, if V_1 relates intervals *A* and *B*, and V_2 relates *B* and *C*, the product of V_1 and V_2 is the least restrictive relation between *A* and *C* that is permitted by V_1 and V_2. Consider, for example, the situation in Figure 3. If we have

V_1 = *(BEFORE MEETS OVERLAPS)*
V_2 = *(BEFORE MEETS)*

then the product of V_1 and V_2 is

$V_1 \times V_2$ = *(BEFORE)*

As with addition, the multiplication of two vectors is computed by inspecting their constituent simple relations. The constituents are pairwise multiplied by following a simplified multiplication table, and the results are combined to produce the product of the two vectors. See [Allen 83] for details.



R<A,B> = (BEFORE MEETS OVERLAPS)
R<B,C> = (BEFORE MEETS)

R<A,C> = (BEFORE)

Figure 3: Intervals whose relations are to be multiplied

Determining Closure in the Interval Algebra

In actual use, Allen's interval algebra is used to reason about temporal information in a specific application. The application program encodes temporal information in terms of the algebra, and asserts this information in the database of the temporal reasoner. This reasoner's job is then to compute those temporal relations which follow from the user's assertions. We refer to this process as completing the closure of the user's assertions.

In Allen's model, closure is computed with a constraint propagation algorithm. The operation of this forward-chaining algorithm is driven by a queue. Every time the relation between two intervals *A* and *B* is changed, the pair *<A, B>* is placed on the queue. The algorithm, shown in Figure 4 operates by removing pairs from the queue. For every pair *<A, B>* that it removes, the algorithm determines whether the relation between *A* and *B* can be used to constrain the relation between *A* and other intervals in the database, or between *B* and these other intervals. If a new relation can be successfully constrained, then the pair of intervals that it relates is in turn placed on the queue. The process terminates when no more relations can be constrained.

As Allen suggests [Allen 83], this constraint propagation algorithm runs to completion in time polynomial with the number of intervals in the temporal database. He provides an estimate of $O(n^2)$ calls to the *Propagate* procedure. A more fine-grained analysis reveals that when the algorithm runs to completion, it will have performed $O(n^3)$ multiplications and additions of temporal relation vectors.

> **Theorem 1:** Let *I* be a set of *n* intervals about which *m* assertions have been added with the **Add** procedure. When invoked, the **Close** procedure will run to completion in $O(n^3)$ time.

> **Proof:** *(Sketch[3])* A pair of intervals *<i,j>* is entered on

[3]Most of the theorems in this paper have rather long proofs. For this reason, we have restricted ourselves here to providing only proof sketches.

```
/* Let Table be a two-dimensional array, indexed by intervals, in
   which Table[i,j] holds the relation between intervals i and j.
   Table[i,j] is initialized to (BEFORE MEETS ... AFTER), the
   additive identity vector consisting of all thirteen simple relations;
   except for Table[i,i] which is initialized to (EQUAL).
   Let Queue be a FIFO data structure that will keep track of those
   pairs of intervals whose relation has been changed.
   Let Intervals be a list of all intervals about which
   assertions have been made. */

To  Add(R<i,j>)
   /* R<i,j> is a relation being asserted between i and j.*/

   begin
      Old ← Table[i,j];
      Table[i,j] ← Table[i,j] + R<i,j>;
      If Table[i,j] ≠ Old
         then Place <i,j> on Fifo Queue;
      Intervals ← Intervals ∪ {i, j};
   end;


To Close
/* Computes the closure of assertions added to the database.  */

   While Queue is not empty do
      begin
         Get next <i,j> from Queue;
         Propagate(i,j);
   end;


To Propagate(I,J)
/* Called to propagate the change to the relation between
   intervals I and J to all other intervals. */

   For each interval K in Intervals do
      begin
         Temp ← Table[I,K] + (Table[I,J] x Table[J,K]);
         If Temp = 0
            then {signal contradiction};
         If Table[I,K] ≠ Temp
            then Place <I,K> on Queue;
         Table[I,K] ← Temp;
         Temp ← Table[K,J] + (Table[K,I] x Table[I,J]);
         If Temp = 0
            then {signal contradiction};
         If Table[K,J] ≠ Temp
            then Place <K,J> on Queue;
         Table[K,J] ← Temp;
      end;
```

Figure 4: The constraint propagation algorithm

Queue when its relation, stored in *Table[i,j]*, is non-trivially updated. It is easy to show that no more than $O(n^2)$ pairs of intervals *<i,j>* are ever entered onto the queue. This is because there are only $O(n^2)$ relations possible between the *n* intervals, and because each relation can only be non-trivially updated a constant number of times.

Further, every time a pair *<i,j>* is removed from *Queue*, the algorithm performs O(n) vector additions and multiplications (in the body of the **Propagate** procedure). Hence the time complexity of the algorithm is $O(n \cdot n^2) = O(n^3)$ vector operations.

The vector operations can be considered here to take constant time. By encoding vectors as bit strings, addition can be performed with a 13-bit integer *AND* operation. For multiplication, the complexity is actually $O(|V_1| \cdot |V_2|)$, where $|V_1|$ and $|V_2|$ are the "lengths" of the two vectors to be multiplied (i.e., the number of simple constituents in each vector). Since vectors contain at most 13 simple constituents, the complexity of multiplication is bounded, and the idealization of multiplication as operating in constant time is acceptable.

Note that the polynomial time characterization of the constraint propagation algorithm of Figure 4 is somewhat misleading. Indeed, Allen [1983] demonstrates that the algorithm is sound, in the sense that it never infers an invalid consequence of a set of assertions. However, Allen also shows that the algorithm is incomplete: he produces an example in which the algorithm does not make all the inferences that follow from a set of assertions. He suggests that computing the closure of a set of temporal assertions might only be possible in exponential time. Regrettably, this appears to be the case. As we demonstrate in the following paragraphs, computing closure in the interval algebra is an NP-hard problem.

Intractability of the Interval Algebra

To demonstrate that computing the closure of assertions is NP-hard, we first show that determining the consistency (or satisfiability) of a set of assertions is NP-hard. We then show that the consistency and closure problems are equivalent.

> **Theorem 2:** Determining the satisfiability of a set of assertions in the interval algebra is NP-hard.
>
> **Proof:** *(Sketch)* This theorem can be proven by reducing the 3-clause satisfiability problem (or 3-SAT) to the problem of determining satisfiability of assertions in the interval algebra. To do this, we construct a (computationally trivial) mapping between a formula in 3-SAT form and an equivalent encoding of the formula in the interval algebra.
>
> Briefly, this is done by creating for each term *P* in the formula, and its negation *~P*, a pair of intervals, **P** and **NOTP**. These intervals are then related to a "truth determining" interval **MIDDLE**: intervals that fall before **MIDDLE** correspond to *false* terms, and those that fall after **MIDDLE** correspond to *true* terms. The original formula is then encoded into assertions in the algebra; this can be done (deterministically) in polynomial time.
>
> The encoding proceeds clause by clause. For each clause $P \vee Q \vee R$, special intervals are created. These intervals are related to the literals' intervals **P**, **Q**, and **R** in such a way that at most two of these intervals can be before **MIDDLE** (which makes them false). The other (or others) can fall after **MIDDLE** (which makes them true).
>
> It can then be shown that the original formula has a model just in case the interval encoding has one too. Satisfiability of a 3-SAT formula could thus be established by determining the satisfiability of the corresponding interval algebra assertions. Since the former problem is NP-complete, the latter one must be (at least) NP-hard.

The following theorem extends the NP-hard result for the problem of determining satisfiability of assertions in the interval algebra to the problem of determining closure of these assertions.

> **Theorem 3:** The problems of determining the satisfiability of assertions in the interval algebra and determining their closure are equivalent, in that there are polynomial time-mappings between them.
>
> **Proof:** *(Sketch)* First we show that determining closure follows readily from determining consistency. To do so, assume the existence of an oracle for determining the consistency of a set of assertions in the interval algebra. To determine the closure of the assertions, we run the oracle thirteen times for each of the $O(n^2)$ pairs *<i,j>* of intervals mentioned in the assertions. Specifically, each time we run the oracle on a pair *<i,j>*, we provide the oracle with the original set of assertions and the additional

assertion *i (R) j*, where R is one of the thirteen simple relations. The relation vector that holds between *i* and *j* is the one containing those simple relations that the oracle didn't reject.

To show that determining consistency follows from determining closure, assume the existence of a closure algorithm. To see if a set of assertions is consistent, run the algorithm, and inspect each of the $O(n^2)$ relations between the *n* intervals mentioned in the assertions. The database is inconsistent if any of these relations is the inconsistent vector: this is the vector composed of no constituent simple relations.

The two preceding theorems demonstrate that computing the closure of assertions in the interval algebra is NP-hard. This result casts great doubts on the computational tractability of the algebra, as no NP-hard problem is known to be solvable in less than exponential time.

Consequences of Intractability

Several authors have described exponential-time algorithms that compute the closure of assertions in the interval algebra, or some subset thereof. Valdés-Pérez [1986] proposes a heuristically pruned algorithm which is sound and complete for the full algebra. The algorithm is based on analysis of set-theoretic constructions. Malik & Binford [1983] can determine closure for a fraction of the interval algebra with the exponential *Simplex* algorithm. As we shall show below, their method is actually more powerful than need be for the fragment that they consider.

Even though the interval algebra is intractable, it isn't necessarily useless. Indeed, it is almost a truism of Artificial Intelligence that all interesting problems are computationally at least NP-hard (or worse)! There are several strategies that can be adopted to put the algebra to work in practical systems.

The first is to limit oneself to small databases, containing on the order of a dozen intervals. With a small database, the asymptotically exponential performance of a complete temporal reasoner need not be noticeably poor. This is in fact the approach taken by Malik and Binford to manage the exponential performance of their *Simplex*-based system. Unfortunately, it can be very difficult to restrict oneself to small databases, since clustering information in this way necessarily prevents all but the simplest interrelations of intervals in separate databases.

Another strategy is to stick to the polynomial-time constraint propagation closure algorithm, and accept its incompleteness. This is acceptable for applications which use a temporal database to notate the relations between events, but don't particularly require much inference from the temporal reasoner. For applications which make heavy use of temporal reasoning, however, this may not be an option.

Finally, an alternative approach is to choose a temporal representation other than the full interval algebra. This can be either a fragment of the algebra, or another representation altogether. We pursue this option below.

A Point Temporal Algebra

An alternative to reasoning about intervals of time is to reason about points of time. Indeed, an algebra of time points can be defined in much the same way as was the algebra of time intervals. As with intervals, points are related to each other through relation vectors which are composed of *simple point relations*. These primitive relations are defined in Figure 5.

As with the interval algebra, the point temporal algebra possesses addition and multiplication operations. These operations, whose tables are given in Appendix , mirror the operations in the interval algebra. Addition is used to combine two different measures of the relation of two points. Multiplication is used to determine the relation between two points *A* and *B*, given the relations between each of *A* and *B* and some intermediate point *C*.

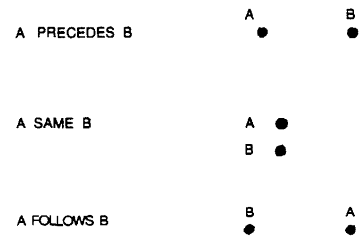


Figure 5: Simple point relations

Computing Closure in the Point Algebra

As was the case with intervals, determining the closure of assertions in the point algebra is an important operation. Fortunately, the point algebra is sufficiently simple that closure can be computed in polynomial time. To do so, we can directly adapt the constraint propagation algorithm of Figure 4. Simply replace the interval vector addition and multiplication operations with point additions and multiplications, and run the algorithm with point assertions instead of interval assertions.

As before, the algorithm runs to completion in $O(n^3)$ time, where *n* is the number of points about which assertions have been made. As with the interval algebra, the algorithm is sound: any relation that it infers between two points follows from the user's assertions. This time, however, the algorithm is complete. When it terminates, the closure of the point assertions will have been correctly computed.

We prove completeness by referring to the model theory of the time point algebra. In essence, we consider any database over which the algorithm has been run, and construct a model for any possible interpretation of the database. If the database is indefinite, a model must be constructed for each possible resolution of the indefiniteness.[4]

We choose the real numbers to model time points. A model of a database of time points is simply a mapping between those time points and some corresponding real numbers. The relations between time points are mapped to relations between real numbers in the obvious way. For example, if time point *A* precedes time point *B* in the database, then *A*'s corresponding number is less than *B*'s.

Theorem 4: The constraint propagation algorithm is complete for the time point algebra. That is, a model can be constructed for any interpretation of the processed database.

Proof: *(Sketch)* We first note that the algorithm partitions the database into one or more partial order graphs. After the algorithm is run, each node in a graph corresponds to a *cluster* of points. These are all points related to by the vector *(SAME)*; note that the algorithm computes the transitive closure of *(SAME)* assertions. Arcs in the graph either indicate precedence (the vectors *(PRECEDES)* or *(PRECEDES SAME)*, or their inverses) or disequality (the vector *(PRECEDES FOLLOWS)*). At the bottom of each graph is one or more "bottom" nodes: nodes which are preceded by no other node.

Further, when the algorithm has run to completion the

[4]This demonstrates completeness in the following sense. If there were an interpretation of the processed database for which no model could be constructed, the algorithm would be incomplete. It would have failed to eliminate a possible interpretation prohibited by the original assertions.

graphs are all consistent, in the following two senses. First, all points are linearly ordered: there is no path from any point in a graph back to itself that solely traverses precedence arcs (time doesn't curve back on itself). Second, no two points that are in the same cluster were asserted to be disequal with the *(PRECEDES FOLLOWS)* vector. If the user had added any assertions that contradicted these consistency criteria, the algorithm would have signalled the contradiction.

Note that all of the preceding properties can be shown with simple inductive proofs by considering the algorithm and the addition and multiplication tables.

The model construction proceeds by picking a cluster of points (i.e., a node) at the "bottom" of some graph and assigning all of its constituent points to some real number. The cluster is then removed from the graph, and the process proceeds on with another real number (greater than the first) and another cluster (either in the same graph or in another one). The process is complicated somewhat because some clusters may be "equal" to other clusters (their constituent points may be related by some vector containing the *SAME* relation). For these cases it is possible to "collapse" several (zero, one, or more) of these clusters together, and assign their constituent points to the same real number. Some other clusters may be "disequal". For these, we must just make sure never to "collapse" them together. Because the choice of which "bottom" node to remove and which clusters to collapse is non-deterministic, the model construction covers all possible interpretations of the database.

Relating the interval and point algebras

The tractability of the point algebra makes it an appealing candidate for representing time. Indeed, many problems that involve temporal sequencing can be formulated in terms of simple points of time. This approach is taken by any of the planning programs that are based on the situation calculus, the patriarch of these being *STRIPS* [Fikes & Nilsson 71].

However, as many have pointed out, time points as such are inadequate for representing many real phenomena. Single time points by themselves aren't sufficient to express natural language semantics [Allen 84], and they are very inconvenient (if not useless) for modelling many natural events and actions [Schmolze 86]. For these tasks, an interval-based time representation is necessary.

Fortunately, many interval relations can be encoded in the point algebra. This is accomplished by considering intervals as defined by their endpoints, and by encoding the relation between two intervals as relations between their endpoints. For example, the interval relation

A *(DURING)* B

can be encoded as several point assertions

A- *(FOLLOWS)* B-
A+ *(PRECEDES)* B+
A- *(PRECEDES)* A+
B- *(PRECEDES)* B+,

where *A-* denotes the starting endpoint of interval *A*, *A+* denotes its finishing endpoint, and similarly for *B*.

This scheme captures all unambiguous relations between intervals, that is all relations that can be expressed using vectors that contain only one simple constituent. It can also capture many ambiguous relations, but not all. One can represent ambiguity as to the pairwise relation of endpoints, but one can not represent ambiguity as to the relation of whole intervals. The vector *(BEFORE MEETS OVERLAPS)* for example can be encoded as point assertions, but the vector *(BEFORE AFTER)* can not. See Figure 6.

INTERVAL VECTOR	POINT TRANSLATION
A (BEFORE OVERLAPS MEETS) B	A- (PRECEDES) B- A- (PRECEDES) A+ A+ (PRECEDES) B+ B- (PRECEDES) B+
A (BEFORE AFTER) B	*No equivalent point form*

ILLUSTRATION

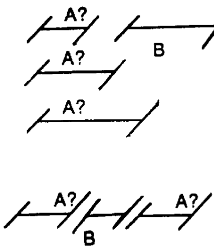


Figure 6: Translation of interval algebra to point algebra

The fragment of the interval algebra that can be translated to the point algebra benefits from all the computational advantages of the latter. In particular, the polynomial-time constraint propagation algorithm is sound and complete for the fragment. This is the interval representation method that Simmons uses in his geological reasoning program [Simmons 83, and personal communication].

This fragment of the interval algebra is also the one used by Malik and Binford [1983] in their spacio-temporal reasoning program. In their case, though, reasoning is performed with the exponential *Simplex* algorithm. This use of the general *Simplex* procedure is not strictly necessary, though, since the problem could be solved by the considerably cheaper constraint propagation algorithm.

Although many applications may be able to restrict their interval temporal reasoning to the tractable fragment of the interval algebra, some applications may not. One program that requires the full interval algebra is the planning system of Allen and Koomen [1983] that we referred to above. In this system, actions are modeled with intervals. For example, to declare that two actions are non-overlapping, one asserts

ACT_1 *(BEFORE MEETS MET-BY AFTER)* ACT_2

As we just showed, this kind of assertion falls outside of the tractable fragment of the interval algebra. In a planner with this architecture, this representation problem can be dealt with either by invoking an exponential temporal reasoner, or by bringing to bear planning-specific knowledge about the ordering of actions.

Consequences of These Results

Increasingly, the tools of knowledge representation are being put to use in practical systems. For these systems, it is often crucial that the representation components be computationally efficient. This has prompted the Artificial Intelligence community to start taking seriously the performance of AI algorithms. The present paper, by considering critically the computational characteristics of several temporal representations, follows this recent trend.

What lessons may we learn from analyses such as this? Of immediate benefit is an understanding of the computational advantages and disadvantages of different representation languages. This permits informed decisions as to how the representation components of application systems should be structured. We can better understand when to use the power of general representations, and when to set these general tools aside in favor of more application-specific reasoners.

A close scrutiny of the ongoing achievements of Artificial Intelligence enables a better understanding of the nature of AI methods. This process is crucial for the maturation of our field.

Appendix: Algebraic Operations in the Point Algebra

Addition and multiplication are defined in the point algebra by the two tables in Figure 7. These operations both have constant-time implementations if the relation vectors between time points are encoded as bit strings. With this encoding, both operations can be performed by simple lookups in two-dimensional (8 x 8) arrays. Alternatively, addition can be performed with an even simpler 3-bit logical *AND* operation.

+	<	<=	>	>=	=	~=	?
<	<	<	0	0	0	<	<
<=	<	<=	0	=	=	<	<=
>	0	0	>	>	0	>	>
>=	0	=	>	>=	=	>	>=
=	0	=	0	=	=	0	=
~=	<	<	>	>	0	~=	~=
?	<	<=	>	>=	=	~=	?

x	<	<=	>	>=	=	~=	?
<	<	<	?	?	<	?	?
<=	<	<=	?	?	<=	?	?
>	?	?	>	>	>	?	?
>=	?	?	>	>=	>=	?	?
=	<	<=	>	>=	=	~=	?
~=	?	?	?	?	~=	?	?
?	?	?	?	?	?	?	?

Key to symbols:

0	is	`()`, the null vector
<	is	`(PRECEDES)`
<=	is	`(PRECEDES SAME)`
>	is	`(FOLLOWS)`
>=	is	`(SAME FOLLOWS)`
=	is	`(SAME)`
~=	is	`(PRECEDES FOLLOWS)`
?	is	`(PRECEDES SAME FOLLOWS)`

Figure 7: Addition and multiplication in the time point algebra

References

[Allen 83] Allen, J. F.
Maintaining Knowledge About Temporal Intervals.
Communications of the ACM 26(11):832-843, November, 1983.

[Allen 84] Allen, J. F.
Towards a General Theory of Action and Time.
Artificial Intelligence 23(2):123-154, 1984.

[Allen & Hayes 85] Allen, J. F. and Hayes, P. J.
A Common-Sense Theory of Time.
In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 528-531. The International Joint Conference on Artificial Intelligence (IJCAI), Los Angeles, CA, August, 1985.

[Allen & Koomen 83]
Allen, James F., and Koomen, Johannes A.
Planning Using a Temporal World Model.
In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 741-747. The International Joint Conference on Artificial Intelligence (IJCAI), Karlsruhe, W. Germany, August, 1983.

[Fikes & Nilsson 71]
Fikes, R., and Nilsson, N.J.
STRIPS: A new approach to the application of theorem proving to problem solving.
Artificial Intelligence 2:189-208, 1971.

[Malik & Binford 83]
Malik, J. and Binford, T. O.
Reasoning in Time and Space.
In *Proceedings of the Eighth Int'l. Joint Conference on Artificial Intelligence*, pages 343-345. The International Joint Conference on Artificial Intelligence (IJCAI), Karlsruhe, W. Germany, August, 1983.

[Schmolze 86] Schmolze, J. G.
Physics for Robots: Representing Everyday Physics for Robot Planning.
PhD thesis, The University of Massachusetts, Amherst, 1986.

[Simmons 83] Simmons, R. G.
The Use of Qualitative and Quantitative Simulations.
In *Proceedings of the Third National Conference on Artificial Intelligence (AAAI-83).* The American Association for Artificial Intelligence, Washington, D.C., August, 1983.

[Valdes-Perez 86] Valdes-Perez, R. E.
Spatio-Temporal Reasoning and Linear Inequalities.
1986.
Unpublished A.I Memo, Massachusetts Institute of Technology Artificial Intelligence Laboratory.