A SYSTEM FOR REASONING ABOUT TIME

Marc B. Vilain

Bolt Beranek and Newman
Cambridge MA 02238

ABSTRACT*

In this paper we describe the salient features of a new system for reasoning about time. The system represents time primarily -- though not exclusively -- in terms of intervals, and performs deductions on this representation. It has a mechanism for maintaining consistency in the representation and discovering the origin of inconsistencies. Further, its deduction mechanisms for intervals are easily extended to deal with time points. Finally, it embodies a simple and elegant scheme for reasoning about absolute dates.

BY MEANS OF INTRODUCTION

Imagine a world in which I were not a computer scientist, but an inveterate explorer, a world traveller. In this world I would have been many places, spending at one time several years in Africa before going on to explore the Peruvian Andes. Imagine also that for some time during my African stay, I contracted a case of beri-beri. Human beings will naturally deduce that my being ill with beri-beri came before my being in Peru. This deduction is typical of the kind of reasoning about time that we have tried to capture in a computer system in current development at BBN. The user of our system makes assertions about the interrelations of events in time. The system in turn deduces new information about the events' interrelations, and makes this information available to the user's queries.

In this paper we will describe the salient features of our system. In particular, we will look at the main representation scheme we have chosen for time (we view time primarily in terms of intervals), and will show how deductions about time can be automated with this representation. We will briefly discuss how our system maintains its representation internally consistent. Finally, we will describe how our deduction mechanisms can be gracefully extended to deal with time points and absolute dates.

*This research was supported by the Advanced Research Projects Agency of the Department of Defense under Contract No. N00014-77-C-0378

A LOGIC OF TIME

There are several ways in which human beings understand time (for example as points, intervals, or with respect to calendar dates). In our system, we have chosen to represent time primarily -- though not exclusively -- in terms of intervals. In so doing we have followed the suggestions of James Allen [2] that intervals are the most computationally natural way of representing time. Relations between time intervals are described in our system by "operators" in a logic. This logic is an extension of that given in [2]; at its core it is composed of 13 relational primitives and a large body of inference rules. The primitives describe unambiguously each of the possible ways that two intervals can be related (they can be equal, overlap, one can precede the other, and so forth). The precise meaning of these primitives is most intuitively communicated by a drawing, and we will hence only give their definitions here in a graphic form (see Figure 1).

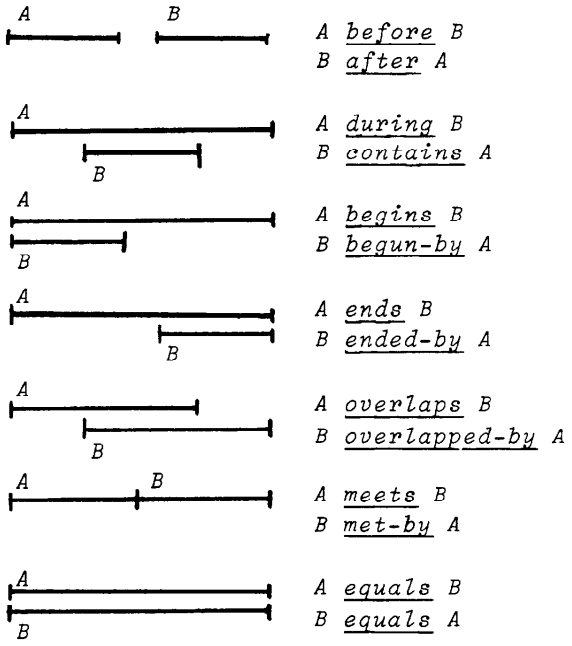

Figure 1: Primitive relations between intervals

The relational primitives can be joined into relational vectors; a relational vector describes a composite relation between two time intervals. For example

A (DURING BEGINS OVERLAPS) B

asserts that interval A is either strictly contained in B (DURING), is contained in B but co-starting with it (BEGINS), or overlaps the "left edge" of B (OVERLAPS). See Figure 2. The semantics of relational vectors is one of exclusive disjunction. That is, exactly one and only one of the primitive components of the vector precisely describes the relation of the intervals linked by the vector. Hence, a vector consisting of only one primitive exactly describes the relation between two intervals, whereas the vector composed of all 13 primitives we interpret as the zero-vector. Asserting that two intervals are related by the zero-vector means that one in fact knows nothing about how they actually relate.

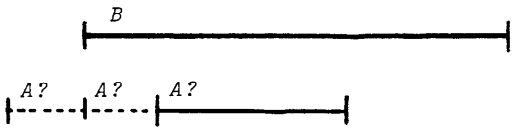

Figure 2: The relation A (DURING BEGINS OVERLAPS) B

We mentioned above that our logic has as part of its core a body of inference rules. These rules are used to combine known assertions and deduce new information. They have the following form.

"If interval A is related to interval B by R1 (1)
and B is related to interval C by R2
then A is related to C by R3"

R1 and R2 are relational primitives and R3 is a vector. The following three rules (illustrated by Figure 3) are typical examples.

A CONTAINS B and B CONTAINS C (2)
=> A (CONTAINS) C

A CONTAINS B and B BEGUN-BY C (3)
=> A (CONTAINS) C

A CONTAINS B and B OVERLAPPED-BY C (4)
=> A (CONTAINS BEGUN-BY OVERLAPPED-BY) C

In our system the rules are used to define the composition properties of the primitive relations of the logic; there is thus one composition rule for each pair of primitive relations (169 rules in total). The rules can be extended in a straightforward way to deal with cases where intervals are related by vectors constructed of

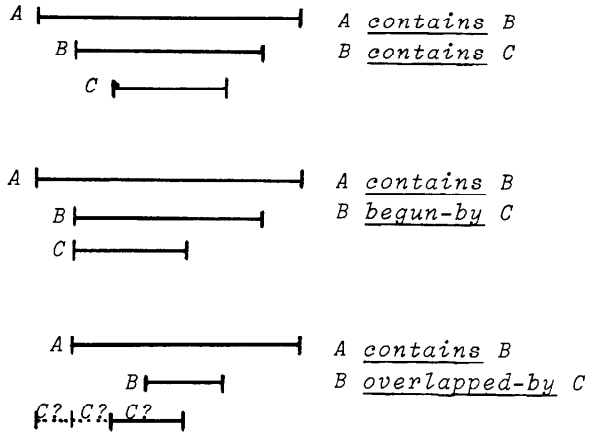

Figure 3: Illustrations of rules 1-3

more than one primitive relation. Consider formula 1 above. Say R1 is actually a vector V = (v_1 ... v_m) and R2 is the vector U = (u_1 ... u_n). Then R3 is computed by combining (disjunctively) the vectors deduced from the composition rules for the pairs of primitives v_i and u_j (for each component v_i of V and each component u_j of U). This process preserves the disjunctive semantics of vectors.

For example, say A is related to B, and B is related to C as in these two assertions:

A (CONTAINS) B
B (CONTAINS BEGUN-BY OVERLAPPED-BY) C.

To compute A's relation to C, we combine the deductions made by the three rules above, and obtain the following result.

A (CONTAINS BEGUN-BY OVERLAPPED-BY) C

USING THE LOGIC

Our system endeavors to maintain a "complete picture" of all the interrelations of all the time intervals the user has declared to exist. That is, for each pair of intervals declared by the user, the system will keep track of the vector that most accurately describes their interrelation. Some of these relation vectors will have been asserted by the user, others must be deduced from the user's original assertions. These deductions are performed in a process of constraint propagation which is guided by the basic composition rules of the time logic. As we saw above, if we know that A relates to B by R1, and B to C by R2, then we can

constrain A's relation to C by the composition rule for R1 and R2. If C is also known to relate to D by R3, then we can constrain A's relation to D by composing R3 with the composition of R1 and R2, and so forth. To maintain the aforementioned complete picture, the system applies the transitive closure of the composition rules to all the relations between all the declared time intervals.

Computing the transitive closure of an operation is a well understood process, but generally requires a clear mathematical formalization of the operation. We have been able to formulate the composition rules of our logic in this way. Indeed, it is possible to state the composition rules in terms of a multiplication and an addition over relation vectors. These operations, along with the appropriate identity elements, define an algebraic structure over relation vectors that is very close to being a semiring. The resemblance to a semiring is sufficiently good that we can compute the transitive closure operation using a modification of a polynomial time algorithm initially designed for closed semirings. The original algorithm, attributed among others to Kleene, is given in [1] and operates in n^3 time and n^3 space, where n is the number of intervals about which assertions have been made.

A GLIMPSE OF CONSISTENCY MAINTENANCE

To deduce the interrelation of two intervals, our system combines information derived from all the assertions that the user has ever made. In so doing, it may discover that some of the user's assertions are in fact mutually contradictory. Contradictions can arise in any number of ways; typical examples include impossible BEFORE/AFTER chains, such as A (BEFORE) B, B (BEFORE) C, and C (BEFORE) A. An important generalization about contradictions in this domain is that they are always due to a set of assertions as a whole. One can not meaningfully single out one assertion from a contradictory set as being the principal cause of the inconsistency.

Our system handles contradictions by a technique inspired by the truth maintenance system of Jon Doyle [5] and the time specialist of Kahn and Gorry [6]. Whenever a new set of assertions is added by the user, the transitive closure operation is recomputed. During this operation, the system monitors partial computations to discover contradictions. If one is found, the system interrupts the transitive closure process and backtraces through its computations. As it backtraces, the system makes use of information it recorded during the deduction process, and isolates the exact set of mutually inconsistent assertions that led to the contradiction. This set is returned to the user.

In part to acknowledge our sources of inspiration, we have named the process by which our system handles contradictions consistency maintenance.

TIME POINTS

At the onset of this paper we noted that intervals are not the only mechanism by which human beings understand time; another common construct is that of time points. Time points are naturally defined by the boundaries of intervals and by certain dating schemes (which we describe below). In fact, much of the earlier literature on reasoning about time describes computer systems whose primary representation of time was in terms of points, not intervals. This is the case with the CHRONOS system of Bruce [4] and the time specialist of Kahn and Gorry.

Our system handles time points in much the same way that it handles intervals: points are objects whose interrelations can be described by primitives in a logic. The logic of points is arrived at by expanding the earlier logic of intervals. To the older logic we add new primitive relations (which like the old ones can be built into vectors), and new composition rules over these primitives (which can be "conjoined" to deal with vectors). The new primitives can be broken into three groups:
(1) those which relate points to other points,
(2) those which relate intervals to points, and
(3) those which relate points to intervals.
As before, we prefer to define these new relations graphically (see Figure 4).

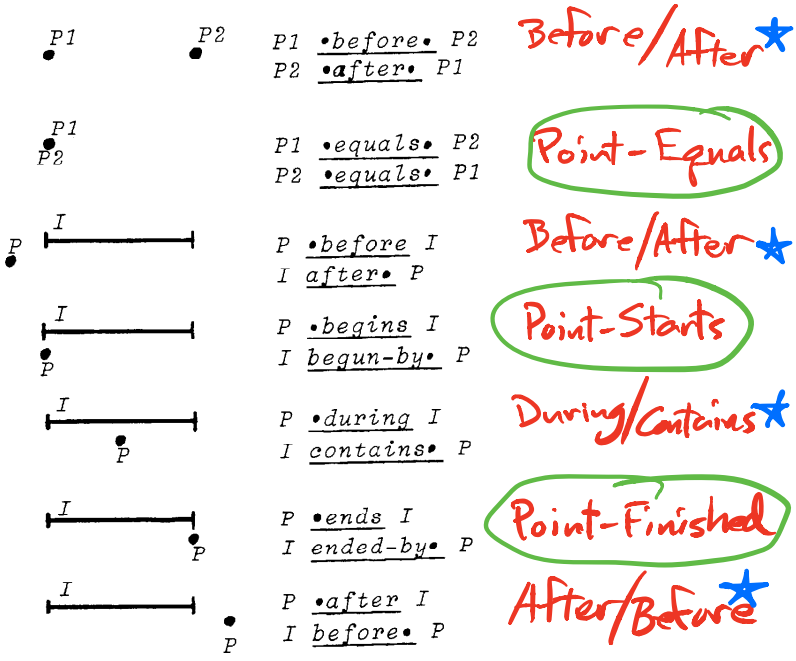

Figure 4: New primitive relations (involving points)

The composition rules that we add to the logic not only define the composition of the new primitive relations with themselves, but also with

the original relations that applied to intervals only. Again, we present some typical examples of these rules (illustrated by Figure 5).

A BEFORE· P1 and P1 ·BEFORE· P2 (5)
=> A (BEFORE·) P2

A BEFORE· P and P ·BEFORE B (6)
=> A (BEFORE) B

A BEGINS B and B CONTAINS· P (7)
=>A (CONTAINS· ENDED-BY· BEFORE·) P

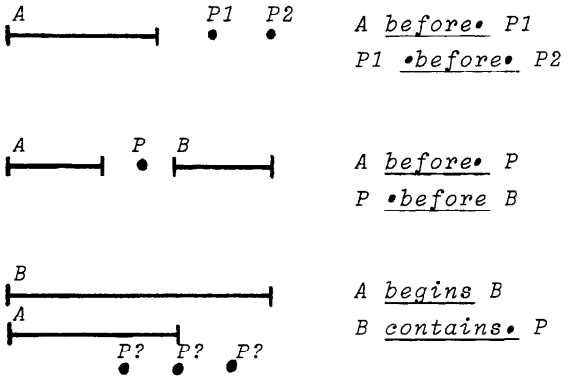

Figure 5: Illustrations of rules 5-7

The mechanism by which our system makes deductions about points is just an extension of that which it uses to make deductions about intervals. As with intervals, the user can declare the existence of certain time points and assert their interrelations to other points or to intervals. Just as before, the system maintains a "complete picture" of all these objects' interrelations by means of a transitive closure operation. The operation is simply performed over the expanded set of composition rules in the newer logic.

As a final note about points, we should state that including them along with intervals in the domain of our system only minimally complicates the deduction algorithms. The polynomial complexity results and the consistency maintenance remain unaffected.

ABSOLUTE DATING

There are two dating mechanisms that are commonly used by people. The first dates entire intervals, and its best example is the standard calendar (which gives a unique name to intervals of an entire day). The second assigns "time stamps" of sorts to particular moments or points in time. This kind of dating is exemplified by the reference to "9:00 o'clock" in the sentence "Bill will arrive by 9:00 o'clock". The time stamps assigned by this method of dating are what we call absolute dates.

We have incorporated a method for reasoning about absolute dates into our system. Our system handles statements about absolute dates by mapping them into the logic of intervals and points. Once this mapping is completed, the original statements involving absolute dates need in fact never be consulted again.

More specifically, whenever the user makes an assertion relating an interval (or point) to a date, the system automatically generates a time point to correspond to the date. This generated time point (which we call a date point) is then appropriately related to the interval (or point) in the user's assertion. The new date point must also be related to all other known date points; the system performs this automatically by simply adding a few new statements to its store of assertions. This process is performed under the guidance of a simple calendar function.

Once the system has generated these (internal) assertions, it can use them to deduce new information by the very same constraint propagation process that operates over intervals and points. It never again need consult the user's original statements relating dates to intervals or points. This is an appealing result since it obviates the need to maintain separate mechanisms for dealing with dated and undated information. This dual reasoning was typically present in earlier time systems, such as that of Kahn and Gorry.

Finally, we should note that the assertions our system generates when creating a date point have the same computational status as the user's undated assertions. This insures that dated assertions will fall under the scrutiny of consistency maintenance just as undated ones do. The system will discover that dated assertions are inconsistent with each other (or with undated information) in exactly the same way that it discovers inconsistencies between undated assertions.

OTHER DIRECTIONS

What we have described in the preceding pages are firmly established features of our system. At the time of writing, most of these features have been implemented. Before closing this paper though, we would like to mention very briefly some of the new directions in which we are extending our work. We are concentrating our efforts primarily in two areas. The first of these is reasoning about absolute duration. We would like to give our system the ability to make deductions based on information about the length of intervals. The second is a search for methods for limiting computation. The polynomial complexity of our deduction algorithms is good, but not ideal: we would like to have an elegant mechanism to limit the amount of computation involved in the deduction

process. We have tentative solutions in both of these areas, and will report upon them more fully in a forthcoming document [8] .

In parting, we would like to place our work in a broader perspective. Recently, several writers have described general models of time and action (specifically James Allen [3] and Drew McDermott [7]). Our efforts are nowhere nearly as ambitious as theirs. Instead we have sought to construct a basic computational tool that could be used by larger programs. (In our own research, for instance, we intend to use our time system as part of a plan recognizer for natural language understanding.) Our approach is actually consistent with that of McDermott and that of Allen. In fact, both of these authors have assumed in their models the existence of underlying time maintenance modules similar to the one described here.

Our goals in this research have all along been to provide a simple but complete inference mechanism over the time domain, one that we hoped would free researchers in AI from having to tackle the low-level details of reasoning about time. We are hoping that our system will permit them to turn their attention to more rewarding investigations in problem solving, language understanding, and other intelligent behavior.

ACKNOWLEDGMENTS

This work would not have been possible without the intellectual stimulation and support of many people. Among them are James Allen, Jim Schmolze, Candy Sidner, and Bill Woods. To them and others, all my thanks.

REFERENCES

[1] Aho, Alfred V., John E. Hopcroft, and Jeffrey D. Ullman.
The Design and Analysis of Computer Algorithms.
Addison-Wesley, Reading, Mass., 1974.

[2] Allen, James F.
Maintaining Knowledge about Temporal Intervals.
Technical Report 86, University of Rochester, Dept. of Computer Science, January, 1981.

[3] Allen, James F.
A General Model of Action and Time.
Technical Report 97, University of Rochester, Dept. of Computer Science, September, 1981.

[4] Bruce, Bertram C.
A Model for Temporal References and Its Application in a Question Answering Program.
Artificial Intelligence 3(1):1-25, 1972.

[5] Doyle, Jon.
Truth Maintenance Systems for Problem Solving.
Technical Report 419, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, January, 1978.

[6] Kahn, Kenneth, and G. Anthony Gorry.
Mechanizing Temporal Knowledge.
Artificial Intelligence 9(1):87-108, 1977.

[7] McDermott, Drew.
A Temporal Logic for Reasoning about Processes and Plans.
Research Report 196, Yale University, Dept. of Computer Science, March, 1981.

[8] Vilain, Marc B.
An Inference Mechanism for Reasoning about Time.
Forthcoming research report, Bolt, Beranek, and Newman Inc.