Time and Time Again: The Many Ways to Represent Time

James F Allen

The University of Rochester

This paper originally appear in the International Journal of Intelligent Systems, 6(4), July 1991.pp341-355 As a result, there are no references to papers past 1990. On the other hand, the issues remain essentially the same.

One of the most crucial problems in any computer system that involves representing the world is the representation of time. This includes applications such as databases, simulation, expert systems and applications of Artificial Intelligence in general. In this brief paper, I will give a survey of the basic techniques available for representing time, and then talk about temporal reasoning in a general setting as needed in AI applications. Quite different representations of time are usable depending on the assumptions that can be made about the temporal information to be represented. The most crucial issue is the degree of certainty one can assume. Can one assume that a time stamp can be assigned to each event, or barring that, that the events are fully ordered? Or can we only assume that a partial ordering of events is known? Can events be simultaneous? Can they overlap in time and yet not be simultaneous? If they are not instantaneous, do we know the durations of events? Different answers to each of these questions allow very different representations of time.

I. Representations Based on Dating Schemes

A good representation of time for instantaneous events, if it is possible, is using an absolute dating system. This involves time stamping each event with an absolute real-time, say taken off the system clock on the machine, or some other coarser-grained system such as we use for dating in everyday life. For instance, a convenient dating scheme could be a tuple consisting of the year, day in the year, hour in the day, minutes and seconds, say. For example, (1990 110 10 4 50) would be the 110th day of 1990, at 10:04 (AM) and 50 seconds. The big advantage of dating schemes is that they provide for constant time algorithm for comparing times and use only linear space in the number of items represented. Time comparisons are reduced to simple numeric comparisons. Date-based representations are only usable, however, in applications where such information is always known, i.e. applications where every event entered has its absolute date identified. There are many applications where this is a reasonable assumption. For instance, applications involving real-time data gathering, databases of transactions on a single machine, say a central machine maintaining banking records, or a system to maintain one's appointment calendar. In these applications, a date-based representation should be the representation of choice. In addition, with absolute dating, we also have information about the duration of time between events - we simply subtract the date of the later event from the date of the earlier one.

Even if absolute dating is not available, if the events are always known in a full linear ordering, then a variant of this approach using pseudo-dates is possible. For instance, assume that the events are always fully ordered when entered. In this case, we can arbitrarily assign a number to each event such that the numeric ordering reflects the linear ordering. When a new event is added to an existing database, say between two known events *e1* and *e2*, then a new number is computed for the new event that is between the numbers for *e1* and *e2*. One simple technique is to take the midpoint between the times, namely (Time(*e1*)+Time(*e2*))/2. This scheme will work until the limits of precision on the machine is reached. In this unlikely event, a new numbering would need to be reassigned to the set of events. On average, this representation is as efficient with regards to space and time as the original dating scheme since all time comparisons are again reduced to numeric comparisons. With pseudo-dates, however, we have lost all information about durations between events.

More complicated representations are needed if the strong assumptions above cannot be met. For instance, what if the exact time of each event is not necessarily known? Rather, the system is given ranges within which the event is known to occur. Assuming that events are still instantaneous and that specific event-to-event relationships are not known except by means of the ranges specified, we can use a modified dating scheme where the time of each event is constrained by the earliest possible and latest possible time for the event. Consider this with an absolute dating scheme first. Each event is entered into the system with a pair of dates (*e,l*) specifying the earliest and latest possible time of occurrence. Temporal comparison is now a comparison of intervals as follows. Given two events with times (*e1,l1*) and (*e2,l2*) respectively, then we have the following method of comparing the event times:

> If $l1 < e2$ then event 1 is before event 2
>
> If $l2 < e1$ then event 2 is before event 1
>
> Otherwise, the ordering is unknown

If the dates are real-time dates, then we can also derive bounds on the duration between events as well. For example, if event 1 is before event 2, then we know the duration between them is at least ($e2 - l1$) and at most ($l2 - e1$).

Unfortunately, the same representation does not directly work with pseudo-times. Consider a simple scheme that might appear to work at first glance: To insert an event between two events with times (*e1,l1*) and (*e2,l2*), where $l1 < e2$, we simply assign the new event the time (*l1,e2*). If event 1 and event 2 were previously unordered, and now we are adding that event 1 precedes event 2, we would need to pick a new pseudo date, say dividing the interval that contains both events into thirds, using the first third for event 1 and the last third for event 2. This would leave room for the possible addition of an event between them at a later stage. Unfortunately, undesired interactions may occur between the partial orderings. In particular, consider the partial ordering with the following events *start, mow, rake, gotobank, buyfood, eat*, where you must both mow and rake the lawn before lunch, and you get money and buy food before lunch, as shown in Figure 1. Each partial ordering, shown by the arrows, is encoded correctly by the pseudo times as shown. But we inadvertently have added undesired orderings between the partial orders: mow now precedes buyfood, for instance, whereas the two events are really unordered with respect to each other.

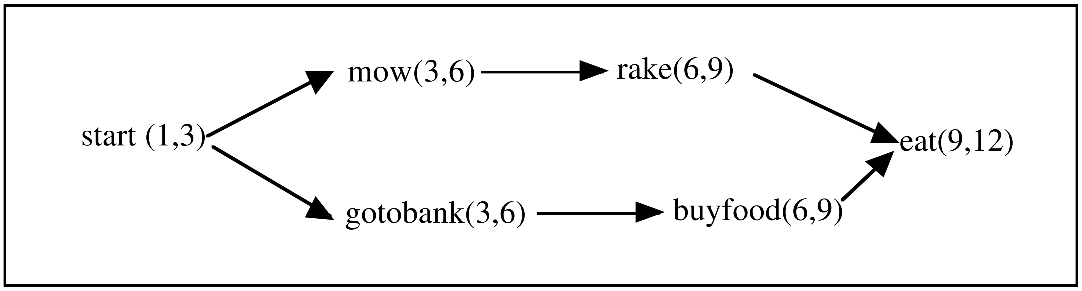

Figure 1: An attempt to encode a partial ordering with pseudo times

No simple modification of this representation can resolve this problem, although the technique can be used as part of a more complex representation. Miller and Schubert (1990), for instance, use a pseudo-date scheme for subsets of events that form a total ordering, and use a graph structure for capturing partial ordering constraints. This allows some queries to be answered in constant time, while others may require a graph search (an order n algorithm). The advantage of this scheme over the constraint -based approaches to be discussed next is mainly the amount of space required. By using a pseudo-date scheme, the Miller & Schubert representation is considerably more space-efficient in the average case.

II. Constraint Propagation Approaches

There has been a considerable amount of work in Artificial Intelligence in defining temporal reasoning systems that used the technique of constraint propagation. These systems use a graph-based representation where each time is linked to each other time with an arc labeled with the possible temporal relationships between the times. Specifically, the possible relations between any two time points are <,>, and =. An arc labeled (<,=) between two time points *e1* and *e2* would represent the fact that $e1 < e2$ or $e1 = e2$. No information between two time points would be indicated by the label (<,=,>), i.e. every relationship is possible. Note that such disjunctions can be represented on an arc label, whereas an arbitrary disjunction involving more than two time points, say $e1<e2$ or $e2<e3$, cannot. Thus this representation provides a very limited form of disjunctive reasoning.When new information is added to this representation, it may constrain existing information because of transitivity constraints. In particular, we know that if $e1<e2$ and $e2<e3$, then *e1* must be before *e3*. These constraints, based on transitivity, can be used to incrementally update the graph. For example, say originally that *e1* is before or equal to *e2*, and *e2* is before or equal to *e3*. This allows us to infer that *e1* must be before or equal to *e3*. This situation is shown in the constraint graph in Figure 2. Now if we add an event *e4* such that $e1<e4<e2$, then this introduces a new constraint by transitivity that $e1 < e2$. So the arc between *e1* and *e2* is updated, and now transitivity can be applied again, since $e1<e2<=e3$ implies that $e1<e3$, and the arc between *e1* and *e3* is updated. The result is shown in Figure 3. Vilain and Kautz (1986) show that such a constraint propagation algorithm operating on constraints that exclude the restriction *e1* (<,>) *e2*, i.e. the two events are not equal but are otherwise unordered, is sound and complete and operates in order n^3 time. van Beek(1989) develops an order n^4 algorithm that is sound and complete for the complete algebra including every possible form of constraint between points.

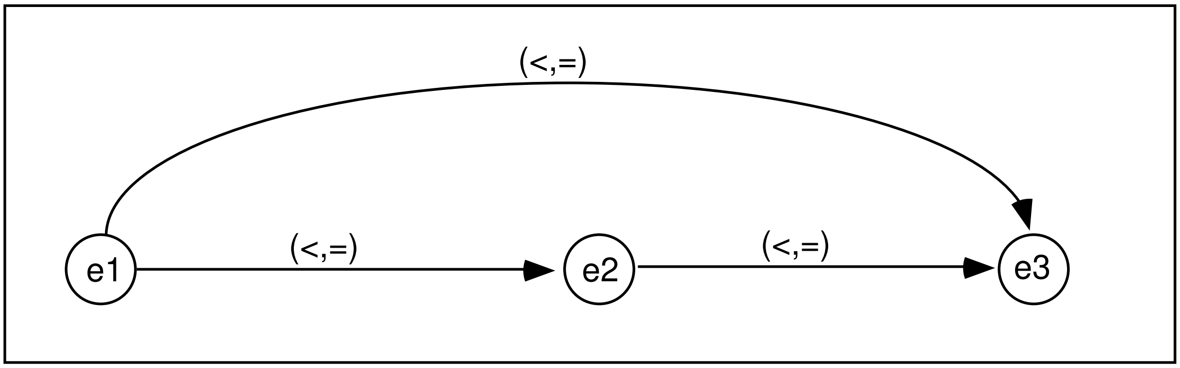

Figure 2: A simple point-based constraint graph

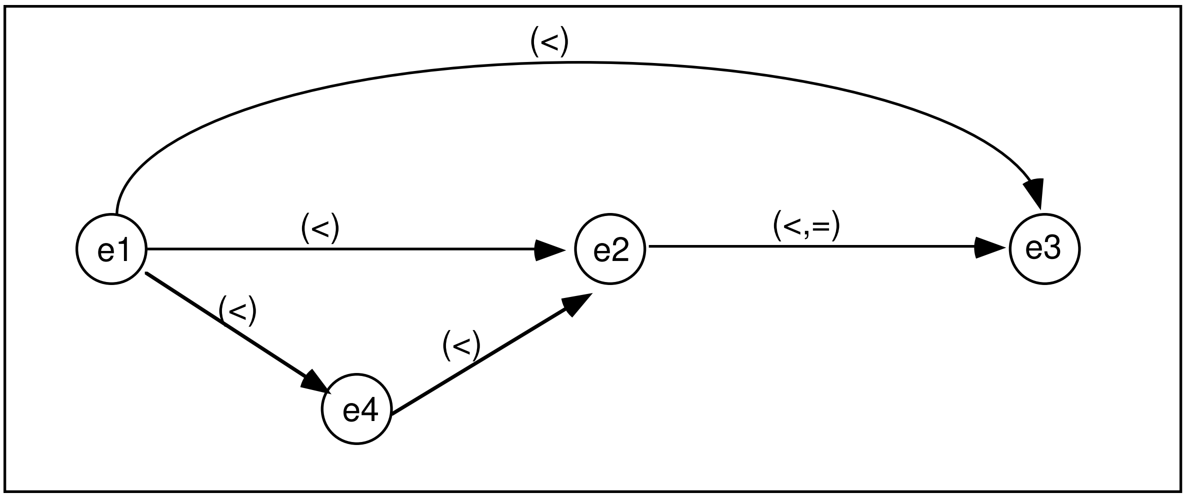

Figure 3: After adding e1<e4 and e4<e2

The situation becomes more complicated as we relax the assumption that all events are instantaneous. In this case, events now take place over intervals. We can express intervals as pairs of points - the start time and the end time, but there are quite natural interval relationships that cannot be captured in a simple point-based representation. In particular, using a constraint-based point reasoner as described above to implement an interval-based reasoner will impose some restrictions.

First consider some examples. Say event 1 occurs over the interval *E1* (= *(e1,l1)*) and event 2 occurs over the interval *E2* (=(*e2,l2*)). Now the interval relationship that *E1* is *:before E2* is easily captured, namely by the point constraint $l1 < e2$. A more complicated example is that *E1 :overlaps E2*, which is captured by the conjunction $e1 < e2$ & $e2 < l1$ & $l1 < l2$. Figure 4 shows the example of this situation in both representations. Allen (1983) defines the thirteen possible interval relationships, each one expressible as a conjunction of point relationships. Seven are shown in Figure 5, and the remaining ones are simply the inverses (i.e. reversing the order of the two arguments to get *:after, :metBy,* etc). The complication arises when uncertainly is introduced. A simple disjunction at the interval level, say that E1 is :before or :after E2, becomes a complex disjunction at the point level, namely $l1 < e2$ or $l2 < e1$. Figure 6 shows an interval relation that cannot be expressed as a simple point-based constraint graph because such disjunctions involving more than two points are not expressible in a single arc label between two points

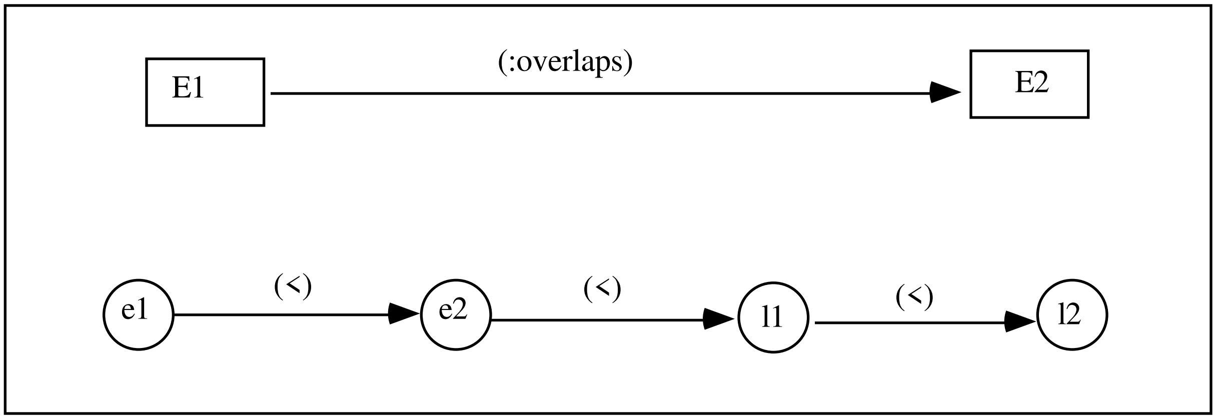

Figure 4: Interval E1 overlaps E2 in an interval-based representation and the equivalent point-based representation

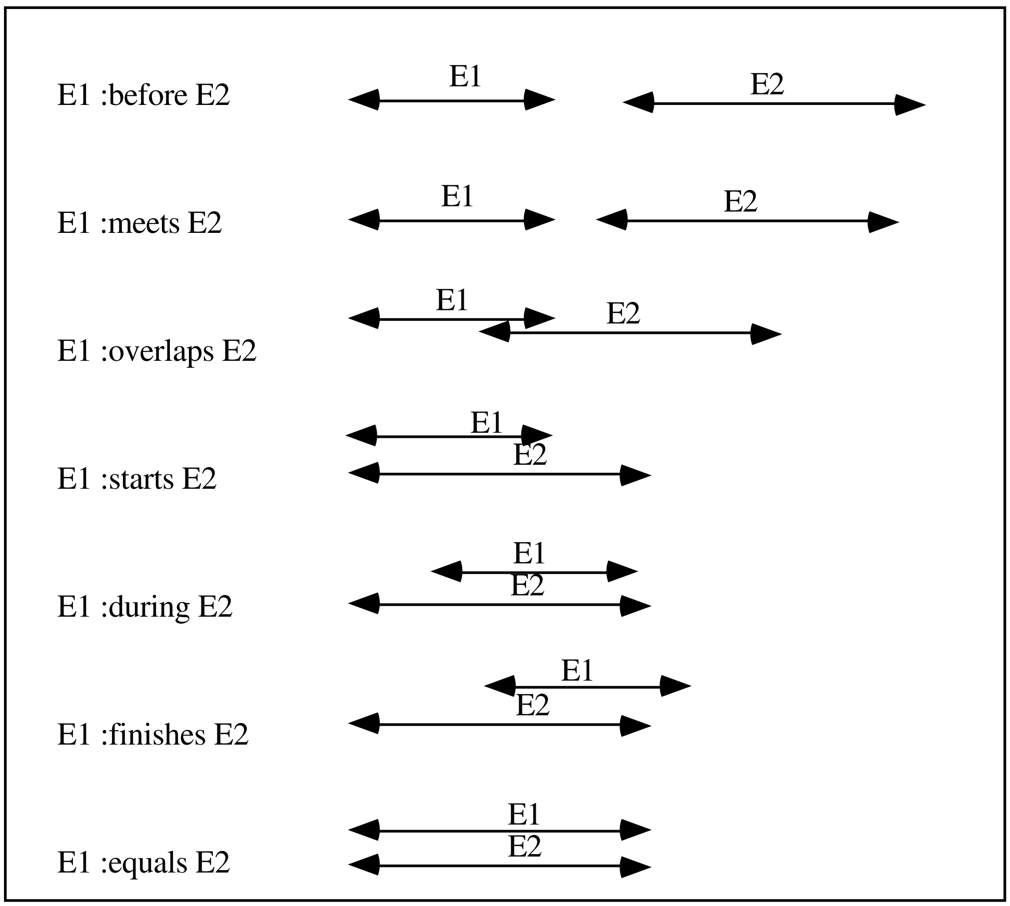

Figure 5: The interval relationships

Allen (1983) developed a constraint propagation algorithm based on intervals rather than points that can capture such relationships directly. Because it can represent any disjunction of the thirteen possible simple relationships between intervals, this representation can capture 2^{13} different possible constraints between any two intervals. Only 181 of these are expressible as simple point-based constraints (Ladkin & Maddux,

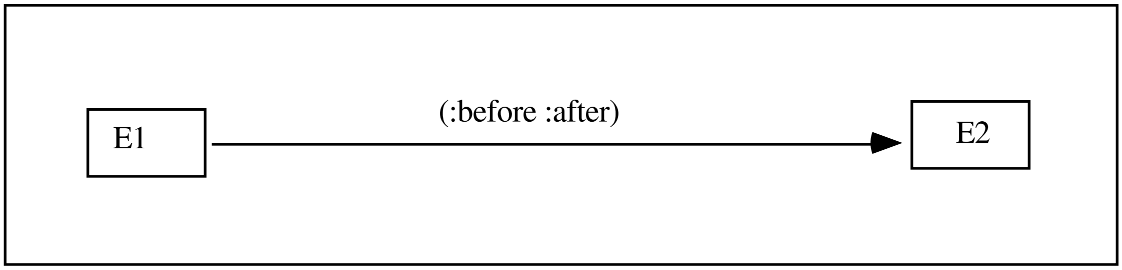

Figure 6: An interval constraint not representable in a point-based constraint graph

personal communication). This representation is powerful enough that any complete algorithm that guarantees consistency on the temporal information is NP-complete in the worse case (Vilain & Kautz, 1986). Allen (1983), however, presents a heuristic order n^3 algorithm that captures most or all intuitively simple cases. Vilain and Kautz show that this algorithm is complete if all the constraints obey the property of "continuity" - that is that all constraints on endpoints could be expressed by a continuous range between some maximum and minimum point-time. In other words, it is complete with respect to any situation that could be expressed by point-based constraints as described above. But the algorithm also handles more complex situations as well, but without the guarantee of completeness. In particular, the algorithm could accept a set of disjunctive labels in the graph, where there is no actual labeling that would be consistent. Allen (1983) gives an example of such a graph. In general, such situations are rather hard to construct and none have yet been found that correspond to some intuitively natural situation. The most intuitive situations devised so far resemble logic puzzles that people have difficulty with as well. So it may be that this algorithm captures those aspects of temporal reasoning that people find intuitively simple, but this is pure speculation. At the present time, however, if you need an efficient algorithm in an application that requires that two intervals might be disjoint, but are not otherwise ordered, there is no choice but to accept the incompleteness.

In some cases, the temporal intervals can be roughly partitioned into subsegments where all the complexity is within the subsegments rather than between them. This can lead to substantial actual efficiency improvements. Koomen (1989) developed a system that can automatically compute such clustering within the framework of Allen's algorithm. Generalizing even further, to arbitrary disjunctions about temporal intervals transforms the problem into general theorem proving techniques, which are not typically efficient enough for use in applications.

A formal definition of interval time can be found in Allen & Hayes (1989). This logic uses one primitive relation, :meets, and one primitive object, the time period. Allen & Hayes present a brief axiomatization of interval logic (5 axioms), and show how the logic can be extended to distinguish between time points, moments of time, and time intervals. Ladkin (1987) shows that all the semantic models of this axiomatization are isomorphic to an interval structure based on an unbounded linear order. Ladkin (1987) and van Benthem (1983) present similar axiomatizations that are characterized by interval structures on an unbounded, dense, linear order. Thus the Allen & Hayes model is weaker in that it does not require the time line to be dense (i.e. if $t1 < t3$, the dense time requires that there exists a $t2$ such that $t1<t2<t3$).

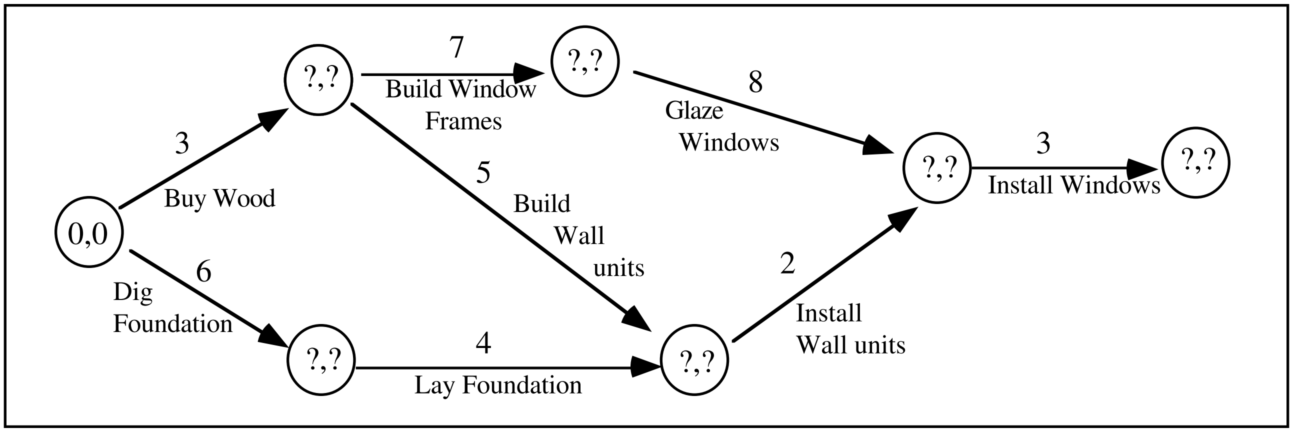

Figure 7: A simple PERT network showing durations of each event (i.e. arc)

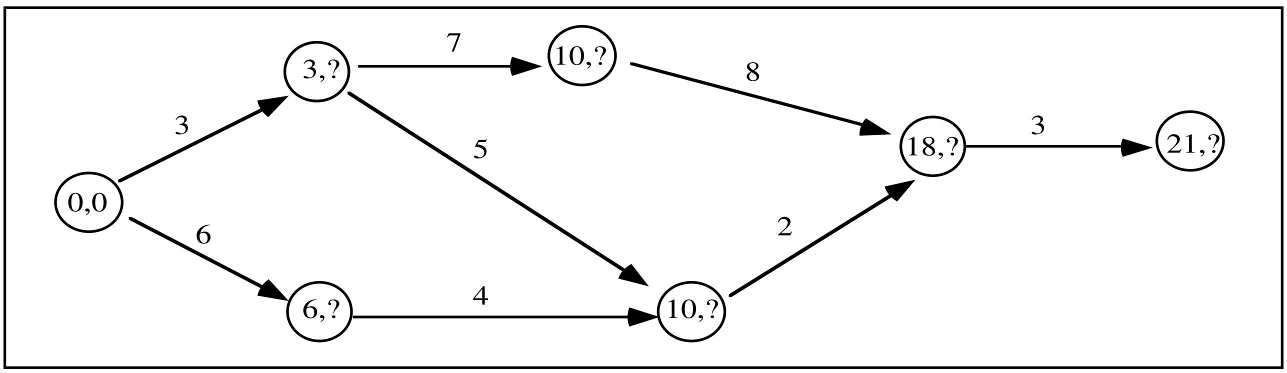

Figure 8: the same network after finding the earliest possible starting times

III. Duration-Based Representations

With the exception of the first technique using absolute dates, we have been ignoring the problem of representing temporal durations. In this section we will examine some representations that operate primarily using duration information. The basic technique for dealing with duration information is seen in PERT networks. This representation maintains a partial ordering of events in an acyclic directed graph that has both a distinguished beginning and ending event. Each node in the graph represents an event and has an associated duration. An example is shown in Figure 7, which represents the initial dependencies between the events needed to build a simple hut. Events are represented by arcs, which are labeled by the duration of the event. The initial node is labeled with (0,0), indicating that the earliest possible starting time is 0, and the latest starting time, given that the task is completed as soon as possible, is also 0. This state represents the beginning of the task. We can now calculate the earliest starting time for every node in the network by starting at the beginning node and assigning it time 0. Each node directly linked to the starting node then has an earliest start time greater or equal to the duration of the event represented by the arc. In general, if node *i* is linked to node *j*, then node *j*'s earliest start time must be equal or greater to the earliest start time of *i* plus the duration of the arc from *i* to *j*. Since there may be many different paths from the

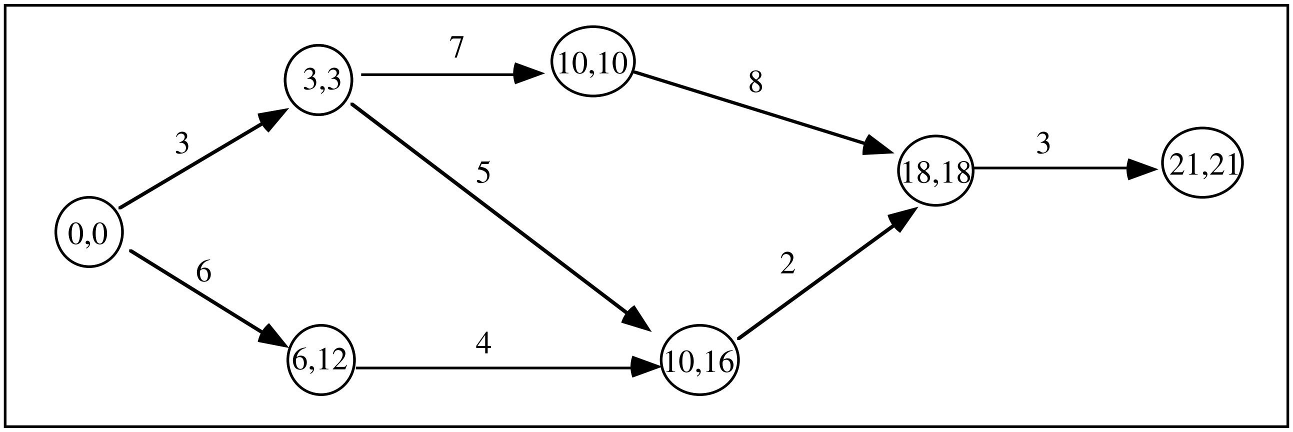

Figure 9: The network after computing the latest possible start times

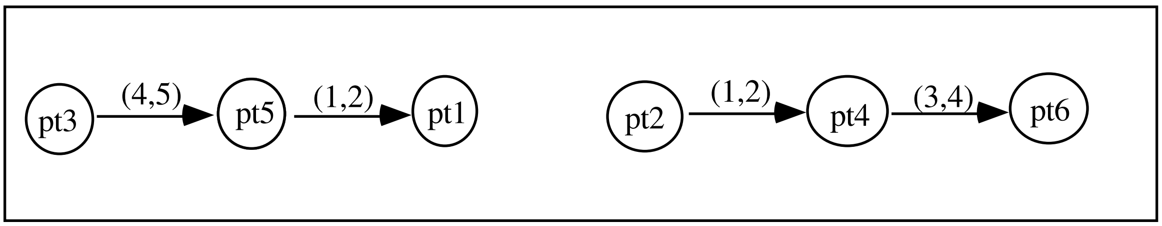

Figure 10: A simple time map

beginning node to an arbitrary node *j*, the earliest start time for *j* will be the maximum computed over any path. The results of computing the earliest start times for the example network is shown in Figure 8. Essentially the same algorithm can then be run backwards from the end node once its earliest start time is derived, to compute the latest start time for each node in order to complete the entire task by the earliest possible time. The results of this step are shown in Figure 9. Thus we have derived a window for the start of each event in order to minimize the total time of the event sequence. This technique is very useful in scheduling applications, for which it was originally developed. As a general representation of time, however, it is lacking. In particular, it is only useful if the duration of each event is known. While we could generalize the algorithm to use a range of possible durations for each event rather than an exact duration, one unknown duration (i.e. an arc with range 0 to infinity) defeats the algorithm. Furthermore, ordering constraints that are not representable by a partial ordering of endpoints are impossible to represent. In cases where duration information is available, the technique yields a representation that is similar in power to the pseudo-date system described earlier. Incrementally adding new events into the network becomes an expensive task however, which can involve recomputing the entire network in the worst case.

Dean & McDermott (1987) develop a representation that encodes all temporal information in terms of duration constraints. Rather than using durations of events as a base, as described above, they represent all information as durations between time points. More qualitative relationships between points can be captured by allowing the use of infinity as a value. Thus the fact that p is before or equal to q is expressed as stating that the duration between p and q is between 0 and infinity. For instance, Figure 10 shows a simple time map encoding an example from Dean & McDermott (1987, p 41). This encodes the constraints that

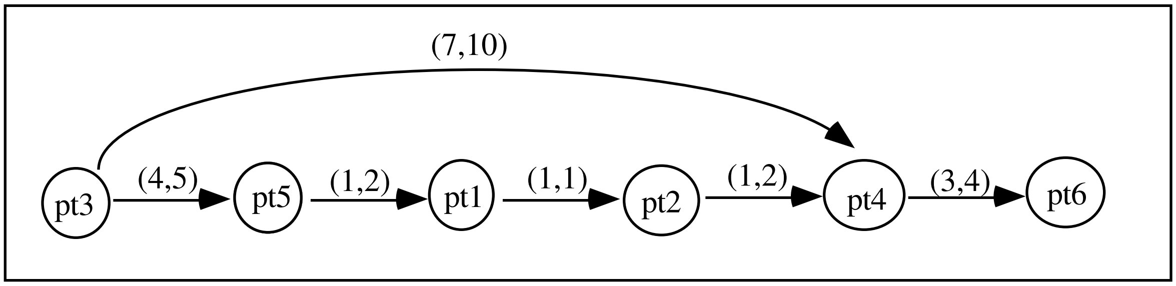

Figure 11: The same map after adding pt1 (1,1) pt2

there is between 4 and 5 units of time between pt3 and pt5, 1 to 2 units between pt5 and pt1, etc. Using simple transitivity of durations over paths, they could derive that the duration between pt3 and pt1 must be between 5 and 7 (i.e.by adding up the lower and upper bounds along the path). If we now add the constraint that pt2 is exactly one unit of time after pt1, then the system uses path finding techniques to infer certain consequences. For instance, we could infer that pt3 is between 7 and 10 units before pt4, as shown in Figure 11. Their system does not automatically compute all consequences of new information as is done with the constraint-based approaches. Rather the user indicates relationships that are of interest and these are monitored as new information is added. For instance, if the user has indicated that the fact that pt3 is before pt4, (i.e. pt3 (0,+inf) pt4) is of interest, this would be inferred in the example discussed above.

If there is no information about the relationship between two points, this is equivalent to asserting the duration between them to be between minus-infinity and plus-infinity. For their algorithm to be effective, duration information must be fairly well known for the most part. Since the algorithm depends on heuristic graph search, it is hard to characterize the actual complexity or power of the system. Using a complete search algorithm however, the algorithm appears to be similar in expressive power to the interval representation restricted to continuous constraints as described in Vilain and Kautz (1986), except that they have durations represented in addition to the qualitative temporal ordering.

IV. Temporal Logics

So far we have only discussed the representation of temporal information. For such a capability to be useful it must be embedded within a more general representation that can encode general assertions about the world. The simplest form of such a representation assumes a linear time model and indexes each fact with a time - e.g. in a logic-based representation the time could simply be an extra argument. Thus *Green(FROG1,T1)* might assert that the object *FROG1* was green at time *T1*. With a predicate like green, it does not matter whether T1 is an interval or a time point. In fact, if *T1* were an interval, we could define the meaning of this formula as an abbreviation of an assertion that *FROG1* is green over all time points contained in *T1*.

This technique does not work with all predicates, however. Consider the predicate stating that some event occurred - say that the sun rose. While we can say that the sun rose over some interval of time, this is not equivalent to asserting that the sun rose over every point in that interval. In fact, the sun didn't rise over any point in time. Such predicates representing events necessarily take intervals as arguments. In a point-based temporal representation, this means that the predicate *Rise* would take two time points, i.e. *Rise(t1,t2)* is true only if the sun rose between times *t1* and *t2*. In an interval-based representation, there need be only one temporal argument, i.e. *Rise(T1)* is true only if the sun rose over interval *T1*. The difference between predicates such a *Green* and *Rise* has been noticed for a long time. Vendler (1957) pointed out these distinctions in his work analyzing the verbs of English. We can define different properties of predicates:predicates like Green are homogeneous - if the predicate holds over an interval, it holds over all subintervals and points in that interval. Predicates like *Rise*, have the opposite property: if it holds over an interval *T* then it doesn't hold over any subinterval of *T*; and the processes, such as the predicate *Running*, which if it holds over two intervals *T1* and *T2* such that *T1* meets *T2*, then it also holds over the union of these two intervals. These observations have been made and formalized in many different ways (e.g. see Allen, 1983, Dowty, 1986, Shoham, 1987) and have interesting consequences. In particular, if one starts with a point-based logic, which defines intervals as the set of points in the interval, and defines a predicate to be true over an interval only if its true over all the points that make up the interval, then such a logic cannot represent non-homogeneous predicates.

More complex models of time have been developed in formal work but have rarely been explicitly incorporated into temporal reasoning systems. For instance, one very common model of time involves forward branching time. In this case, all times are not directly comparable, as they are when we think of times as linearly ordered.Rather, some times are incomparable. Branching time is used to represent the notion of future possibility. There might be two futures of interest, for instance, one where the agent eats lunch, and the other where the agent goes jogging instead. In the first possible future, there might be a time tgain when the agent has gained weight. In the latter, there might be a time tloss where the agent has lost weight. Because these represent two possibilities, it might not make sense to ask whether tloss is before or after tgain. The times simply can't be compared. McDermott (1982) outlines such a model in detail and discusses its application to AI reasoning problems.

Another common temporal representation, variants of which are found in philosophy, linguistics (e.g. tense logic, Prior (1967)) and theoretical computer science (e.g. dynamic logic, Pratt, 1978, Harel, 1979), involve modal "tense" operators. Rather than having explicit time in the logic, modal operators such a *PAST* are introduced. Thus the formula *PAST(Green(Frog1))* means that *FROG1* was green sometime in the past. A similar operator can be defined for Future, and in models that use discrete time, an operator *NEXT* can be defined as well. For example, *NEXT(Green(Frog1))* would be true only if *FROG1* was green at the next time step. While this logics have been useful in certain applications, such as the representation of tense in natural language, and in the formal semantics of programs, they have not been found to be generally useful in AI. In particular, without an explicit representation of time, they do not provide a fine-grained enough representation to capture complex situations, external events, the changing world over time, or other issues important in AI applications.

V. Temporal Knowledge Representation Systems

Given this discussion, it remains to be seen how a temporal reasoner such as the ones described above can be embedded into an actual reasoning system. In some cases, the representation is restricted enough that the temporal reasoning can be incorporated directly. For example, consider a database application where all relations are homogeneous predicates, and time intervals are represented by pairs of dates, or pseudo-dates. The temporal reasoning is then built into the retrieval mechanism: to retrieve a relation R between dates d1 and d2, the system uses its normal retrieval mechanism to find candidate matches - say R between d3 and d4, and then checks if d3 <= d1 and d4>= d2. If so the retrieval succeeds. A similar technique would be used for a variant on the retrieval, namely to find any instances of relation R during the interval between d1 and d2. The only difference would be the temporal constraint check at the end of the retrieval process.

In many AI applications, we would like to integrate a temporal reasoner into a general purpose reasoning system. Here we briefly describe such a system that has been developed at Rochester. RHET (Allen & Miller, 1989) is a Horn-clause based AI representation language that has as a subcomponent the TIMELOGIC temporal reasoning system developed by Koomen (1989) and based on Allen's interval logic. RHET is a hybrid system, rather than using a single uniform proof technique, each predicate defined in RHET could potentially use its own specialized techniques for computing its truthhood. To the user, however, each predicate appears to be represented uniformly. Thus the Horn-clause

    [P ?x ?y] < [Q ?x] [R ?x ?y]

has its usual interpretation: To prove [P ?x ?y], we must first prove [Q ?x] and then prove [R ?x ?y], where ?x and ?y are variables and unification is used throughout the proof process.

In RHET, however, a predicate may be proved by calling a specialized reasoner rather than simply recursively matching into the Horn-clause database as done in PROLOG. In particular, all temporal relations are interpreted through a special predicate [TimeReln ?t1 ?r ?t2], where ?t1 and ?t2 must be times and ?r is a list of the possible temporal relationships between ?t1 and ?t2. If all three variables are bound, then the truth of this formula is computed simply by directly inspecting the temporal database. If one or more of the variables are not bound, then TIMELOGIC is called to find all possible bindings of the variable. RHET subsequently stores this list of values and iterates through them as needed throughout the rest of the proof (i.e. it binds the variable to the first value, and if the proof ever backtracks to this predicate again, it uses the next value, and so on until all values are tried). To the user, it appears that the [TimeReln ?t1 ?r ?t2] literal is being proven by successive matches into the RHET database just as it would in a normal Horn-clause proof. When a literal of the form [TimeReln ?t1 ?r ?t2] is asserted, RHET intercepts the operation and passes the information to TIMELOGIC and the constraint propagation algorithm is run to compute the consequences of the new information. This simple technique provides a conceptually simple, but powerful, temporal logic reasoning system. All other aspects of the temporal logic can be encoded simply as Horn clauses. For instance, the predicate BEFORE can be defined simply as follows:

    [BEFORE ?t1 ?t2] < [TimeReln ?t1 :before ?t2]

and the predicate IN as

    [IN ?t1 ?t2] < [TimeReln ?t1 (:starts :during :ends :equal) ?t2].

Given these definitions, a predicate HOLDS for homogeneous properties can be defined as followed, where ?p is a variable over such properties:

[HOLDS ?p ?t] < [HOLDS ?p ?t1] [IN ?t ?t1].

Koomen (1989) and Allen & Miller (1989) contains more details on the implementation of this system.

More general interfaces between a specialized temporal reasoner and a theorem prover can be obtained using the techniques of theory resolution (Stickle, 1985). Miller and Schubert (1990) use this technique in their system.

VI. The Limits of Deductive Temporal Models

A significant number of applications can be covered by the techniques described so far. But there are still some very difficult problems remaining. One of the most crucial issues relating to time is the persistence problem. Given that we know that some property P holds over some time interval T1, what can we say about P being true after T1? Logically, we can say nothing. If my car was in the driveway when I arrived home tonight, I can't know for sure that the car is still there. It might have been stolen, for instance. Nevertheless, for the purposes of my everyday reasoning, I would like to assume that it is still there. I do not want to have to go check again and again if the car is still there as I plan to make a shopping trip in the next hour. Such assumptions are called *persistence assumptions* because the reflect that the world generally remains the same from one moment to the next.

Any technique that uses a persistence assumption to conclude that a given proposition is true must be non-monotonic - i.e. it may make conclusions that it may later have to retract as further information is obtained. Thus I may conclude my car is still in the driveway, but will have to revise this assumption if I receive a call from the police saying they just found my car in a ditch somewhere. In general, non-monotonic persistence rules are of the form: If I know *P* is true at time *t*, and it is consistent that *P* is still true now, then I may assume that *P* still holds. Dean & McDermott (1987) use such a technique to make persistence assumptions into the future: if *P* started to be true somewhere between times *t1* and *t2*, and is not known to become false before time *t3*, where *t3* is after *t2*, then their system can assume that *P* is still true at time *t3* if persistence assumptions are allowed. The only requirement is that *P* must definitely be known to have become true prior to the time *t3*. Allen (Allen et al, 1990) defines a more general persistence rule. If we know that P is true over some interval *T1*, and we are interested in whether it could be true over interval *T2*. If it is consistent that *T1=T2* given the known temporal constraints, then we can assume *P* is true by persistence. In cases where *T1* could not possibly start after *T2* starts, this is equivalent to Dean & McDermott's technique. In cases where the relationship between the start of the two intervals is not known, Allen allows the assumption while Dean & McDermott do not. These techniques remain quite crude because they depend heavily on assumptions based on consistency checks. For instance, they do not apply well to situations where the agent only has limited knowledge about the world, for in these situations many contradictory conclusions can be consistent. More plausible models are being developed that deal with the probability that a given fact remains true given that it is true over some interval (e.g. see Dean & Kawazawa, 1988, Weber, 1989).

VII. Conclusion

The representation of time is crucial to many applications of artificial intelligence, and work in computational models of time is still in its early stages. Given the lower-bound complexity results for various forms of temporal inference, however, we know that there are not significantly better general techniques possible than those that have been developed so far. However, within particular applications, efficient complete techniques might yet be developed. In this paper, I have given a brief description of the most common techniques and attempted to identify the conditions under which they are applicable. The result is a wide range of different techniques, each effective under differing basic assumptions.

Acknowledgements

Many thanks to Josh Tenenberg and Lou Hoebel for their helpful comments on earlier drafts. This work was supported in part by ONR/DARPA grant N00014-80-C-0193 and ONR grant N00014-80-C-0197.

References

Allen, J. (1983) Maintaining Knowledge About Temporal Intervals, *Communications of the ACM* 26(11) pp 832-843.

Allen, J. (1984) Towards a General Theory of Action and Time, *Artificial Intelligence* 23(2), pp 123-15

Allen, J. & Hayes, P. (1989) Moments and Points in an Interval-based Temporal Logic, *Computational Intelligence*, 5, pp225-238.

Allen, J, Kautz, H, Pelavin, R and Tenenberg, J. (1990) *Reasoning about Plans*, Morgan-Kaufman.

Allen, J. and Miller, B. (1989) *The RHET System,* TR 325, Dept. of Computer Science, University of Rochester.

Dean, T and Kanazawa, K. (1988) Probabilistic Temporal Reasoning, *Proc. of the National Conference of the American Association for Artificial Intelligence* (AAAI-88), Morgan-Kaufman.

Dean, T. and McDermott, D.(1987) Temporal data base management, *Artificial Intelligence*, 32, pp1-55

Dowty, D (1986) The effects of aspectual class on the Temporal Structure of Discourse, *Linguistics and Philosophy* 9, p37-61.

Harel, D. (1979) *First-Order Dynamic Logic*, Springer-Verlag, NY.

Koomen, H. Localizing Temporal Constraint Propagation, *Proc. Principles of The First International Conference on Knowledge Representation* (KR89), Morgan-Kaufman, 1989.

Ladkin, P. Models of Axioms for Time Intervals, (1987) *Proc. of the National Conference of the American Association for Artificial Intelligence* (AAAI-87), Morgan-Kaufman.

Ladkin, P. and Maddux, (1988) R. Representation and Reasoning with Convex Time Intervals, TR KES.U.88.2, Kestrel Institute, Palo Alto, Ca.

McDermott, D. (1982) A temporal logic for reasoning about processes and plans, *Cognitive Science* 6, pp101-155

Miller, S and Schubert, L. (1990) Time Revisited, *Computational Intelligence*, 6:108-118.

Pratt, V. (1978) *Six Lectures on Dynamic Logic*, Tech. Report MIT/LCS/TM-117, MIT.

Prior, A.N. (1967) *Past, Preset and Future*, Oxford University Press.

Shoham, Y. (1987) Temporal Logics in AI: Semantical and Ontological Considerations, *Artificial Intelligence* 33, pp89-104.

Stickel, M.(1985) Automated Deduction by Theory Resolution, *Proc. International Joint Conference on Artificial Intelligence* (IJCAI-85), Morgan-Kaufman. pp1181-1186

van Beek, Peter (1989) Approximate Algorithms for Temporal reasoning, *Proc. International Joint Conference on Artificial Intelligence* (IJCAI-89), Morgan-Kaufman.

van Benthem, J. (1983) *The Logic of Time*, Reidel.

Vendler, Z. (1957) Verbs and Times, *Philosophical Review* 66. pp143-160.

Vilain, M. and Kautz, H. (1986) Constraint Propagation Algorithms for Temporal Reasoning, *Proc. of the National Conference of the American Association for Artificial Intelligence* (AAAI-86), Morgan-Kaufman.

Weber, J. (1989) A Parallel Algorithm for Statistical Refinement and its use in Causal reasoning, *Proc. International Joint Conference on Artificial Intelligence (*IJCAI-89), Morgan-Kaufman.