# Exploiting Domain Structure to Achieve Efficient Temporal Reasoning

Mike Williamson and Steve Hanks*
Department of Computer Science and Engineering
University of Washington,
Seattle, WA 98195
{mikew,hanks}@cs.washington.edu

## Abstract

We take temporal reasoning to be the problem of maintaining a set of constraints between time points and/or intervals, and responding to queries about the temporal separation between those individuals. Formal investigations of this constraint-satisfaction problem have demonstrated tradeoffs between the expressive power of the constraint language and the time required to answer queries. A simple constraint language admits an algorithm cubic in the number of individuals; allowing unrestricted disjunctive constraints makes the algorithm exponential. The problem is that applications of temporal reasoning, e.g. plan projection, need both disjunctive constraints and an algorithm much faster than $0(n^3)$.

It is significant, however, that the nature of the constraints added by and the queries posed by an application tend to be structured and predictable. Our solution to the problem is to exploit the structure of the application domain to provide fast responses to typical queries.

We consider the problem of plan projection under uncertainty and build a temporal representation—hierarchical interval constraints (HIC)—that allows appropriate disjunctive constraints. We then implement the HIC representation in a temporal-reasoning module, and test it using a plan-projection application. Applying the HIC module to a simple temporal projection problem shows orders-of-magnitude improvement over running the same projector using current implementations of domain-independent temporal constraint propagators.

## 1 Introduction

Temporal reasoning (TR) has become identified in the literature, e.g. [Kautz and Ladkin, 1991], [Dechter et a/., 1991], [Meiri, 1991], as a domain-independent problem of constraint satisfaction: given (1) a set of temporal individuals (time points and/or intervals), (2) explicit constraints on their temporal separation, and (3) implicit constraints like the interval transitivity relations of Allen, 1983] or the transitivity properties of Euclidean metric distance, compute the tightest bounds on the separation of any two individuals. [Dean and McDermott 1987], [Koomen, 1988], and [Kautz and Ladkin, 1991] report on implementations of these domain-independent algorithms.

There have been two predominant approaches to temporal reasoning. Interval-based systems, [Allen, 1983] take intervals to be the fundamental temporal entities, and allow the specification of qualitative constraints that hold between intervals. Systems based on time points [Dean and McDermott, 1987; Dechter et a/., 1991], on the other hand, take instants in time (points) to be the fundamental temporal entities, and allow both qualitative and metric constraints among those individuals (Constraints specify that the temporal distance between two time points falls within some particular interval.) There has been a significant amount of work comparing these two approaches, and also some recent work showing how the two frameworks can be unified [Vilain et a/., 1989; Kautz and Ladkin, 1991; Meiri, 1991].

Analysis of these systems reveals a clear tradeoff between the expressive power of the constraint language and the worst-case running time of the constraint propagation algorithm, tradeoffs that hold regardless of the underlying representation. Detailed analysis of these results is beyond the scope of this paper, but shows that "simple" constraints[1] over either a time-point or time-interval-based system admit an $0(n^3)$ solution algorithm, where n is the number of temporal individuals (points or intervals). Simple constraints cannot represent certain disjunctive information, however, and algorithms for processing sets of unrestricted disjunctive constraints require time exponential in n in the worst case.

*This work was supported in part by NSF grants IRI-9008670 and IRI-9206733 and by an NSF Graduate Research Fellowship. Thanks to Dan Weld for commenting on an earlier draft

[1]See [Dechter et a/., 199l] and [Vilain et al., 1989] for a precise definition of simple constraints. [Davis, 1987] analyzes this tradeoff as well. We defer a detailed comparison of these systems to [Williamson and Hanks, 1993].

Many application areas in AI do some sort of temporal *causal* reasoning: reasoning about changes in propositions, the occurrence of events, and so on. Planning, temporal projection, motivation analysis, and qualitative simulation are prominent examples. Causal reasoning in turn requires temporal reasoning, to keep track of the durations of events, the time at which propositions change state, and so on.

These formal results about temporal-reasoning algorithms are troublesome to programs that do causal reasoning. Causal reasoning almost always involves some sort of disjunction, as discussions in [Hanks, 1990a], [Dechter *et* al., 1991], and [Weld and de Kleer, 1989, Chapter 2] demonstrate, so simple temporal constraints will not generally suffice.

Moreover, the application program may generate great numbers of temporal individuals, and will call the TR module many times in the course of doing higher-level causal reasoning. Experiments in [Hanks, 1990b] show that projecting a plan of roughly 25 steps generates about 400 time points, and that about 65% of the projector's time is spent performing temporal-reasoning tasks. Even the cubic TR algorithm will be unacceptable for most applications.

We must conclude, then, that domain-independent or "uninformed" temporal reasoning is too slow to support reasonable application-program performance. In this paper we advance the idea that the application domain can provide the "information" that allows temporal reasoning to be efficient The temporal constraints imposed by an application, and the queries it typically poses to the TR module, are structured and regular. A TR module can exploit this structure and these regularities and by doing so can provide appropriate functionality while delivering acceptable performance.

We demonstrate our approach by considering a particular causal reasoning problem, a simplified version of plan projection under uncertainty [Hanks, 1990a] and [Hanks, 1990b]. We define formally the plan language used by the projector and a corresponding structured representation for the temporal information it generates The representation, called *hierarchical interval constraints* (HIC), combines time point and interval information in a nested format that, mirrors the structure of the projector's plans. We implement a temporal-constraint propagator using the HIC representation, then run experiments by instantiating a simple projection algorithm using our HIC' propagator, Dean's [1987] time-map manager, and Kautz and Ladkin's [I99I] MATS system as subroutines. The HIC system runs faster by orders of magnitude; furthermore, its performance degrades more slowly as problem size increases.

## 2 The Plan Projection Problem

[Hanks, 1990a] discusses the problem of projecting totally ordered sequences of actions under uncertainty. Uncertainty about the state of the world or the effects of an action means that an action sequence (plan) may have many possible outcomes. The projector builds a *scenario tree*[1], which is a temporal trace of the plan's execution. Each path through the tree, a *chronicle,* is one possible, internally coherent course of execution. The projection problem is to control branching in the scenario tree while still being able extract useful information about the plan's effects.

We will simplify the projection problem somewhat:

- Instead of the probabilistic representation in [Hanks, 1990a] we will use a three-valued truth assignment *true, false, unknown* for each proposition. A world state then consists of propositions along with their truth assignments. Any proposition not *true* or *false* is assumed to be *unknown.*
- A *simple action* consists of a name and a set of *outcomes.* Each outcome is a pair, *(condition, effect).* The condition describes the world states in which the corresponding effect will be realized—a condition is a set of propositions. The corresponding effect consists of a *duration* (a time interval) and a set of proposition/truth-value pairs.

  The idea is that if, at execution time, all of a condition's propositions are *true,* then all of the effect's propositions will take on the corresponding truth value We require that an action's conditions be mutually exclusive and exhaustive, so at execution time exactly one outcome will be realized. The projector may not know which one, however, since one or more of the action's conditions may have truth value *unknown.* Projecting a simple action will cause a branch in the scenario tree each time more than one of an action's outcomes has a condition that evaluates to *true* or *unknown.*
- From the simple actions we define *complex actions* representing sequential, parallel, conditional, and iterated execution of simple actions[3].

  If $a_1$, $a_2$, $a_n$ are actions and P is a proposition, then the following are actions too:

  - (seq $a_1$, $a_2$, . . . , $a_n$)
  - (par $a_1$, $a_2$, • , $a_n$)
  - (if P $a_1$ $a_2$)
  - (while P $a_1$)

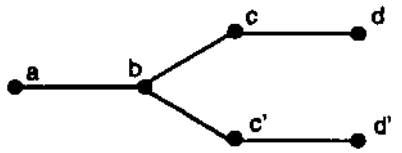
The projection algorithm is presented with an initial world state and an action. It projects the action fully, branching every time a subaction has multiple possible outcomes.

Consider, for example, projecting a plan such as (seq *a b c d).* Suppose that the projector can identify a single outcome for each action except c,[4] which has two possible outcomes. Figure 1(a) shows what the scenario tree would look like. The nodes in the tree represent the outcomes of the primitive actions in the plan. There are two chronicles, representing the two possible courses of execution.

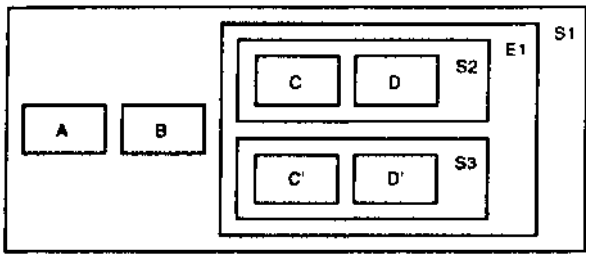
[2]This structure is similar to an *environment graph,* [Davis, 191)0, Chapter 4].

[3][Shaw, 11)81)] uses similar program-description operators.

[4]That is, exactly one condition for a, 6, and *d* evaluates to *true.*




(a) Scenario tree




(b) HIC representation

Figure 1: Projection Example

The projector may then have to answer questions both about the states of propositions at various points of time—"is P true when action A is executed?" "is G true when the plan finishes executing?"—and also temporal queries like "how long will the plan take to execute?" and "will this segment of the plan take more than 15 minutes?" We address queries only of the latter type, and furthermore, will concentrate mainly on temporal queries involving the distance separating time points within the same chronicle

We will next describe the Hierarchical Interval Constraint structure, which is an attempt to mimic the sorts of temporal structures generated by the projector

## 3 Hierarchical Interval Constraints

A hierarchical interval constraint (HIC) represents an interval of time, and may comprise one or more suhiutervals. Subintervals, if any, are also represented by HICs. The interval's duration—its minimum and maximum temporal length—depends on the two sorts of information: the durations of its subintervals, and the interval's *type.* HIC types represent common temporal reasoning structures, some of which cannot be represented as simple temporal constraint graphs

Formally, a HIC is a 3-tuple $(\tau, \sigma, \delta)$, where

- $\tau \in \{simple, sequence, selection, parallel\}$ is the HIC *type*,
- $\sigma = (I_1, I_2, \ldots)$ is a list of *sub-HICs* (or *children*), and
- $\delta = [\delta_l, \delta_h]$ is the temporal distance (duration) constraint denoted by the HIC.

(We may refer to the type, children, and duration of an arbitrary interval $I$ as $I_\tau$, $I_\sigma$, and $I_\delta$ respectively.)

In the case when $\tau = simple$, the list $\sigma$ must be empty, and the application must supply the duration $\delta$. The duration for a *complex* HIC (one whose type is not *simple*) is calculated according to the HIC's type and the durations of its children. See Table 1 for the specific definition

| $\tau$ | $\delta_l$ | $\delta_h$ |
|---|---|---|
| *sequence* | $\sum_{I \in \sigma} I_{\delta_l}$ | $\sum_{I \in \sigma} I_{\delta_h}$ |
| *selection* | $\min_{I \in \sigma} I_{\delta_l}$ | $\max_{I \in \sigma} I_{\delta_h}$ |
| *parallel* | $\max_{I \in \sigma} I_{\delta_l}$ | $\max_{I \in \sigma} I_{\delta_h}$ |

Table 1: How $\delta$ is calculated for complex HICs

of $\delta$ for each HIC type. A HICs duration $\delta$ is computed increment ally by the system when it is defined, a process requiring time linear in the number of its children.

For example, imagine that we wish to reason about an action go-to-work, which will be carried out by performing one of two possible subactions: either by walk-to-work, taking from 30 to 40 minutes, or bike-to-work, taking from 10 to 15 minutes. We create *simple* HICs *W* and *B* to represent the last two actions, and specify $W_\delta$ = [30,40] and $B_\delta$ = [10,15]. We then create a *selection* HIC, G, to represent the go-to-work action and specify that $G_\sigma$ = *(B, W).* The system will calculate that $G_\delta$ [10,40].

The *descendants* of a HIC are its children, all their children, etc. (The terms *parent* and *ancestor* also have the obvious definition.) A HIC must have at most one parent, and no HIC may be a descendant of itself; i.e. HICs must be properly nested. A *top-level* HIC is one that has no parent A leaf HIC has no children. Leaves must be type *simple.*

Our temporal reasoning module reasons about isolated time points in addition to these HIC structures. Time points may be useful to represent the occurrence of exogenous events, or to synchronize the plan's execution with known clock times Time points are integrated with HIC' structures to form an arbitrary temporal constraint graph (TCG), with time points as the vertices and top-level HICs as the edges If all of the HICs were of type *simple* and *sequence,* the expressive power of the system would be equivalent to a network consisting of simple temporal constraints as described in [Dechter *et* al., 1901]. This is also equivalent to the constraints allowed by the TMM system [Dean and McDermott, 1987].

The additional HIC types, *selection* and *parallel,* significantly increase the expressive power of the system. They allow an application to represent and reason efficiently about commonly occurring temporal situations which previously existing formalisms could only accommodate by resorting to more general, exponential-time algorithms An example of this is the go-to-work situation described above. Simple temporal constraint networks (in the technical sense of [Dechter *et* al., 1991]) cannot represent this situation at all. General temporal constraint networks can find the more precise solution $G_\delta = [10, 15] \cup [30, 40]$, but are NP-Hard to solve.

Temporal queries request the distance between two time points, which may be the beginning or end of any hierarchical interval (known as *virtual* time points), or one of the isolated time points in the TCG.

We compute the temporal distance between two time points as follows:

- If the two time points are both endpoints of HICs that share some common ancestor we compute the distance from each time point to each end point of that common ancestor. Although this requires time exponential $(0(2^d))$ in the depth of nesting of the HICs, this is still at most linear in the number of descendants of the intervals' common ancestor[5].
- If the two time points are not end points of HICs that share a common ancestor, or if either of them is an isolated time point in the TCG, we have no choice but to propagate bounds through the entire graph. We use the Floyd-Warshall transitive closure algorithm, which is $0(n^3)$ in the number of explicit time points, but independent of the number of HICs. (The algorithm requires only the lengths of the top-level HICs, which have been pre-computed.)

[Williamson and Hanks, 1993] provides a complete discussion of the query algorithm and its computational complexity.

Our algorithm's computational advantage derives from the fact that it allows the application to represent a temporal reasoning situation using the HIC framework almost exclusively—only a small number of explicit time points are typically required. In fact most queries do not require solution of the solution the TCG at all, and even when the entire graph must be considered it is much smaller than it would be if the same situation had been represented using existing, uninformed TR systems.

By letting the application provide information about the structure of the TR task, HICs effectively allow the query algorithm to ignore constraints irrelevant to the particular query. As a striking example, we shall consider our plan projection application, which does not require the construction of a TCG at all, but represents the projection scenario in a single nested HIC structure

### 3.1 HICs in Plan Projection

HIC types *simple, sequence, selection,* and *parallel* directly support the plan projection task. A *simple* HIC represents a single, atomic action. It has no sub-intervals, and its temporal length (minimum and maximum) is supplied by the application. A *sequence* HIC represents two situations: a compound action composed of a number of sub-actions executed sequentially, as well as a series of adjacent actions each with a single outcome. In other words, a scenario tree with a single branch appears as a *sequence* interval constraint.

*Selection* HICs represent actions that have more than one possible outcome due to uncertainty: A sub-HIC is created for each possible outcome.

As an example, consider projection of the plan mentioned above, which gave rise to the chronicle tree in Figure 1(a). Figure 1(b) shows the corresponding HIC structure. The smallest boxes are *simple* intervals representing the outcomes of each primitive action. E1 is a *selection* HIC representing the fork in the scenario tree. The fact that it is a *selection* HIC codes the information that exactly one of the branches will actually be realized at execution time. S1, S2 and S3 are *sequence* HICs, SI representing the temporal interval over which the entire plan is executed.

Queries to the temporal-reasoning module might ask about the amount of time separating any two time points: the beginning or end of any pair of the structure's intervals. Note again that neither the initial projection task nor queries involving the plans steps require construction of a temporal constraint graph. We represent the entire scenario tree in a single, nested HIC structure, meaning that temporal queries will require at worst time linear in the number of time points. In many cases the time required will be significantly less.

An example of the increased expressiveness of our system is that the duration of the entire plan can be obtained with a single temporal query. Existing implementations based on simple temporal constraint networks would require the application to minimize and maximize over the durations of each individual chronicle to obtain the same information.

## 4 Empirical Results

We implemented our projector on top of three different TR modules: our own HIC system, the Metric/Allen Time System (MATS) of Kautz and Ladkin [1991], and the temporal reasoning component of Dean's [1987] TMM All systems were implemented in Common Lisp, and tests were run on the same machine under similar loading We will first describe how we implemented the projector using the other two temporal-reasoning modules, then report the comparative results.

TMM's temporal reasoning facility essentially builds a graph of time points constrained by simple interval metric constraints. TMM allows no disjunctive constraints. We represented the projected plan's scenario tree by creating a pair of time points to represent the beginning and ending of each plan outcome, which were then constrained by the associated action's duration. The end point of each outcome and the begin point(s) of the following outcome(s) were constrained to be consecutive by adding the constraint [0,0] between them.

The MATS system integrates metric information into Allen's framework of qualitative interval relationships. It allows the creation of temporal intervals, specification of their qualitative relationships, and specification of interval-valued metric constraints between the end points of intervals. We represented the scenario tree by creating an interval for each action outcome. The duration of such outcome's interval once again has the application-supplied duration. Intervals for successive outcomes are constrained to *meet* each other.

For each implementation we projected plans that generated scenario trees of various sizes. We then performed 100 random temporal queries on the resulting temporal structures The plans were all less than ten steps long. We changed the number of possible outcomes in the scenario tree by varying the number of possible outcomes for the first action. In effect this introduced uncertainty early in the projection. The number of initial branches varied between one and eight; doubling the number of

[5]Assuming that each non-leaf HIC has two or more children.

initial branches essentially doubles the size of the resulting scenario tree.

We formed a temporal query by randomly choosing a chronicle from the tree, choosing two outcomes from that chronicle, and then choosing one of the end points of each of those outcomes. Table 2 shows the CPU time (in milliseconds on a DECstation 5000) required for projection and query processing.

Note that the query time required by the MATS system is cubic in the size of the scenario tree, making it completely impractical for larger problems. (MATS also requires space quadratic in the size of the tree ) MATS spends almost all of its query-processing time in computing the solution for the entire TCG, which is then cached. MATS would therefore take essentially the same amount of time to process one query as it did to process all 100 queries. TMM and HIC, on the other hand, compute query answers incrementally, and would therefore process a single query roughly 100 times faster

TMM's temporal reasoning mechanism required only linear time to answer queries, but seemed to require quadratic time to construct the constraint graph initially, we discuss TMM's propagation mechanism below

Our HIC system required only linear time for projection, and performed queries in sub-linear time This is possible because the constraint network's structure allows the query algorithm to isolate a particular chronicle and ignore the (irrelevant) constraints in the others.

These results suggest that our system is capable of handling projection problems several orders of magnitude larger than either of the alternative implementations.

## 5 Related Work

[Kautz and Ladkin, 199l] and [Dechter *et al*., 1991] both present metric TR algorithms that are completely domain independent. We noted above our opinion that these algorithms are mainly useful for pointing out explicitly the tradeoffs between expressive power and speed. We doubt that either algorithm, implemented without problem-dependent optimizations, will prove useful for applications of a reasonable size.6

[Allen, 1983] introduces the notion of a *reference interval*7 as a means of controlling temporal inference Each interval added to the database is assigned to one or more reference intervals, and the system caches the full interval transitivity table for all the intervals contained within a given reference interval. To infer the relationships between two intervals that do not share a reference interval the system searches for a path between the two intervals, restricting its search to reference intervals alone. Allen discusses the possibility of applying this idea to the problem of reasoning about events and processes, though he does not implement such a system. We provide an implementation, and furthermore extend the temporal representation to include quantitative as well as qualitative constraints.

The Timelo system of [Koomen, 1988) implements a temporal reasoning algorithm based on Allen's qualitative interval framework. It provides an algorithm for building reference-interval structures automatically. In some sense Koomen' approach is diametrically opposed to our own. Both approaches recognize that an application's constraints and queries exhibit regularities, and furthermore that exploiting these regularities is crucial to achieving good performance. In Koomen's system the application adds constraints *without* communicating that structure to the temporal database module; Timelogic rediscovers that structure by constructing an appropriate hierarchy of reference intervals. Our approach allows the application to communicate its temporal structure directly, which is then incorporated into the structure of the temporal database itself.

Dean's [1991] time-map manager (TMM) implements both causal and temporal reasoning patterns. Its temporal-reasoning mechanism constructs graphs whose nodes are time points connected by simple metric constraints It implements a heuristic, limited-depth search to infer the mst restrictive constraints binding any two points in the graph. Underlying this scheme is the assumption that paths through the constraint graph will be short The search can be quite efficient if the assumption holds, but if it is violated—if the graph contains long paths or is highly connected—-its performance can degrade to worse than the standard $On^3$ algorithms. And since the algorithm abandons paths that exceed a constant bound, it can potentially report incorrect results.

A second notable feature of TMM is its temporal indexing scheme. This technique, which is also similar to reference intervals, allows the application to provide information about the granularity of the temporal reasoning problem so that time points can be indexed for more efficient retrieval This functionality is an important step toward the kind of informed temporal reasoning necessary for realistic applications.

Our work also builds on the work in [Hanks, 1987], which was an early attempt to optimize a temporal database manager to perform tasks like temporal inferencing, temporally scoped database fetches, and hypothetical recisoning about the future, under conditions of uncertainty and ignorance.

## 6 Conclusion

Formal explorations of temporal-reasoning algorithms have made the tradeoffs between expressive power and computational efficiency extremely clear. The results point out that an application cannot use a domain-independent temporal constraint propagator and expect reasonable performance: reasonable functionality comes at the price of an exponential algorithm, and the algorithms are too slow in the worst case even under the most restrictive limitations.

Our paper has shown how to build an efficient temporal reasoning module by exploiting regularities both in the constraints typically added by the application and

6On the other hand, the MATS implementation available through [Kautz and Ladkin, 199l] is extremely elegant and easy to use, and so provides a nice vehicle for exploring small problems.

7Allen in turn acknowledges [Kahn and Gorry, 1977].

| Scenario Size | HIC | | TMM | | MATS | |
|---|---|---|---|---|---|---|
| | Project | Query | Project | Query | Project | Query |
| 25 | 33 | 366 | 834 | 11984 | 133 | 4583 |
| 51 | 134 | 1033 | 3101 | 27067 | 350 | 34133 |
| 101 | 200 | 1050 | 12633 | 64983 | 1050 | 256250 |
| 201 | 434 | 1233 | 61717 | 173584 | — | — |

Table 2: CPU time (in msec) for projection and 100 temporal queries

in the queries typically posed. We considered the problem of plan projection, and advanced a formal model of the problem's constraint structure. A temporal projector using an implementation of our model dramatically outperformed current domain-independent implementations in absolute terms, and furthermore showed better performance degradation as problem size grew.

It remains to be seen the extent to which our H1C model for temporal reasoning can be applied to other application domains, or whether a class of similar models can be developed for other applications. It is clear, however, that if a temporal-reasoning system is to be of practical value to a realistic implementation, it will have to have some model of its applications1 behavior Identifying a set of HIC-like structures offers a promising means of providing functionality midway between truly domain-independent temporal reasoning and problem-specific programs.

## References


[Allen, 1983] J. Allen. Maintaining Knowledge About Temporal Intervals CACM, 2(>( 11):832 843, 11)83 Reprinted in [Weld and de Kleer, 1989]

[Davis, 1987] Ernest Davis Constr ami propagation with interval labels. Artificial luttllujt-net, 32(3) 281 331,1987

[Davis, 1990] E. Davis. Representations of Commonsense Knowledge. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1990.

[Dean and McDermott, 1987] T. Dean and 1) McDermott. Temporal data base management Artificial Intelligence, 32(1), April 1987.

[Dean, 1991] Thomas Dean Using temporal hierarchies to efficiently maintain large temporal databases Journal of the ACM, 1991.

[Dechter ct ai, 1991] R. Dechter, 1. Mein, and J Pearl Temporal Constraint Networks. Artificial Intelligent , 49, May 1991.

[Hanks, 1987] Steve Hanks. Temporal reasoning about uncertain worlds. In Proceedings, Uncertainty in Artificial Intelligence, pages 114-122, 1987.

[Hanks, 1990a] S. Hanks. Practical Temporal Projection. In Proceedings of AAAI-90, August 1990.

[Hanks, 1990b] S. Hanks. Projecting PI ans about Uncertain Worlds. Ph.D. Thesis, Yale University Computer Science Department, January 1990.

[Kahn and Corry, 1977] Kenneth Kahn and G Anthony Gorry. Mechanizing temporal knowledge Artificial Intelligence, 9(2):87 108, 1977.

[Kautz and Ladkin, 199l] H. A. Kautz and P. B. Ladkm Integrating Metric and Qualitative Temporal Reasoning In Proceedings of AAA1-91, July 1991.

[Koonien, 1988] Johannes Koomen The TIMELOGIC Temporal Reasoning System Technical Report 231, University of Rochester, Department of Computer Science, October 1988.

[Mem, 199l] 1 Mein Combining Qualitative and Quantitative Constraints in Temporal Reasoning. In Procttdings of AAAI-91, July 1991.

[Shaw, 1989] Alan C. Shaw. Reasoning about time in higher-level language software. IEEE Transactions on Software Engineering, 15(7):875-889, July 1989.

[Vilain el al., 1989] M. Vilain, H. Kautz, and P. van Beek Constraint propagation Algorithms for Temporal reasoning: A Revised Report. In Readings in Qualitative Reasoning about Physical Systems, chapter 4, pages 373-381 Morgan Kaufmann, San Mateo, CA, 1989.

[Weld and de Kleer, 1989] D Weld and J. de Kleer, editors Readings in Qualitative Reasoning about Physical Systems. Morgan Kaufmann, San Mateo, CA, August 1989

[Williamson and Hanks, 1993] Mike Williamson and Steve Hanks Informed temporal reasoning. Technical report, University of Washington, Department of Computer Science, 1993 Forthcoming.