N90-29906

AN EFFICIENT TEMPORAL LOGIC FOR ROBOTIC TASK PLANNING

Jeffrey M. Becker
Martin Marietta Astronautics Group
P.O. Box 179, M.S. 4372
Denver, CO 80201

ABSTRACT

Computations required for temporal reasoning can be prohibitively expensive if fully general representations are used. Overly simple representations, such as a totally ordered sequence of time points, are inadequate for use in a nonlinear task planning system. This work identifies a middle ground which is general enough to support a capable nonlinear task planner, but specialized enough that the system can support online task planning in real time. A Temporal Logic System (TLS) was developed during the Martin Marietta Intelligent Task Automation (ITA) project to support robotic task planning. TLS is also used within the ITA system to support plan execution, monitoring, and exception handling.

1. INTRODUCTION

Most task planning systems that have been developed to date have represented the change in the truth of propositions over time within the representation used for plans [Chapman84], [Wilkins84], [Sacerdoti77], [Sacerdoti73]. Some systems have also represented temporal durations within the plan representation [Vere83]. By using a temporal logic system to support planner development the planner itself becomes conceptually much simpler. Temporal truth maintenance and duration constraint issues can be separated from planning issues. Using a temporal logic system also simplifies state projection for simultaneous planning and execution and other temporal representation problems related to plan execution. The temporal logic system described here is used within the Intelligent Task Automation (ITA) system developed at Martin Marietta. An overview of the ITA system is given in [Becker87].

To be able to formulate plans, a planner must be able to represent the effects of the proposed actions that constitute a (partial) plan. Many facilities for representing the effects of actions can be provided in a temporal logic system, but not every conceivable facility is needed to support planning. Temporal logic systems provide two primary functions: reasoning about duration constraint relations, and reasoning about the persistence of facts - also referred to as *temporal truth maintenance* [Dean87]. The combination of duration constraint relations and logical assertions is a kind of database referred to as a *time map* after [McDermott82]. Mechanisms must be provided to define time points, time intervals and the relationships between them, and to associate facts with times.

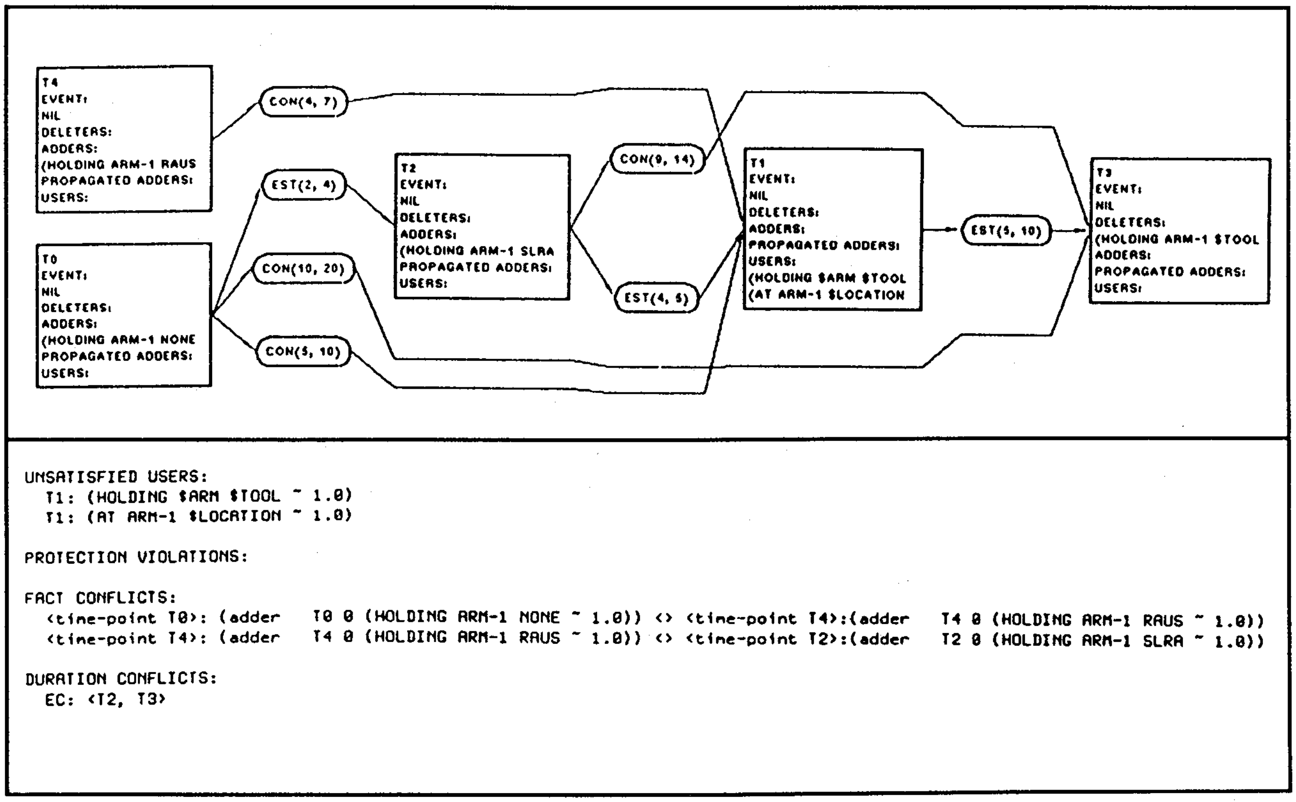
Time intervals may be specified qualitatively or quantitatively. For many planning problems quantitative duration information is needed so a quantitative representation is used in TLS. Typical operations on durations include consistency checking and deriving implied temporal relationships between time points that are not directly connected by a user-specified time interval.

This work was performed at the Intelligent Task Automation project facilities of Martin Marietta Information and Communications Systems. This work was supported by the Air Force Wright Aeronautical Laboratories and the Defense Advanced Research Projects Agency under contract F33615-82-C-5139.

The primary concern in temporal truth maintenance is managing the *persistence* of facts. A fact indexed in the time map persists until it is *clipped* at a later time in the time map. Two additional mechanisms of particular importance to planning are *protections* [Sussman 75] and *floating queries*. If a fact is protected at a certain time (or over a given time interval) then the temporal logic system will flag a warning if it becomes untrue. Floating queries are similar - if the query pattern is not matched by a fact at the time point where the query is indexed, then a flag is raised. Backward and forward chaining inference mechanisms can also be provided to support reasoning about facts asserted in the time map. It may also be desirable to represent alternative futures.

2.0 TEMPORAL LOGIC SYSTEM FEATURES

A time map in TLS (*Figure 2.0-1*) consists of time points, time intervals, assertions, and inconsistency records. Assertions can be declarations, adders (facts), users (floating queries), deleters (floating retractions), or rules. Assertions are indexed to time points. Time is represented as a directed acyclic graph where nodes are time points and edges are time intervals. There are two distinguished time points in the time map: *always* and *now*. Rules and declarations are indexed at *always*. *Now* is the default time used for assertions and queries. Interval durations are represented numerically by a maximum and minimum value and can indicate either an estimate or a constraint. Inconsistency records are kept for duration constraint violations, fact conflicts, unsatisfied users, and protection violations. TLS is implemented in Common Lisp and is used by invoking Common Lisp functions. TLS functions can be procedurally attached so Common Lisp functions can be called from within TLS rules as well.



```
UNSATISFIED USERS:
  T1: (HOLDING $ARM $TOOL ~ 1.0)
  T1: (AT ARM-1 $LOCATION ~ 1.0)

PROTECTION VIOLATIONS:

FACT CONFLICTS:
  <time-point T0>: (adder   T0 0 (HOLDING ARM-1 NONE ~ 1.0)) <> <time-point T4>:(adder   T4 0 (HOLDING ARM-1 RAUS ~ 1.0))
  <time-point T4>: (adder   T4 0 (HOLDING ARM-1 RAUS ~ 1.0)) <> <time-point T2>:(adder   T2 0 (HOLDING ARM-1 SLRA ~ 1.0))

DURATION CONFLICTS:
  EC: <T2, T3>
```

Figure 2.0-1: A Typical Time Map.

2.1 Logic System Language

The logic language supported by TLS consists of declarations, facts, and rules. Facts come in two flavors - functions and relations. A function, such as `color(object,red)`, can only have one value at a time. The last symbol in a function pattern represents its value. This is used within TLS for persistence clipping and to identify conflicting assertions. For example, the assertion `color(object,red)` would be clipped by `color(object,blue)` at a later time point, perhaps the result of a painting operation. This rule does not apply to relations such as `brother(Ed,John)`. Relations are handled internally as functions with boolean values, e.g. `brother(Ed,John,T)`. TLS provides syntactic sugar so that the boolean value may be omitted by the user when entering relation patterns.

Each functor (first symbol in a fact pattern) must be declared. The declaration specifies whether the functor denotes a function or a relation, the type of each argument, the value type, and procedural attachment bindings for procedures to be activated on assertions, retractions, and queries. The built-in functors include the logical connectives and a number of metalogical operations. Some example functor declarations from the domains of list and math operations are:

```
functor (member object list boolean)
functor (append list list list)
functor (+ number number number) :procedure +
```

The first example declares a predicate that could be defined using rules to determine if an object is a member of a list, the second defines a function that could be defined using rules to combine two lists into a third list. The third example declares the mathematical addition function to be procedurally attached to the Lisp + function. The Lisp function is called during queries to determine the sum of two numbers.

Besides regular fact assertions as provided in Prolog, which are called *adders* in TLS, TLS provides *user* assertions and *deleter* assertions. An adder assertion is propagated forward through the time map according to persistence rules. A user assertion is a query pattern that is indexed to a time point. Whenever the time map is modified, a record of whether the user is *satisfied* (unifies with some adder at the same time point) is updated. A user assertion can be *protected* so that a flag will be raised when it becomes unsatisfied. A deleter assertion is a retraction pattern that is indexed to a time point. A propagated adder that matches the deleter will be clipped by it. Neither users nor deleters are propagated. However, an unsatisfied user may have an associated *ghost adder* that is propagated to represent what might be true at later times in the time map if the user were satisfied. Confidence values can also be associated with adder facts and rules. Adder confidences are combined according to user-defined functions for disjunction, conjunction, and implication during forward and backward chaining.

Query operations accept a time parameter that can be a single time point or a set of time points. If a set of time points is given, then the user can specify whether the proof must hold at *every* time point in the given set, or at *some* time point. The theorem prover used in TLS uses a backward chaining algorithm as in Prolog. Two important extensions are provided. First, since facts have associated confidence values, the prover can keep track of the accumulated confidence associated with a branch of the proof search tree and prune that path if the confidence goes below some threshold. Second, relations can be declared `TRANSITIVE`, in which case the prover will check for and prune circularities. This allows the prover to handle rules for such things as transitive equality that would otherwise cause infinite loops in the proof. A limited form of forward chaining is also supported.

2.2 Duration Consistency Checking

A time map is a directed acyclic graph where nodes are time points and edges are time intervals. A time map is created by asserting the time points and time intervals of interest. Each time interval is associated with two time points - its start and end. There are two types of time intervals, estimated ("it takes 30 to 40 minutes to get to the ticket office") and constrained ("tickets will be available from 1 to 3 P.M. only"). Time intervals have a minimum and a maximum duration that is expressed numerically. The notation used is `EST(min,max)` for estimated intervals and `CON(min,max)` for constrained intervals.

Duration consistency checking is based on two operations on time intervals: serial composition, denoted by "&", and parallel composition, denoted by "||". Serial composition is the process of finding the most constraining interval to represent the combination of two intervals linked end to end. Parallel composition is the process of finding the most constraining interval to represent two intervals connected in parallel between the same endpoints, if a consistent composition exists. The rules for serial and parallel composition are given in *Figure 2.2-1* .

SERIAL COMPOSITION:

EST(min1,max1) & EST(min2,max2) -> EST ((min1 + min2), (max1 + max2))

CON(min1,max1) & CON(min2,max2) -> CON ((min1 + min2), (max1 + max2))

If ((max1 - min1) < (max2 - min2))
then EST(min1,max1) & CON(min2,max2) -> CON ((max1 + min2), (min1 + max2))
else EST(min1,max1) & CON(min2,max2) -> EST ((min1 + max2), (max1 + min2))

PARALLEL COMPOSITION:

EST(min1,max1) || EST(min2,max2) -> EST(min1,max1)
consistent when: min1 = max1 = min2 = max2

CON(min1,max1) || CON(min2,max2) -> CON (max (min1, min2), min (max1, max2))
consistent when: max (min1, min2) ≤ min (max1, max2)

EST(min1,max1) || CON(min2,max2) -> EST(min1,max1)
consistent when: min2 ≤ min1 ≤ max1 ≤ max2

Figure 2.2-1: Duration Composition Rules

Figure 2.2-2 shows a simple time map with a duration conflict due to two parallel paths from T2 to T3. One path (a single interval) constrains the maximum duration to 14, but the other path composes to an estimate that the duration can be as long as 15. The estimates must be tightened or the constraint relaxed in order to resolve the conflict.

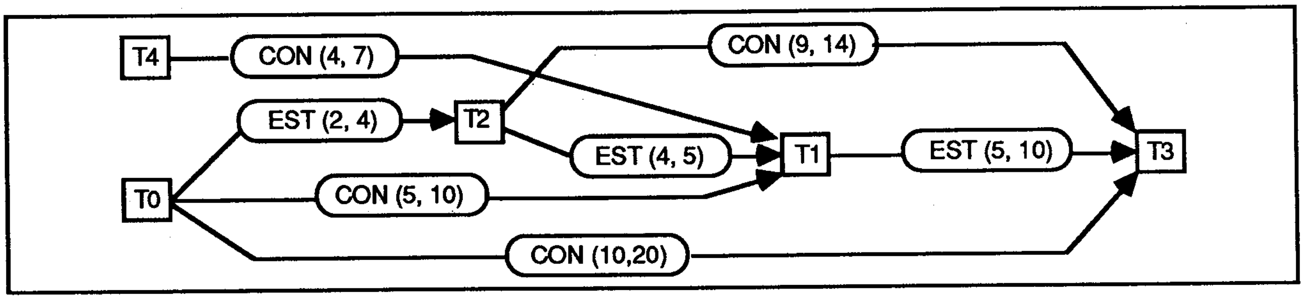


Figure 2.2-2: Duration Consistency Example

2.3 Managing Persistences

Adders are propagated through successor times in the time map until a time point with a conflicting assertion is reached. Adders can be clipped by other adders and by deleters, but not by users. Adder-adder clipping occurs when fact patterns have the same arguments, but different values. Adder-deleter clipping occurs when the patterns unify. For example, the adder `location(table,hall)` will be clipped by the deleter `location(table,$x)`, where "`$<symbol>`" denotes a pattern variable.

Two types of clipping can occur, serial and parallel. Serial clipping occurs when an adder propagates to a time point at which there is a conflicting assertion. Parallel clipping occurs when an adder propagates to a time point in parallel with a time point at which there is a conflicting assertion. The two types of clipping are illustrated in *Figure 2.3-1.*

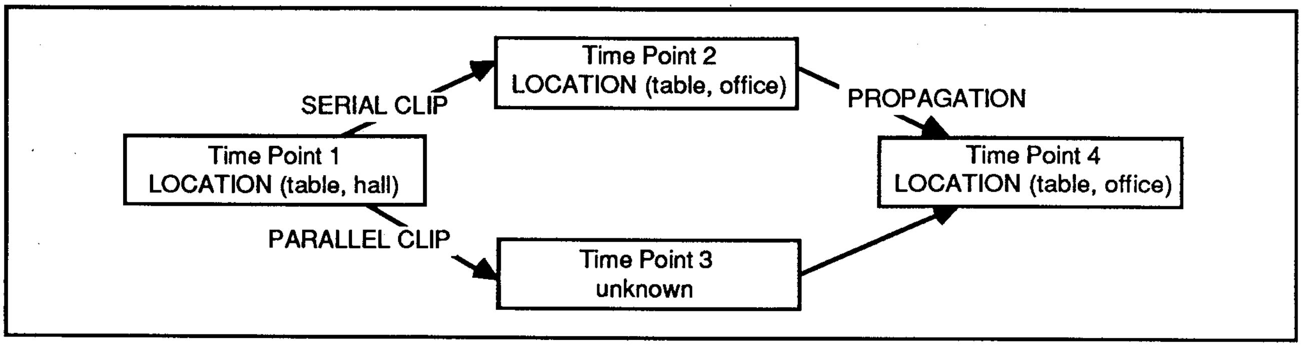


Figure 2.3-1: Clipping of Persistences

Fact conflicts can occur between assertions at the same or parallel time points. Fact conflicts never involve propagated adders because of the clipping mechanism. Fact conflicts can occur between any pair of adder, user, or deleter assertions. When a fact conflict involves an adder, it is depropagated (unindexed from all time points except the point it was originally asserted at) to make the time map reflect its uncertain status. When a fact conflict is found by the time map mechanism, it is recorded in a slot of the time map. *Figure 2.3-2* shows a time map with two fact conflicts, both involving the assertion `holding(arm-1,hammer)` of time point T4. Time point T4 must be ordered with respect to time points T0 and T2 in order to resolve the conflicts.

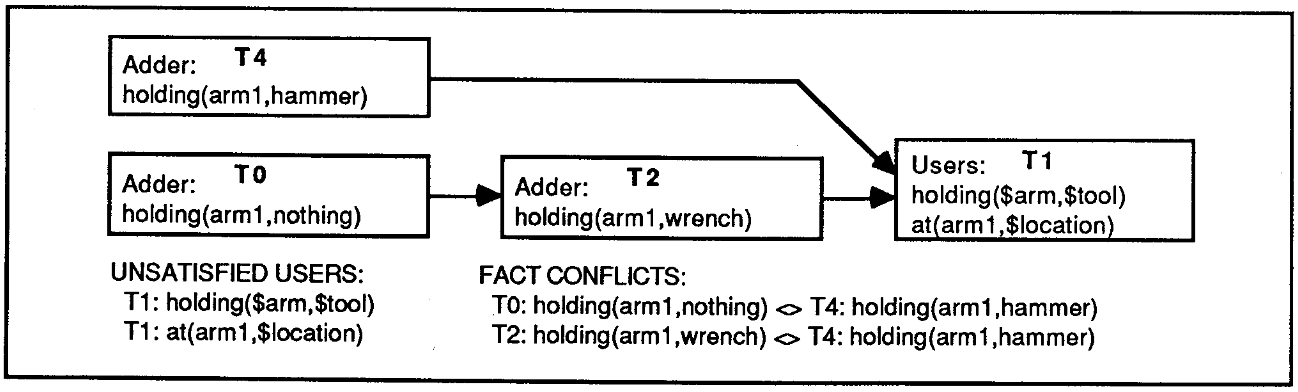


Figure 2.3-2: Time Map With Fact Conflicts

3.0 IMPLEMENTATION DETAILS

Key details for implementation of the capabilities described above will now be discussed. This discussion is not complete but does illustrate many aspects of the approach taken in the implementation of TLS.

3.1 Dynamic Sets

The foundation for the assertion indexing scheme and directed acyclic graph abstract data type used in TLS is an abstract data type called *Dynamic Sets*. Dynamic Sets are like the set data type defined in Pascal, except that the elements of a set type may be defined and changed as a program runs. This makes it possible to represent a set of assertions or a set of graph nodes. Set type operations include declaring a new type and adding and removing elements in the type. Set instance operations include creation, deallocation, and the usual boolean combiners (e.g. union, intersection, ...) and predicates (e.g. subset, empty-set?, ...). A set instance is a record structure with slots for: a pointer back to the set type descriptor, the current size (number of bits) of the set instance, and a bit vector for specifying members of the set instance. A set instance may be placed on an "active" list in which case it is automatically updated when an element is removed from the set type. The bit vectors of set instances are automatically grown as the number of elements in the set type increases.

3.2 Directed Acyclic Graphs

A directed acyclic graph (DAG) data type was defined to support operations on time maps such as interval installation and removal, finding predecessor, successor, and parallel time points, and traversals for assertion persistence computations. The DAG abstract data type is built on top of the Dynamic Sets abstract data type. Set operations are used instead of mark-and-sweep techniques for all graph operations. A graph node includes slots for: lists of immediate predecessors and successors of the node, sets of all predecessors and successors of the node, the node id, and data to be associated with the node. Updating the sets of all predecessors and successors of each node when the graph is modified requires a single traversal of the graph. Then, for example, the graph nodes between two ordered nodes can be determined by intersecting the set of all successors of the first node and the set of all predecessors of the second node. A mark-and-sweep algorithm would require marking the successors of the firt node and collecting the marked predecessors of the second node, i.e. two partial traversals. Since queries of this type far exceed graph modifications the net computation time savings is significant.

3.3 Assertion Indexing

The indexing scheme used in TLS is illustrated in *Figure 3.3-1*. Associated with each time point in the time map is a set of assertions - the assertions that are indexed at that time point. Each assertion is also indexed according to the elements of the pattern that represents it. Associated with each position of a pattern is a table of symbols and associated with each symbol is a set of assertions that contain that symbol in that position in some pattern. Assertions are also indexed in one of three sets according to their type: adder, user, or deleter.

Rules are indexed by their conclusion part. Finding all assertions that occur at every (some) time point in some set of time points is simply an intersection (union) operation. Finding all patterns that contain certain literals in certain positions is also done by intersecting the appropriate assertion sets. Pattern variables in assertions are also accounted for in the indexing scheme. All variables are mapped to a single symbol ($X). When looking up matches to a query pattern the union of the assertion set for data patterns with a variable in a particular pattern position and the assertion set for data patterns with the same literal in that position as the query pattern gives the set of assertions that match the query pattern at that position. The intersection of these sets over all positions gives the set of patterns that match the query pattern. This match is not equivalent to unification since variable binding consistency is not checked so unification test generally follows the initial lookup.

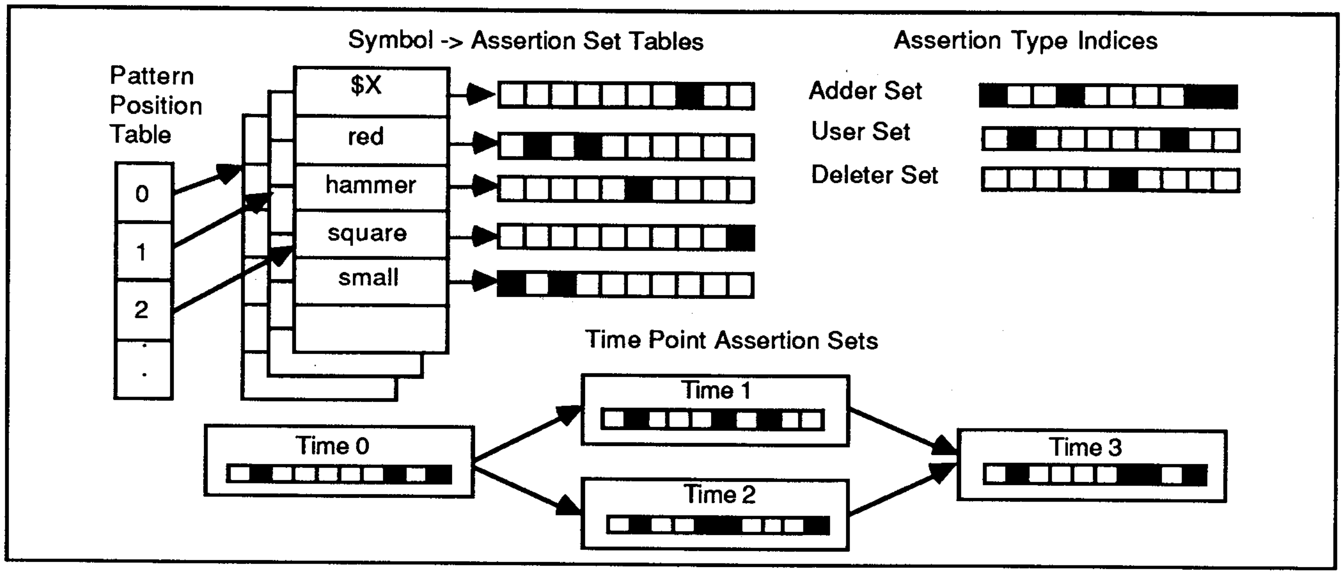


Figure 3.3-1: TLS Indexing Scheme

3.4 Duration Consistency Checking

Duration consistency checking is performed using a dynamic programming approach. This computation is non-incremental so it is possible to make many changes to the time map before checking for duration consistency. The first step is to partition the time map by removing redundant time intervals (shortcut constraint intervals with duration zero to infinity) then removing intervals with no parallel neighbors. For each remaining subgraph, paths of length 2 (two intervals) are built from paths of length 1 using serial composition. The most constrained path (MCP) of length 2 is then found between each pair of predecessor/successor time points using parallel composition. Paths of length 3 are constructed from MCPs of length 1 and 2, and MCPs of length 3 are found, and so on. This reduction builds new data structures rather than actually modifying the time map so no constraint information is lost in the process. Duration conflicts found according to the parallel composition rules are flagged on a slot on the time map.

4. EXAMPLE APPLICATION: THE ITA TASK PLANNER

Task level planning and plan execution functions in the ITA system use TLS as their knowledge representation substrate. As shown in *Figure 4.0-1*, the planner adds new states to a projected future view of the state of the world as a plan is constructed for a given goal. When a plan is completed, it is decomposed into a set of commands which are added to pending command queues along with additional synchronization steps. After execution of a command is completed, an entry is added to a separate history time map. Using this scheme, planning can occur concurrently with execution. If an exception occurs, planning stops and is resumed only after the exception handler has modified the current state model to correspond to the actual current state.

The task planner is a hierarchical, nonlinear planner in the same family as TWEAK [Chapman84], Nonlin [Tate77], and SIPE [Wilkins84]. It is described in more detail in [Garrett88]. When the task level of the controller receives a new set of goal conditions start and end time points for the plan to be constructed are installed in the time map after the end time point of the previous plan. The expected state of the world is automatically projected by the time map mechanisms. The goal conditions are installed as *users* at the end time point of the plan. Any goals not satisfied by conditions at the start time point of the plan will now appear on the unsatisfied users slot of the time map. First, a check is made of the conflict slots of the time map. Any conflict at this point indicates that conflicting goals were given to the planner, so the planner fails without doing any work.

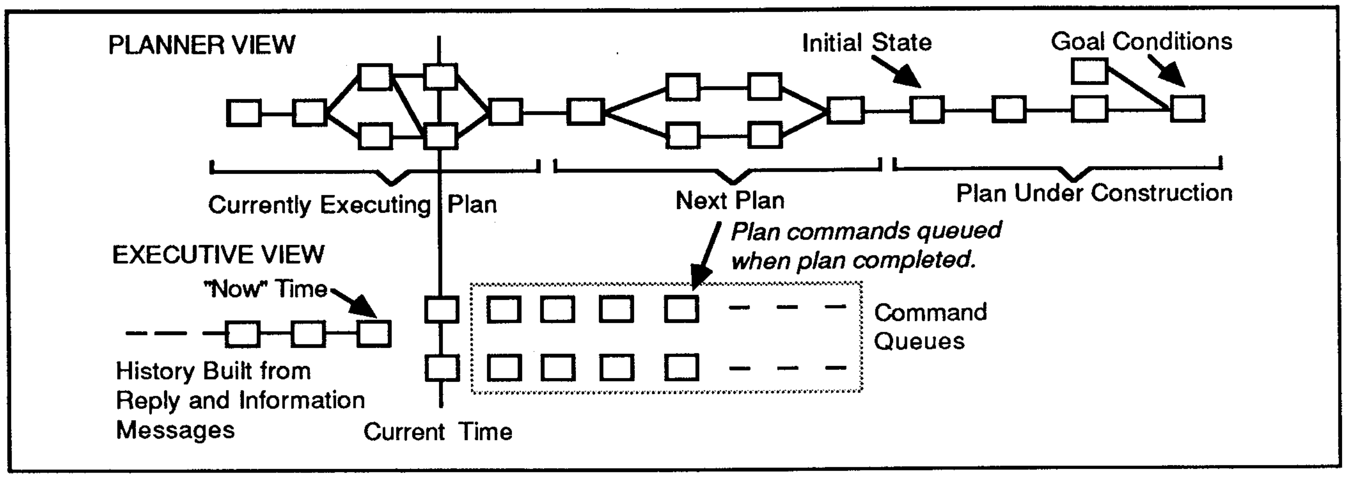


Figure 4.0-1: Use of Time Map for Planning and Plan Execution

The task planner finds the set of highest level unsatisfied goals and then finds the set of operators that can plan for any of the goals. The operators are ranked according to user-provided heuristics and the "best" operator is installed in the time map by creating a time interval that represents the start, end, and duration of the operator. Additional intervals are asserted so that the operator is constrained to occur within the plan and before all operators whose subgoals it plans for. Goals that the new operator plans for are protected so that the time map will indicate when those goals are *clobbered.* Goals that are satisfied but not explicitly planned for are not protected. When these goals are clobbered they are said to be *reactivated.* Preconditions of the new operator are installed as *user* assertions in the time map at the start time point of the operator. Postconditions are installed as *adder* and *deleter* assertions in the time map. If the fact conflict slot of the time map is not empty, the planner uses heuristics to choose a preferred ordering and installs additional intervals in the time map. Backtracking choices are recorded for alternative operators and orderings. Backtracking occurs when a planned-for goal is clobbered, or when no satisfactory operator or ordering can be found.

5. RELATED WORK

Much of the work done on temporal reasoning has been concerned with computing closures of temporal relations expressed in a qualitative interval logic [Allen83], [Vilain86], [Ladkin88]. However, TLS is closest in spirit to the Time Map Manager (TMM) system described in [Dean87]. Durations in TLS are represented numerically rather than qualitatively as in TMM. For many real world problems, such as travel planning, the ability to represent durations numerically is a necessity. With a numerical representation consistency checking can be performed without computing the closure of implied duration relationships between all time points thus avoiding exponential computation time. The consistency checking mechanism used in TLS is based on a dynamic programming rather than a propagation approach as in TMM. An advantage of the dynamic programming approach is that many changes can be made to the time map before checking the constraints again. This can result in a savings in computation time. In addition, TLS supports *estimates* as well as constraints in its representation of duration. In [Brooks82] a similar though more powerful mechanism is applied in the domain of geometric error analysis for robot planning.

Assertions in TLS are indexed to time points as in TWEAK [Chapman84] rather than to time intervals as in TMM - this is primarily an implementation detail since the two methods are conceptually equivalent. The modal truth criterion defined by Chapman is not fully supported by either TMM or TLS since neither supports representation of *possible* persistence of assertions. The deductive proof mechanism used in TLS is similar to that used in Prolog [Clocksin84] with extensions for proof subtree pruning based on recurring goals in the proof tree as described in [Smith85]. TMM provides more complex mechanisms to support temporal imagery than TLS which only provides simple forward chaining. Some ideas about inheritance between theories used in TLS are based on experience with MRS [Genesereth84].

6. CONCLUSIONS

A temporal logic system that was implemented to support task level planning and plan execution in a hierarchical robot controller has been described. The system is efficient enough to support online real-time planning. Some features have been omitted that might be desirable for supporting other applications. Careful consideration is being given to possible extensions that could be implemented without significantly degrading performance. Other applications of TLS are being investigated, such as causal modeling of electrical power distribution systems to support fault diagnosis. The availability of a temporal logic system provides not only a new way of thinking about how to build a planning system, but provides a new way of thinking about solving many different kinds of problems.

ACKNOWLEDGEMENTS

Special thanks to Fred Garett and Don Mathis for discussions that lead to a number of design decisions in TLS. Fred was also the first user of the system and uncovered many bugs that otherwise would have been missed. Thanks also to Dennis Haley (ITA Project Manager) and Roy Greunke (ITA Software Lead) for shielding us from distractions during the development of this system.

REFERENCES

[Allen83] Allen, J., and Kooman, J., "Planning Using a Temporal World Model", *Proceedings IJCAI-83*, pp. 741-747.

[Becker87] Becker, J., and Garrett, F., "An Architecture for Intelligent Task Automation", *Proceedings AAAI-87*, 1987, pp. 672-676.

[Brooks82] Brooks, R. A., *Symbolic Error Analysis and Robot Planning*, A.I. Memo No. 685, Massachusetts Institute of Technology Artificial Intelligence Laboratory, September, 1982.

[Chapman84] Chapman, D., *Planning for Conjunctive Goals*. MIT Technical Report 802, January, 1984.

[Clocksin84] Clocksin, W. F., and Mellish, C.S., *Programming in Prolog,* Springer, Berlin, 1984.

[Dean87] Dean, T., and McDermott, D., "Temporal Data Base Management", *Artificial Intelligence 32*, 1987, pp. 1-55.

[Garrett88] Garrett, F., and Becker, J., Task Level Planning in the Intelligent Task Automation System, publication pending - available on request from authors.

[Genesereth84] Genesereth, M., Greiner, R., Grinberg, M., and Smith, D., *The MRS Dictionary,* Heuristic Programming Project Report No. HPP-80-24, Stanford University, Stanford, CA, January 1984.

[Ladkin88] Ladkin, P. B., "Satifying First-Order Constraints About Time Intervals", *Proceedings AAAI-88*, 1988, pp. 512-517.

[McDermott82] McDermott, D.V., "A Temporal Logic for Reasoning about Processes and Plans", *Cognitive Science 6*, 1982, pp. 101-155.

[Sacerdoti73] Sacerdoti, E., "Planning in a Hierarchy of Abstraction Spaces", *Proceedings IJCAI-73*, 1973, pp. 412-422.

[Sacerdoti77] Sacerdoti, E., *A Structure for Plans and Behavior,* North-Holland, New York, 1977.

[Smith85] Smith, D. E., Genesereth, M. R., and Ginsberg, M. L., *Controlling Recursive Inference*, Report No. STAN-CS-85-1063 (also numbered HPP-84-6), Stanford University, Stanford, CA, June 1985.

[Sussman75] Sussman, G. J., *A Computer Model of Skill Acquisition*, American Elsevier, New York, 1975.

[Tate77] Tate, A., "Generating Project Networks", *Proceedings IJCAI-77,* 1977, pp. 888-893.

[Vere83] Vere, S., "Planning in Time: Windows and Durations for Activities and Goals", *IEEE Trans. Pattern Anal. Mach. Intell. 5*, 1983, 246-267.

[Vilain86] Vilain, M., and Kautz, H., "Constraint Propagation Algorithms for Temporal Reasoning", *Proceedings AAAI-86*, 1986, pp. 377-382.

[Wilkins84] Wilkins, D., "Domain-Independent Planning: Representation and Plan Generation", *Artificial Intelligence 22*, 1984, pp. 269-301.