

# Temporal Deduction in a Graphical Logic \*

L. E. Moser, P. M. Melliar-Smith, Y. S. Ramakrishna, G. Kutty, L. K. Dillon  
Department of Electrical and Computer Engineering  
Department of Computer Science  
University of California, Santa Barbara

## Abstract

Graphical Interval Logic is a modal logic for reasoning about time in which the basic modality is the interval. The logic differs from other logics in that it has a natural intuitive graphical representation that resembles the timing diagrams drawn by system designers. We have implemented an automated deduction system for Graphical Interval Logic that includes a graphical user interface and an automated theorem prover. The graphical user interface enables the user to create Graphical Interval Logic formulas and proofs on a workstation display, and the theorem prover checks the validity of deductions in the logic. In this paper we describe the logic, the automated deduction system, and an application to robotics.

## 1 Introduction

Real systems, particularly those that involve temporal properties, often manifest great complexity in their design. The basic logic appropriate for reasoning about temporal properties, S4.3 modal logic [9], has been known for many years, but planners and systems designers find this logic unintuitive and difficult to use. If temporal logics are to become more widely used in planning and system design, a logic must be found that is easier to use and understand. Moreover, effective mechanical support must be provided for reasoning in the logic if the risk of errors in the design is to be reduced.

---

\*This research was supported in part by NSF/ARPA grant CCR-9014382.

Graphical Interval Logic is a modal temporal logic and, as such, is nonstandard, but it is also nonstandard in its graphical representation, which differs from the linear textual representation of other temporal logics. We believe that a graphical representation of time can be a substantial aid in understanding temporal properties. Graphical Interval Logic formulas resemble the “back-of-the-envelope” timing diagrams drawn by system designers; nonetheless, the logic is a formal mathematical logic based on S4.3 modal logic. These graphical formulas provide a pictorial representation of the relative ordering of events in a complex system. The logic also supports reasoning about real-time properties, expressed as constraints on the durations of intervals.

The automated deduction system we have implemented for Graphical Interval Logic has as its front-end a graphical user interface that enables the user to create Graphical Interval Logic formulas on a workstation display. The core of the automated deduction system is a theorem prover that checks the validity of deductions in the logic and generates counterexamples for invalid proofs.

## 2 Related Work

Temporal logic has been widely recognized as an appropriate formalism for reasoning about the relative ordering of events in complex systems [13]. Interval logics have been investigated by a number of researchers [1, 8, 14] as logics that provide higher-level temporal constructs. Recently, there has been much activity in the development of temporal logics to support reasoning about real-time constraints. Alur and Henzinger [3] have characterized the decidability of a variety of real-

time temporal logics. Real-time Graphical Interval Logic [18], like Metric Interval Temporal Logic [4], is one of the few known dense real-time temporal logics that is decidable.

Procedures for deciding the satisfiability of formulas in temporal logics have been implemented by many researchers, both as free-standing decision procedures and within general-purpose theorem provers. In [20] Wolper gave an overview of the tableau method for temporal logics, which is the basis of the decision procedure for Graphical Interval Logic. Niemelä and Tuominen [16] have developed the Helsinki Logic Machine, which is based on a tableau decision procedure for dynamic logic. Decision procedures for temporal logics are realized by translating temporal logic formulas into formulas of dynamic logic and using the decision procedure for dynamic logic.

The TIMELOGIC reasoning system of Koomen [11] is an interval-based forward chaining inference engine and database manager of temporal constraints. The TEMPOS subsystem, based on Allen's interval calculus [1], handles time as operations on intervals and relationships between intervals, rather than as values of predicates as in Graphical Interval Logic. TEMPOS is an extension of the Horn clause logic programming and knowledge representation system RHET, which provides a predicate interface to TIMELOGIC. RHET can be used either as an automated theorem proving system or as a frame-based representation system.

In [10] Kono described two theorem provers for Moszkowski's Interval Temporal Logic [14]. One of these theorem provers employs a deterministic tableau expansion and a binary decision tree representation of terms, while the other is based on the generation of an automaton and uses Binary Decision Diagrams. Interval Temporal Logic involves the next and chop operators and, thus, is more expressive than Graphical Interval Logic. To reduce the cost of the decision process, Kono has restricted the logic to finite intervals and to formulas without the chop operator in the scope of a negation.

Clarke [6] and others have implemented a variety of tools for model checking in Computa-

tion Tree Logic (CTL), a branching-time temporal logic. Model checking in CTL is computationally less expensive than satisfiability checking in linear-time temporal logics, such as Graphical Interval Logic. Unfortunately, the computational cost advantages of CTL model checking do not extend to general-purpose theorem proving. Recently, Burch, Clarke and others [5] have used a model checking algorithm for the Mu-Calculus based on Binary Decision Diagrams to obtain efficient decision procedures for CTL model checking and satisfiability of linear-time temporal logic formulas.

In [19] Schlör and Damm described a graphical specification language, called Timing Diagrams, and a system similar to ours. Specifications resemble traditional informal timing diagrams and are created using a graphical editor. The language is carefully restricted so that it can be compiled into a linear-time temporal logic related to CTL. Model checking for the logic uses computationally efficient CTL algorithms.

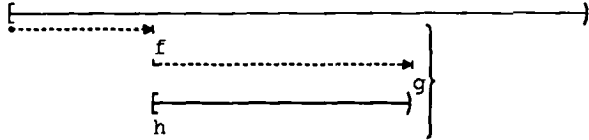
### 3 Graphical Interval Logic

The basic construct of Graphical Interval Logic is the interval, which provides a context within which properties are asserted to hold. An interval is a linear order of states which is discrete for the base logic and dense for the real-time extension. The progression of states in time is shown using the horizontal dimension. Graphical Interval Logic formulas are read from left to right and from top to bottom. The topmost interval in a Graphical Interval Logic formula represents the enclosing context, *i.e.* the entire computation.

Intervals are constructed by means of search operators, and properties are asserted to hold on the intervals so constructed. To construct an interval, two search patterns are specified: one to locate the left endpoint of the interval and the other to locate the right endpoint. A search pattern consists of a sequence of searches; the state located by one search is the state at which the next search begins. A search within a search pattern locates the first state at which its target formula holds. An interval begins with the state located by the

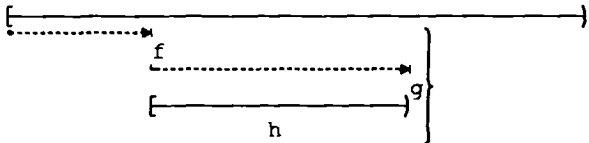
first search pattern and extends up to but does not include the state located by the second search pattern.

To assert that a formula holds at the first state in an interval, the formula is left-justified below the left endpoint of the interval. For example,



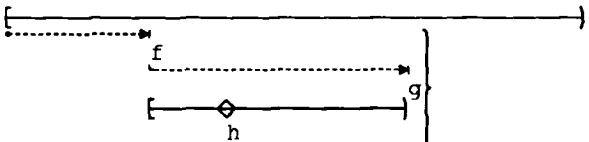
asserts that  $h$  holds at the first state in the interval that begins with the first state at which  $f$  holds and ends just prior to the next state at which  $g$  holds.

To express an invariant (henceforth) property that holds throughout an interval, the formula that is asserted to be invariant is positioned below the interval and indented to the right of the bracket that delimits its start. For example,



asserts that  $h$  holds at every state in the interval that begins with the first state at which  $f$  holds and extends up to, but does not include, the next state at which  $g$  holds.

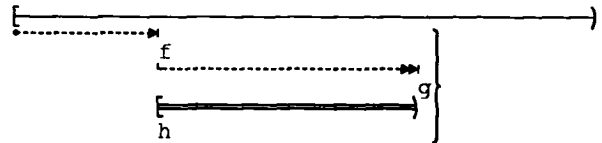
To express an eventuality property, a diamond  $\diamond$  is placed on the interval, with the eventuality property left-justified below the diamond. For example,



asserts that  $h$  holds at some state in the interval that begins with the first state at which  $f$  holds and extends up to, but does not include, the next state at which  $g$  holds. The subformulas  $f$ ,  $g$  and  $h$  in the above examples may be state predicates or more complex interval formulas.

If the target formula of a search does not hold at any state in the future of the state at which the search begins, the search to the formula fails. If either of the searches for the left or right endpoints of an interval fails or, if the state located by the search for the right endpoint coincides with the state located by the search for the left endpoint, the interval cannot be constructed. If the interval cannot be constructed, the interval formula holds vacuously.

The single-arrow searches and the single-line intervals in the previous examples are referred to as *weak* searches and *weak* intervals, respectively. The logic also provides as derived constructs *strong* searches, denoted by double arrows, and *strong* intervals, denoted by double lines as, for example, in



Here the strong search expresses the requirement that the search to  $g$  must not fail. More specifically, it requires that the search to  $g$  must succeed unless the weak search to  $f$  fails. The strong interval requires that the interval is non-empty provided that the searches to  $f$  and  $g$  do not fail.

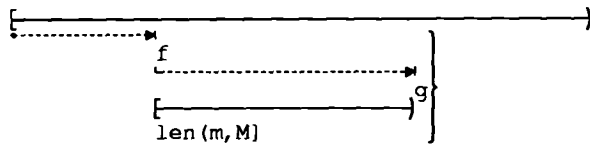
Formulas can be combined using standard logical infix operators laid out vertically. In vertical layout a conjunction is indicated by placing the formulas one below the other without the conjunction operator. Braces are used to disambiguate formulas.

### 3.1 Real-Time Extension

Like the Graphical Interval Logic introduced above, pure temporal logic makes no reference to time. Formulas in temporal logic define only invariants, eventualities, and ordering constraints. Much of the elegance of temporal logics is that the value of time and the quantification over time are hidden by the logic, which facilitates automatic processing by decision procedures. When we introduce real-time constraints into Graphical Inter-

val Logic, we wish to preserve this characteristic. Thus, we extend the logic to express durations of intervals rather than absolute times.

Graphical Interval Logic expresses real-time constraints by imposing real-time bounds on the lengths of intervals using the *len* predicate. Thus,



asserts that the length of the indicated interval, if it can be constructed, is greater than  $m$  time units and less than or equal to  $M$  time units, where  $m$  and  $M$  are non-negative rational constants. This simple construct appears to suffice for describing most real-time constraints directly and easily, while retaining decidability of the logic. Moreover, it does not permit the formulation of undesired expressions in which time is manipulated inappropriately.

## 4 Automated Deduction System

### 4.1 Graphical User Interface

We first briefly describe the graphical user interface that acts as a front-end to the theorem prover. The graphical user interface is a graphical editor that facilitates the interactive generation and manipulation of Graphical Interval Logic formulas. The graphical editor enables the user to draw formulas on a workstation display by invoking high-level editing operations based on the constructs of the logic. As a syntax-directed editor, it ensures that the formulas drawn by the user are syntactically correct.

The menu-and-button interface of the editor allows the user to create and edit graphical formulas and to compose them into more complex formulas. The editor provides capabilities for automatically replacing formulas with other formulas, resizing formulas to suit the context length, etc. In case a formula does not fit into the space allotted, an error is indicated by highlighting the formula. The user may then resize the context length or the

search arrows to enable the formula to be drawn correctly. All subformulas of the formula are automatically resized to scale.

The graphical syntax of the logic is defined by means of an attribute grammar that specifies the set of attributed syntax trees that correspond to “legal” formulas. These trees are used solely as internal representations; the user only interacts with the formulas in their graphical representation. Attributes within these trees give the relative sizes of search lines and interval contexts that allow the formulas to be drawn on the screen.

As a formula is constructed, the corresponding attributed syntax tree is built incrementally and the attributes defining the graphical representation of the formula are determined. Initially, the formulas generated by the editor contain unexpanded templates that must be expanded into legal formulas during subsequent editing. Each expansion of a template corresponds to a production in the grammar. The consistency of the attribute values for a tree constructed using the editor ensures that the formulas are syntactically correct.

The graphical editor has been implemented in Common Lisp using the Garnet graphics toolkit [15], which runs within the X window system. The editor uses Garnet functions to generate postscript output; all of the graphical formulas that appear in this paper were created with the graphical editor. Formulas produced using the graphical editor can be stored in and retrieved from a specification database held in Unix files.

### 4.2 Theorem Prover

Automated deduction in temporal logic generally has a higher complexity than automated deduction in propositional calculus because temporal logic subsumes propositional calculus. Our approach to controlling that complexity is to have the human work closely in conjunction with the theorem prover. Thus, the human creates the proof using the graphical user interface and then invokes the theorem prover to check the validity of the proof. It is the responsibility of the user to devise proof steps small enough so that the decision time of the theorem prover is not excessive.

To prove a theorem  $T$  from the premises  $P_1, \dots, P_n$ , the user invokes the theorem prover to check the validity of the formula  $P_1 \wedge \dots \wedge P_n \Rightarrow T$ . The theorem prover, which is a refutation prover, takes the negation of this formula and looks for a satisfying model. If no such satisfying model exists, the formula is valid and the purported proof is indeed a proof. If such a satisfying model is found, it is a counterexample to the formula and the purported proof fails. The user can then invoke the theorem prover to display the counterexample model to investigate why the proof is invalid.

The theorem prover exploits the fact that Graphical Interval Logic is decidable. As for most temporal logics, a decision procedure may be given as an automata-theoretic method, in which form it is presented in [17]. The decision procedure essentially consists of first reducing the formula into an "equivalent" Büchi automaton and then checking the emptiness of the language of the resulting automaton. The first step involves a double exponential blow-up, *i.e.* given a formula of size  $n$  and depth  $k$  of interval nesting, its Büchi automaton can have  $2^{O(n^k)}$  states. The second step is linear in the number of states of the automaton.

The decision procedure for the real-time extension of Graphical Interval Logic [18] reduces the decision problem for the extended logic to the emptiness problem for Timed Büchi Automata [2]. The decision procedure first constructs a Büchi Automaton and then annotates the states with duration formulas that involve the *len* predicate. Duration formulas allow the automaton's states to be augmented with active clock sets and its edges with clock setting and comparison actions that enforce the timing restrictions. The complexity of the extended decision procedure is  $2^{O(n^{2k} \cdot 2k \cdot \log n + n^{2k} + t \cdot \log t)}$ , where  $t$  is the size of the binary encoding of the largest constant appearing in the edge conditions of the Timed Büchi Automata.

The automata method for the logic without real-time has been refined into a more traditional tableau method [20] for the theorem prover. The tableau method produces a compact form of the automaton. Each node of the tableau represents

a set of states of the automaton; two such nodes are connected in the tableau if and only if there is a possible transition from any state in the first node to some state in the second node. By clustering states and transitions in this manner and exploring sets of states and transitions simultaneously, the tableau method achieves better time and space efficiency on average than the automata method.

The tableau method for the real-time extension of the logic is more difficult because of the denseness of the underlying time domain. As for the logic without real-time, the procedure begins by constructing a tableau and by pruning the tableau based on eventualities. A timed tableau is then constructed by adding timing constraints to the edges of the tableau using the duration formulas in the nodes of the tableau. Using Dill's algorithm [7], a region tableau is then constructed in which each node has an associated clock-region in place of the timing constraints on the edges of the timed tableau. This may eliminate possible traces from the original tableau because of timing constraints, and so additional pruning based on eventualities may be required. If at any stage the tableau becomes empty or the initial node is eliminated, the procedure terminates with a valid proof; otherwise, a counterexample corresponding to a timing consistent trace through the tableau can be extracted.

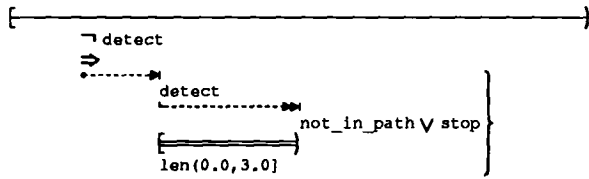
The theorem prover is a ground term prover that operates on Graphical Interval Logic formulas without quantification. Formulas that involve quantification must be explicitly quantified by the user. Recently, we have augmented the deduction system with a mechanism that allows the user to order two states in different formulas in a specific manner and to identify states in different formulas as being the same state. We are currently investigating the possibility of using the counterexample model to realign the states of the graphical formulas that constitute the invalid proof to assist the user further in correcting the fallacious reasoning.

The theorem prover has been implemented in Common Lisp. A proof maintenance system that supports the construction and management of formulas and proofs has also been implemented.

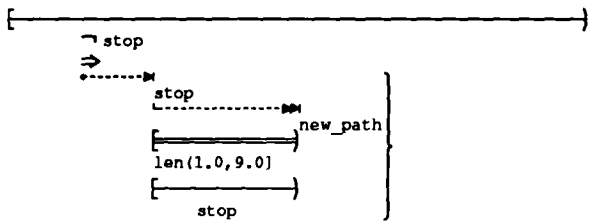
## 5 Application to Robotics

We now consider a simple application of the use of the logic and of the automated deduction system in specifying and proving properties about a robot. In each of these specifications, the indentation of the formula below the topmost interval indicates that the property being expressed is an invariant property.

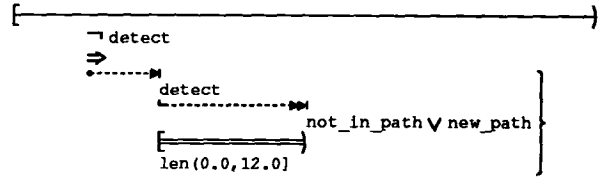
The first two specifications are implementation specifications. The first specification requires that if the robot detects an obstacle then either it will determine that the obstacle is not in its planned path or it will stop. Furthermore, the length of the interval from first detecting the obstacle to determining that the obstacle is not in its planned path or to stopping has positive duration but must not exceed 3.0 seconds.



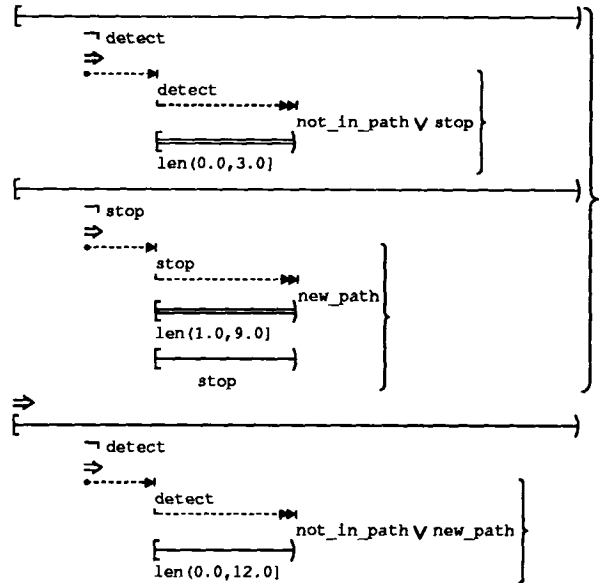
The second specification requires that if the robot stops then it remains stationary until it has decided on a new path. Moreover, the robot decides on a new path within an interval of length greater than 1.0 second and less than or equal to 9.0 seconds after stopping.



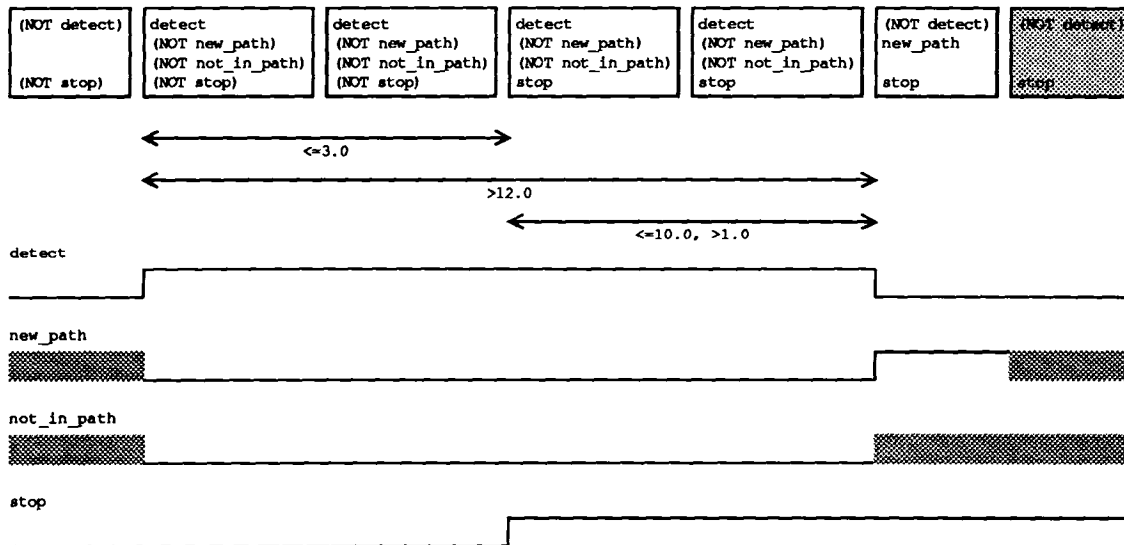
The third specification is a requirement specification; it is the theorem we wish to prove from the two implementation specifications. It states that if the robot detects an obstacle then, in 12.0 seconds or less, either it determines that the obstacle is not in its planned path or it determines a new path.



We now exhibit the proof that was validated by the theorem prover. The first two specifications are conjoined, represented by vertical composition, and imply the third.



The limit of 9.0 seconds in the second specification was then changed to 10.0 seconds, and the proof was found to be invalid. The theorem prover generated the counterexample shown on the top of the next page. Across the top of the representation is a sequence of states together with the values of the predicates in those states. If a predicate is not listed in a state, it can have either value true or false in that state. Below the sequence of states are the real-time constraints satisfied by the model along with the timing diagram for the predicates. Note that the interval from *detect* to *stop* is at most 3.0 seconds and from *stop* to *new\_path* is at most 10.0 seconds, but the interval from *detect* to *new\_path* is greater than 12.0 seconds, which contradicts the purported theorem.



## 6 Conclusion

Graphical Interval Logic supports reasoning about the relative ordering of events in complex systems and about the durations of intervals delimited by those events. The logic is a nonstandard logic in its interval modalities and its graphical representation of formulas. The graphical representation is intended to be more attractive and intuitive to planners and system designers than the linear textual representation of other temporal logics. The automated deduction system we have implemented provides a graphical user interface that enables the user to generate and manipulate graphical formulas on a workstation display. The system also provides an automated theorem prover that supports the user in validating deductions in the logic and in displaying counterexamples to invalid proofs. In this paper we have illustrated the use of the logic and of the automated deduction system with a simple application to robotics.

## References

- [1] J. F. Allen, "Maintaining knowledge about temporal intervals," *Communications of the ACM*, vol. 26, no. 11 (1983), pp. 832-843.
- [2] R. Alur and D. Dill, "Automata for modelling real-time systems," *Proceedings of the 17th International Conference on Automata Languages and Programming*, Warwick University, England (July 1990), Lecture Notes in Computer Science 443, Springer-Verlag, pp. 322-335.
- [3] R. Alur and T. Henzinger, "Real-time logics: Complexity and expressiveness," *Proceedings of the 5th IEEE Symposium on Logic in Computer Science*, Philadelphia, PA (June 1990), pp. 390-401.
- [4] R. Alur, T. Feder and R. Henzinger, "The benefits of relaxing punctuality," *Proceedings of the 10th ACM Conference on Principles of Distributed Computing*, Montreal, Canada (August 1991), pp. 139-152.
- [5] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill and L. J. Hwang, "Symbolic model checking:  $10^{20}$  states and beyond," *Information and Computation*, vol. 98 (1992), pp. 142-170.
- [6] E. M. Clarke, E. A. Emerson and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Transactions on Programming Languages and Systems*, vol. 8, no. 2 (April 1986), pp. 244-263.

- [7] D. L. Dill, "Timing assumptions and verification of finite-state concurrent systems," *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, France (June 1989), Lecture Notes in Computer Science 407, Springer-Verlag, pp. 196-212.
- [8] J. Halpern and Y. Shoham, "A propositional modal logic of time intervals," *Journal of the ACM*, vol. 38, no. 4 (October 1991), pp. 935-962.
- [9] G. E. Hughes and M. J. Creswell, *An Introduction to Modal Logic*, Methuen and Co. Ltd., London, England, 1968.
- [10] S. Kono, "Automatic verification of interval temporal logic," *Proceedings of 8th British Colloquium for Theoretical Computer Science* (March 1992).
- [11] J. A. G. M. Koomen, "The TIMELOGIC temporal reasoning system," Technical Report 231, Dept. of Computer Science, The University of Rochester, Rochester, NY (March 1989).
- [12] H. Lewis, "A logic of concrete time intervals," *Proceedings of the 5th IEEE Symposium on Logic in Computer Science*, Philadelphia, PA (June 1990), pp. 380-389.
- [13] Z. Manna and Z. Pnueli, *The temporal logic of reactive and concurrent systems: Specification*, Springer-Verlag, New York, NY (1992).
- [14] B. Moszkowski and Z. Manna, "Reasoning in interval temporal logic," *Proceedings of the Workshop on Logics of Programs*, Lecture Notes in Computer Science 164, Springer-Verlag, Pittsburgh, PA (June 1983), pp. 371-382.
- [15] B. A. Myers, D. A. Giuse, R. B. Danneberg, B. VanderZanden, D. S. Kosbie, E. Pervin, A. Mickish and P. Marchal, "Garnet: Comprehensive support for graphical, highly interactive user interfaces," *IEEE Computer* (November 1990), pp. 71-85.
- [16] I. Niemelä and H. Tuominen, "Helsinki Logic Machine: A system for logical expertise," Technical Report B1, Helsinki University of Technology, Digital Systems Laboratory, Espoo, Finland, 1987.
- [17] Y. S. Ramakrishna, L. K. Dillon, L. E. Moser, P. M. Melliar-Smith and G. Kutty, "An automata-theoretic decision procedure for future interval logic," *Proceedings of the Twelfth Conference on Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science 652, Springer-Verlag, New Delhi, India (December 1992), pp. 51-67.
- [18] Y. S. Ramakrishna, L. K. Dillon, L. E. Moser, P. M. Melliar-Smith and G. Kutty, "A real-time interval logic and its decision procedure," *Proceedings of the Thirteenth Conference on Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science, Springer-Verlag, Bombay, India (December 1993).
- [19] R. Schlör and W. Damm, "Specification and verification of system-level hardware designs using timing diagrams," *Proceedings of the European Conference on Design Automation*, Paris, France (February 1993), pp. 518-524.
- [20] P. Wolper, "The tableau method for temporal logic: An overview," *Logique et Analyse, Nouvelle Série 28<sup>e</sup> Année, 110-111 Numero Special, Automated Reasoning in Non-Classical Logic* (June-September 1985), pp. 119-136.