The Real-Time Graphical Interval Logic Toolset

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 93106

1 Introduction

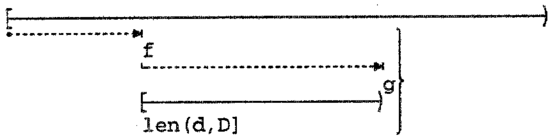
The tools that we have developed for Real-Time Graphical Interval Logic (RTGIL) are intended for specifying and reasoning about time-bounded safety and liveness properties of concurrent real-time systems. These tools include a syntax-directed editor that enables the user to construct graphical formulas on a workstation display, a theorem prover based on a decision procedure that checks the validity of attempted proofs and produces a counterexample if an attempted proof is invalid, and a proof management and database system that tracks proof dependencies and allows graphical formulas to be stored and retrieved.

2 Real-Time Graphical Interval Logic

RTGIL is a linear-time temporal logic in which formulas are interpreted on traces of states indexed by the non-negative real numbers. To exclude the occurrence of instantaneous states and Zeno runs, these traces are required to be right continuous and finitely variable. Right continuity requires that each primitive proposition holds its value for a non-zero duration, while finite variability ensures that there are only a finite number of state changes in any finite duration.

The key construct of RTGIL is the interval, which provides a context within which properties are asserted to hold. An interval is defined by two search patterns, which locate its left and right endpoints. A search pattern is a sequence of one or more searches. Each search locates the first state at which its target formula holds. The state located by one search is the state at which the next search begins. An interval is half-open in that it begins with the state located by the first of its two search patterns and extends up to but does not include the state located by the second search pattern. Once an interval is defined, properties can be asserted to hold on the interval, including initial, henceforth, and eventuality properties. Most importantly, real-time bounds on the duration of an interval can be specified.

For example, in the following RTGIL formula,



the interval begins with the first state at which the formula f holds and ends just prior to the next state at which the formula g holds. The duration of that interval is asserted to be greater than d time units and less than or equal to D time units.

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

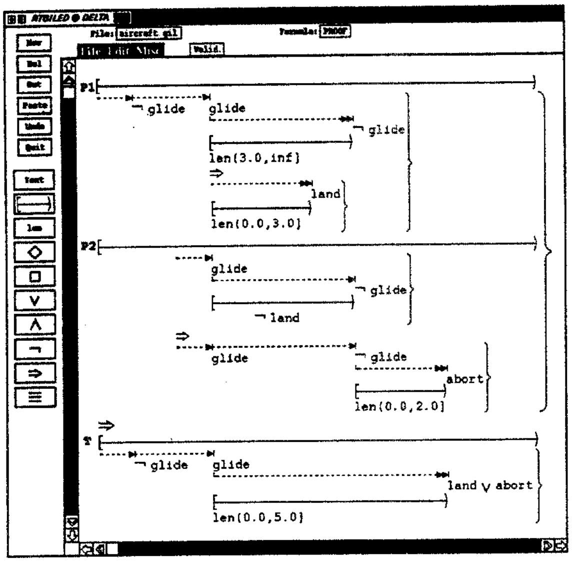


Fig. 1. The graphical user interface with a validated proof. The premises $P1$ and $P2$ are conjoined, represented by vertical composition, and imply the theorem T. Premise $P1$ requires that if the aircraft is in the glide path for more than 3.0 secs, then it will land within 3.0 secs of entering the glide path. Premise $P2$ requires that if the aircraft does not land before leaving the glide path, then it will abort within 2.0 secs of leaving the glide path. Theorem T states that within 5.0 secs of entering the glide path, the aircraft will either land or abort.

Formulas in RTGIL are read from top to bottom and from left to right, starting with the topmost interval. Formulas can be combined using standard logical infix operators laid out vertically. In vertical layout, a conjunction is indicated by stacking the formulas one below the other without the conjunction operator. Braces are used to disambiguate formulas.

3 The Graphical Editor

The graphical user interface to the RTGIL editor is shown in Fig. 1. The editor provides high-level editing operations and supplies templates containing boxes for formulas that enable the user to construct graphical formulas incrementally. The mouse enables the user to select a box or formula on the display and to highlight it.

The pull-down menus (File, Edit, Misc) at the top of the display contain commands for storing and retrieving formulas, for overriding the default layout of formulas, and for invoking the theorem prover. The buttons on the upper left (New, Del, Cut, Paste, etc) provide editing operations that enable the user to create a new formula, delete a selected formula, store a selected formula in a buffer, and subsequently insert that formula in a selected box. The buttons on the lower left (Text, $\vdash\!\!\dashv$, len, etc) enable the user to select an appropriate RTGIL construct to apply to the currently highlighted subformula. Scroll bars allow the user to view large formulas.

The editor provides capabilities for automatically replacing formulas with other formulas, resizing formulas to suit the context length, etc. If a formula does not fit into the allotted space, an error is indicated by highlighting the formula. The user can then resize the context length or the search arrows to allow the formula to be drawn correctly. All subformulas of the formula are automatically resized to scale.

The editor also enables the user to align corresponding points in the formulas that comprise a proof. The user can thus see how states in different formulas are ordered relative to one another, how intervals are aligned relative to each other, and how durations of intervals are related to satisfy real-time constraints. Alignment is helpful in constructing proofs and in debugging attempted proofs that are invalid.

4 The Theorem Prover

The RTGIL theorem prover is a satisfiability checker based on a decision procedure, rather than a Gentzen-style theorem prover based on inference rules. The decision procedure for RTGIL is given as an automata-theoretic method in [4]. The implementation, however, is a tableau-theoretic method that achieves better time and space efficiency, on average, than the automata-theoretic method. It employs the notion of timed tableau, the analogue of the timed automaton of Alur and Dill [1].

The user, working in the theory defined by his specifications and the underlying logic, creates theorems and proofs and submits the proofs to the decision procedure for validation. To prove a theorem T, the user selects a subset of the axioms and previously proved lemmas and theorems as the premises $P1, \ldots, PN$ of the proof. The editor displays the proof represented by the formula $P1 \wedge \ldots \wedge PN \Rightarrow T$ in its graphical form. The graphical representation is converted into a Lisp S-expression, is negated, and is then submitted to the decision procedure.

The decision procedure checks the satisfiability of the negated implication by building a tableau for that formula and checking the emptiness of the tableau. The procedure first constructs an untimed tableau for the formula and performs the standard eventuality-based pruning of the tableau. Using the duration formulas in the nodes of the remaining tableau, it then constructs a timed tableau by adding timing constraints to the edges of the untimed tableau. Timing consistency of the timed tableau is checked using Dill's algorithm [2]. This step may eliminate some possible traces from the original tableau because of timing restrictions and, consequently, a further round of eventuality-based pruning is required. If, at any stage, the tableau becomes empty or the initial node is eliminated, the negated implication is unsatisfiable and the attempted proof is valid. Otherwise, there exists a timing consistent trace through the final tableau that constitutes a counterexample to the attempted proof.

If the decision procedure determines that an attempted proof is invalid, the user can invoke the theorem prover to produce a counterexample by extracting a satisfying model for the negated implication from the tableau. The counterexample is displayed in an accompanying window, shown in Fig. 2, as a sequence of states and, additionally, as a timing diagram if the user selects that option. By associating the targets of the searches in the formulas of the proof with the states in the sequence at which the predicates become true or false, or the points in the timing diagram at which the signals rise and fall, the user can more readily discover the fallacy in the attempted proof and correct it.

The worst-case time complexity of the decision procedure is $2^{O(n^{2k} \cdot k \cdot \log n + t \cdot \log t)}$, where n is the number of logical connectives, k is the depth of interval nesting, and t is the size of the binary encoding of the largest duration constant in the formula.

5 The Proof Management and Database System

RTGIL formulas saved to disk are stored in a simple database consisting of Unix files. Several formulas can be stored in the same file by associating a unique name with each of them. The user can invoke the editor to display the names of the formulas in a file and also to load, add or delete a formula to or from a file. For each formula in a file, the user can invoke the proof manager to determine if a proof already exists, and to list the premises of an existing proof.

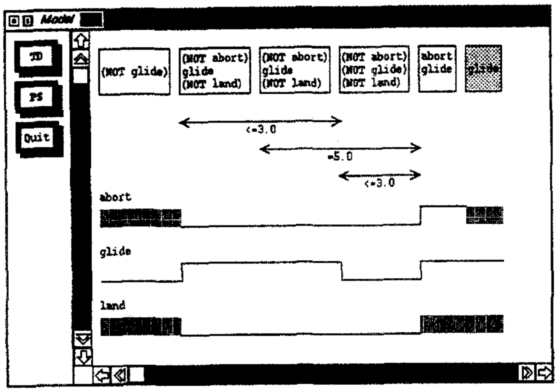


Fig. 2. The graphical user interface with a counterexample model. The proof in Fig. 1 is modified so that the upper bound of 2.0 secs in premise $P2$ is replaced by 3.0 secs. The attempted proof is invalid, and this counterexample is generated. Note that the interval from *glide* to $\neg$*glide* without an intervening *land* is at most 3.0 secs and the interval from $\neg$*glide* to *abort* is at most 3.0 secs. Thus, the interval from *glide* to *land* $\vee$ *abort* is at most 6.0 secs, which is greater than the 5.0 secs in theorem T.

If an attempted proof of a theorem is valid, the proof dependency file is updated with information about the premises of the proof and the time at which the proof was performed. To confirm that a proof is up-to-date, the proof manager checks that neither the theorem nor any of the premises has been modified since the time of the proof. It also detects circularities in a proof and.ensures that the proof dependency graph is acyclic.

6 Conclusion

Our experience in using the RTGIL tools has shown that these tools and the graphical representation of the logic are very helpful for specifying and verifying properties of concurrent real-time systems. In addition to the aircraft example, we have used these tools to specify and verify properties of a railroad crossing system, a robot, an alarm system, and a four-phase handshaking protocol.

The RTGIL tools are implemented in Lucid Common Lisp and also in Franz Allegro Common Lisp, and require at least 32 MBytes of main memory and 64 Mbytes of swap space. The graphical editor was implemented using the Garnet graphics toolkit [3], which runs within the X window system. The RTGIL tools and related papers are publicly available, and can be obtained by anonymous ftp from alpha.ece.ucsb.edu in directory /pub/RTGIL.

References

1. R. Alur and D. Dill, "Automata for modelling real-time systems," *Proceedings of 17th International Conference on Automata Languages and Programming*, Warwick University, England (July 1990), LNCS 443, Springer-Verlag, pp. 322-335.
2. D. L. Dill, "Timing assumptions and verification of finite-state concurrent systems," *Proceedings of International Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, France (June 1989), LNCS 407, Springer-Verlag, pp. 196-212.
3. 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.
4. 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 Thirteenth Conference on Foundations of Software Technology and Theoretical Computer Science*, Bombay, India (December 1993), LNCS 761, Springer-Verlag, pp. 173-192.