A Graphical Property Specification Language*

Insup Lee
Real-Time Systems Group
University of Pennsylvania
lee@central.cis.upenn.edu

Oleg Sokolsky
Computer Command and Control Company
sokolsky@cccc.com

July 7, 1997

Abstract

The paper presents a language for specification of high-level properties of real-time systems. The language is based on a temporal logic. Properties expressed as temporal logic formulas are known to be very obscure. In the design of the language, we tried to identify sources of this and make expressions as easy to comprehend as possible. To enhance flexibility of the language, we employ a two-level approach: expert level and user level. This allows to hide obscure formal notation from the user, and at the same time adjust the language to any desired problem domain. User-level expressions have grapical notation, which brings out the structure of expressions in a natural way and leads to easier understanding of formulas.

1 Introduction

The paper presents the language L_R for specification of high-level properties of real-time systems. L_R is based on a propositional temporal logic. Temporal logic has been used to specify behavioral properties of reactive systems for the last 20 years, following the pioneering work by Pnueli [15]. Model checking [6] has emerged in the past decade as an important technique to verify that a system has the property specified by a temporal logic formula.

Many different kinds of temporal logics have been put forward for specification of real-time reactive systems [2, 4, 7, 10, 11, 14, 16]. All of these logics share the same drawback: formulas representing properties of even moderate complexity tend to be hard to understand. Usually, an expert in formal methods is required to create such formulas. Attempts to make temporal logics more intuitive usually include wrapping up obscure formal notation in a user-friendly packaging. These approaches either assume that certain constructs will be intuitive for the wide range of users, or target a specific user group.

We propose a more flexible approach. It involves two levels of development. At the first level, an expert constructs partial formulas that capture the users' intuition for the problem domain. This is done based on the interaction between the expert and the user. This guarantees that the constructs given to the user will indeed be adequate to his problem. At the second level, the users use the patterns provided by the experts at the first level, to express properties of their systems. The patterns can be organized into libraries and reused in later projects. This two-level approach provides for a natural division of responsibilities, when the users, who are not necessarily well versed in formal methods, are able to express properties in terms, familiar to them. On the other hand, involvement of an expert ensures the proper use of the formal machinery.

We use L_R to express properties of specifications expressed in the real-time process algebra ACSR [13]. ACSR employs the notion of discrete time. Operational semantics for ACSR gives rise to a labeled transition

*This work is supported in part by ONR SBIR Phase II grant N00014-95-C-0131 and AFOSR grant F49620-95-1. The first author is also supported by ARO DAAH04-95-1-0092.

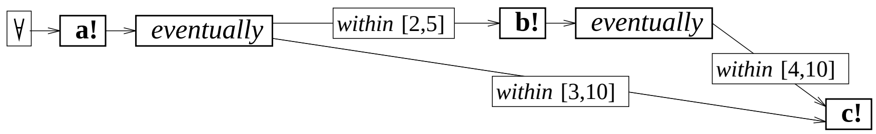


Figure 1: A net representation of the sample property.

system (LTS) with two types of transitions. Transitions of the first kind are labeled with communication events and represent instantaneous interaction between processes in the specification. Transitions of the second kind represent passage of time and are labeled with sets of resources that are used by the system during the time interval. This explains our choice of modalities in the underlying temporal logic (see Section 4).

The rest of the paper is organized as follows. Section 2 explores the source of obscurity in temporal logic specifications, and uses an example to explain out approach to alleviate it. Section 3 outlines the structure of user-level specifications in L_R. After that, we introduce the underlying temporal logic, detailed in Section 4 and explain the translation of user-level constructs into this temporal logic in Section 5.

2 Motivation and Goals

When choosing a logic for expressing behavioral properties, one has to balance expressive power of the logic with efficiency of verification algorithms. Additionally, easiness to express properties and comprehend the intuitive meaning of formulas is a very important factor. This is especially true for real-time logics, where one has to express delays between events and formulas quickly become too complex to understand.

We motivate our design decision in the choice of a property specification language by the following example. Consider the following property: after event a, we will observe event b, and then event c. We require that b happens between 2 and 5 time units after a, and c happens between 4 and 10 units after b, and between 3 and 10 units after a.

In TCTL [1] the property is represented as $\Box a \Rightarrow x.\Diamond(b \Rightarrow y.(x \in [2,5] \wedge \Diamond(c \wedge (x \in [3,10] \wedge y \in [4,10]))))$, which is quite hard to read. This shows that we have to wrap the formal notation of a temporal logic in more intuitive user-level constructs. An additional source of visual complexity of the formula is the use of named clocks. Named clocks are essential to represent this property, however, since the occurrence of event c is related to two points in the past. Temporal logics like RTCTL [9] that rely on the use of bounded temporal operators cannot express such properties.

In L_R, the property is represented very intuitively as shown in Figure 1. The graphically presented structure of a formula allows us to dispense with the clocks at the user level.

In general, a formula is an acyclic graph. Nodes of the graph are operators of the logic - state predicates, propositional connectives, modal and temporal operators. The novel part is in the representation of temporal operators. Each temporal operator – illustrated by operator `eventually` in the example above – is represented as a node with an arbitrary number of outgoing edges, each of which may be labeled with a set of conditions, to which we refer as modifiers, to be applied to the corresponding subformulas. In addition to modifier `within` shown above, modifiers can refer to events and resources expected or prohibited to be encountered in the system's behavior.

Structures like the one presented above are used only as a user-level view of the formulas, striving to achieve better understandability of formulas. Internally, the formulas are translated into the alternation-free fragment of a version of real-time μ-calculus, for which efficient and easy to implement verification algorithms exist [17]. Semantics of L_R expressions is given in terms of this translation.

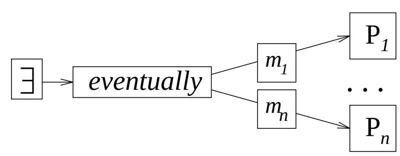
(a) (b)

Figure 2: Legal (a) and illegal (b) graphs.

3 Syntax of L_R

L_R does not have a fixed list of constructs, but consists of a set of patterns that the users can employ in property specifications. The actual constructs used in specifications are provided by formal methods experts on the first level. In this section, we present the framework of the user level language. Properties are represented as directed acyclic graphs. Nodes of the graph represent predicates, logical connectives, modal operators and quantified temporal operators. *Predicates* are non-recursive properties that can be decided locally for each state (e.g. "state is deadlocked" or "state is time-stopping"). *Logical connectives* are conjunction, disjunction and negation.

Modal operators are "next state" properties and represent single occurrences of a communication event from a specified set. P is an arbitrary L_R expression. The property expressed by such expression is true in a given state of an LTS if, after taking the transition labeled one of e_i, P is true in the target state. If there is no transition labeled by an appropriate event possible from the state, the property is false. We differentiate between input and output events in the specification. Following a common notation, we represent input events by adding '?' to the event name, and output events, by adding '!'. If an event name does not have either of the two marks, both input and output events match the modality.



Temporal operators express properties of a path. The two path quantifiers are "along some path" and "along every path", represented as nodes labeled by symbols $\boxed{\exists}$ and $\boxed{\forall}$. A temporal operator is a keyword, like "*eventually*", "*always*" etc. The set of temporal operators is defined at the first level (see Section 5). The out-edges of the temporal operator node are labeled by *path modifiers*, denoted here by m_i. A temporal operator can have multiple outgoing edges, and edges can be labeled by several modifiers each. All modifiers attached to a temporal operator have to be satisfied along the path.

We distinguish several kinds of modifiers:

- interval modifiers, e.g. "*within* $[t_1, t_2]$".
- path restrictions, which specify conditions on transitions of the path, e.g. "requiring resource R" or "avoiding event E".
- path terminators, which specify conditions on states of the path, e.g. "while having access to resource R".

Specific modifiers are also definied at the first level.

To ensure that formulas have well-defined semantics, we place a syntactic restriction on the structure of a graph. Let there be multiple paths between two nodes in a graph. If we view these paths as sequences of nodes, then one of the sequences would contain all others as subsequences. Figure 2, a shows a legal graph, which expresses the property that there is an execution path along which the system receives the signal `set` in the interval between 3 and 8 time units, and will respond with signal `clear` no later than 5 time units thereafter. Additionally, it states that `reset` signal does not appear on this path until `clear` is received.

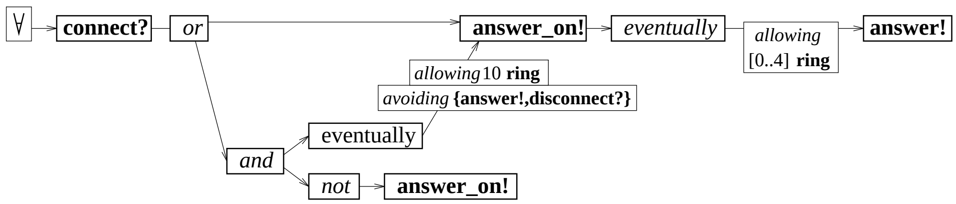


Figure 3: A partial telephone specification.

Figure 2, b gives an example of an invalid expression, because operators *always* and *eventually* lie on parallel paths.

To give a more realistic example, in Figure 3 we show a partial specification of a telephone with a built-in answering machine. If the answering machine is on (represented by the event `answer_on`), the answer comes within 4 rings (represented by the event `ring`). Modifier *allowing* puts a range on the number of occurrences of the specified event along the path. We do not distinguish between a human answering a phone and a recorded response. If the answering machine is not on, the phone continues ringing. After 10 rings, if a human does not pick the receiver, the answering machine is turned on and a recorded message is played.

The syntax for L_R has been motivated by *constraint graphs* introduced in [5]. Constraint graphs present an intuitive way to represent end-to-end timing constraints for a system. They allow the user to describe a black-box view of the system in terms of observable events and allowed delays between them. In this work, we tried to extend constraint graphs to more general properties while keeping the intuitiveness of the resulting specifications.

4 Real-Time μ-calculus

We employ a version of a propositional modal mu-calculus [12], restricted to its alternation-free [8] subset. The set of modal operators of the mu-calculus has been extended to accommodate for the peculiarity of LTSs corresponding to ACSR terms. Note that L_R semantics do not have to be tied to any specific process specification language. Any language whose terms give rise to finite-state LTSs can be used just as well.

The advantage of μ-calculi is the simple "step" semantics of operators, which with the exception of the fixed-point operator is given in terms of a single state, or its immediate successors. The formulas are interpreted with respect to a given labeled transition system of an ACSR term. Each subformula evaluates to a set of states satisfying it.

To provide quantitative analysis of specifications, we employ two kinds of variables in formulas. Clocks are special variables that are spontaneously incremented whenever time passes, and can be reset by the specification. Since we are dealing with a discrete time model, operation of a clock amounts to counting time steps in the LTS since the last time it has been reset. Another type of variables are counters. A counter is labeled by an event set. The value of a counter is incremented whenever an event from the corresponding set occurs. Like clocks, counters can be reset by the specification.

The logic contains two state predicates, **T** for true (satisfied by all states in the LTS) and **F** for false (satisfied by the empty set). Other state predicates are of the form $x \in [a,b]$, or $x - y \in [a,b]$, where x and y are clocks or counters, and a and b are integers ($a \leq b$). Clock values may only be compared with clocks, and counters with counters. A state satisfies such predicate if the value of the specified variable is within the given bounds. We have two kinds of modal operators, corresponding to the two kinds of transitions in the LTS. Operators of the first kind are labeled with a set of event names and serve as existential ($\langle E\rangle$) and universal ($[E]$) quantification over successors of the current state, reachable by an event transition labeled with an event from the specified set. Operators of the seconds kind are labeled by a set of resources and serve as existential ($\langle\langle R\rangle\rangle$) and universal ($[[R]]$) quantifiers over successors of the current state reachable by a time transition that uses all resources in R.

Due to space limitations, we omit the formal definition of the semantics for the logic. It can be found in the full version of the paper.

5 Translation

Each node of an L_R expression is translated into a template. A template is a partially constructed formula of the underlying temporal logic. Logical connectives and modal operators are directly translated into corresponding operators of the logic. Translation of temporal operators is more complex since they can have modifiers labeling its outgoing edges. Therefore, we first have to consider how modifiers contribute to the translation.

Modifiers place restrictions on traces to which the temporal operator is applied. A modifier labeling an edge between two nodes affects the translation of both source and destination node of the edge, as well as all the nodes that are bypassed by the edge.

Each modifier gives rize to up to four components:

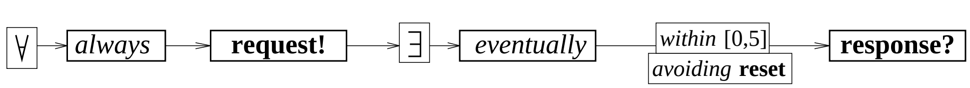
- H is a set of clocks or counter variables that are used to monitor passage of time and occurrence of events.
- m_e and m_r are sets of events or resources, respectively, that are required to be present along the path.
- r is a condition that controls recursion, i.e. stepping along the path. When r is violated, recursion terminates. How such termination affects the property being checked along the path is left to the temporal operator that the modifier is attached to.
- c is the condition to be checked in addition to the target property; it is employed mostly by interval modifiers.

Components c and r are mu-calculus formulas that are plugged into the designated places within the internal representation of temporal operators.

Each temporal operator is translated into a template, which is filled in by the associated quantifiers and modifiers. A temporal operator is always in the scope of a single path quantifier, the closest among its ancestors in the graph. The restriction on the structure of a graph ensures that this is well-defined. A temporal operator may be in the scope of several modifiers. The modifiers whose scope contains a temporal operator node are those that label either the out-edges of the node itself, or the forward edges that bypass the node. A template corresponding to a temporal operator node can be an arbitrary mu-calculus formula, parameterized by its path quantifier and modifier components.

Templates are provided by formal methods experts. The experts, through interaction with the users of a specific application domain, decide which operators are natural for the users to employ and encode them in the underlying logic. The use of templates for temporal operators provides for extensibility of the language. Encapsulating the most commonly used constructs in a readable form, they hide intricacies of mu-calculus from the end user.

To give a complete example of translation, we show, without any explanations because of space limitations, the representation of the property that whenever a request is sent, it is always possible to obtain a response within 5 time units. This property is represented as the following L_R expression:



This expression is translated into the following mu-calculus formula:

$$\nu X_1.(([-]X_1 \wedge [[-]]X_1) \wedge [request!]\mu X.y.((\langle -reset\rangle X \wedge \langle\langle -\rangle\rangle X \wedge \neg deadlocked) \vee (\langle response?\rangle T \wedge y \in [0,5])$$

Note the clock y, which is reset when *request* happens and then, when *response* occurs, its value is checked against the restrictions. Also, occurrences of *reset* are disallowed in the recursive step of the inner fixed point, which corresponds to the operator *eventually*.

6 Conclusions and Future Work

We have presented a language for specification of high-level behavioral properties of real-time systems. The appeal of the language is twofold. First, is set of constructs is extensible and thus can be tailored to match the needs of users in a specific problem domain. Second, the graphical nature of expressions in the language allows us to represent the structure of the properties in a natural way, which makes the specifications more understandable.

To provide semantics to L_R expressions, we had to place severe restrictions on the structure of the expression graphs. We feel that these restrictions can be alleviated in many cases. Research on the ways to make L_R expressions more flexible is an important avenue of future work.

A tool that uses L_R to express and verify properties of ACSR specifications is being implemented as a part of PARAGON toolset [3]. The tool contains a graphical editor for L_R expressions and a multithreaded model checker for the underlying logic.

References

[1] R. Alur. *Techniques for Authomatic Verification of Real-Time Systems.* PhD thesis, Stanford University, 1991.

[2] R. Alur and T. A. Henzinger. Real-time logics: Complexity and expressiveness. *Information and Computation*, 104(1):35–77, 1993.

[3] H. Ben-Abdallah, D. Clarke, I. Lee, and O. Sokolsky. PARAGON: A Paradigm for the Specification, Verification, and Testing of Real-Time Systems. In *IEEE Aerospace Conference*, pages 469–488, Feb 1-8 1997.

[4] S. Berezin and N. Shilov. An approach to effective model-checking of real-time finite-state machines on mu-calculus. In *Proceeedings of LFCS '94*, pages 47–55. LNCS 813, 1994.

[5] D. Clarke. *Testing Real-Time Constraints.* PhD thesis, Department of Computer and Information Science, University of Pennsylvania, November 1996.

[6] E. M. Clarke and E. A. Emerson. *Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic.* LNCS 131, 1981.

[7] E. A. Emerson. Real-time and the mu-calculus. In J. de Bakker *et al.*, editor, *Real-Time: Theory in Practice*, Proceedings of REX Workshop. LNCS 600, 1991.

[8] E. A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Proceedings LICS '86.* IEEE Computer Society Press, 1986.

[9] E. A. Emerson, A. K. Mok, A. P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. In *Proceedings of CAV'89*, 1989.

[10] E. Harel, O. Lichtenstein, and A. Pnueli. Explicit-clock temporal logic. In *Proceedings of LICS '90*, pages 402–413. IEEE Computer Society Press, 1990.

[11] F. Jahanian and A. K. Mok. Safety analysis of timing properties in real-time systems. *IEEE Transactions on Software Engineering*, 12(9):890–904, Sept. 1986.

[12] D. Kozen. Results on the propositional mu-calculus. *Theoretical Comput. Sci.*, 27:333–354, 1983.

[13] I. Lee, P. Bremond-Gregoire, and R. Gerber. A process algebraic approach to the specification and analysis of resource-bound real-time systems. *Proceedings of the IEEE*, 82(1), January 1994.

[14] J. Ostroff. *Temporal Logic of Real-Time Systems.* Research Studies Press, 1990.

[15] A. Pnueli. The temporal logic of programs. In *Proceedings of FOCS '77*, pages 46–57. IEEE Computer Society Press, 1977.

[16] Y. S. Ramakrishna, P. M. Melliar-Smith, L. E. Moser, L. K. Dillon, and G. Kutty. Interval logics and their decision procedures. Part II: A real-time interval logic. *Theoretical Comput. Sci.*, 170(1–2):1–46, 15 Dec. 1996.

[17] O. Sokolsky and S. A. Smolka. Local model checking for real-time systems. In P. Wolper, editor, *Proceedings of CAV '95*, LNCS 939, pages 211–224, 1995.