Specification and Analysis of Real-Time Systems Using Real-Time Maude

Peter Csaba Ölveczky[1,2] and José Meseguer[1]

[1] Department of Computer Science, University of Illinois at Urbana-Champaign
meseguer@cs.uiuc.edu
[2] Department of Informatics, University of Oslo
peterol@ifi.uio.no

Abstract. Real-Time Maude is a language and tool supporting the formal specification and analysis of real-time and hybrid systems. The specification formalism is based on rewriting logic, emphasizes generality and ease of specification, and is particularly suitable to specify object-oriented real-time systems. The tool offers a wide range of analysis techniques, including timed rewriting for simulation purposes, search, and time-bounded linear temporal logic model checking. It has been used to model and analyze sophisticated communication protocols and scheduling algorithms. Real-Time Maude is an extension of Maude and a major redesign of an earlier prototype.

Tools based on timed and linear hybrid automata, such as Uppaal [1], HyTech [2], and Kronos [3], have been successful in modeling and analyzing an impressive collection of real-time systems. While their restrictive specification formalism ensures that interesting properties are decidable, such finite-control automata do not support well the specification of larger systems with different communication models and advanced object-oriented features.

The Real-Time Maude language and tool emphasizes ease and generality of specification, including support for distributed real-time object-based systems. The price to pay for increased expressiveness is that many system properties may no longer be decidable. However, this does not diminish either the need for analyzing such systems, or the possibility of using decision procedures when applicable. Real-Time Maude can be seen as complementing not only automaton-based tools, but also traditional testbeds and simulation tools by providing a wide range of formal analysis techniques and a more abstract specification formalism in which different forms of communication can be easily modeled.

Real-Time Maude specifications are *executable* formal specifications. Our tool offers various simulation, search, and model checking techniques which can uncover subtle mistakes in a specification. Timed *rewriting* can simulate *one* of the many possible concurrent behaviors of the system. Timed *search* and *time-bounded linear temporal logic model checking* can analyze *all* behaviors – relative to a given treatment of dense time as explained below – from a given initial state up to a certain duration. By restricting search and model checking to behaviors

© Springer-Verlag Berlin Heidelberg 2004

up to a certain duration, the set of reachable states is restricted to a finite set, which can be subjected to model checking. For further analysis, the user can write his/her own specific analysis and verification strategies using Real-Time Maude's reflective capabilities.

The time domain may be discrete or dense. Timed automata "discretize" dense time by defining "clock regions" so that all states in the same clock region are bisimilar and satisfy the same properties [4]. The clock region construction is possible due to the restrictions in the timed automaton formalism but in general cannot be employed in the more complex systems expressible in Real-Time Maude. Real-Time Maude deals with dense time by offering a choice of different "time sampling" techniques, so that instead of covering the whole time domain, only *some* moments in time are visited. For example, one strategy offers the choice of visiting at user-specified time intervals; another strategy allows time to advance "as much as possible" before something "interesting" happens. Real-Time Maude's search and model checking capabilities analyze *all* behaviors *up to* the given strategy for advancing time. Search and model checking are "incomplete" for dense time, since there is no guarantee that the sampling strategy covers all interesting behaviors. However, all the large systems we have modeled in Real-Time Maude so far have had a discrete time domain, and in this case search and model checking completely cover all behaviors from the initial state.

Real-Time Maude has been used in some large case studies, including the specification and analysis of a new and sophisticated suite of protocols for reliable and adaptive multicast in active networks, where formal analysis uncovered subtle design errors which could not be found by traditional testing, while independently finding all bugs discovered by testing [5]. Other tool applications include: analyzing a series of new scheduling algorithms for real-time systems and a reliable multicast protocol being developed by the Internet Engineering Task Force, and developing an execution environment for a real-time extension of the Actor model [6].

Real-Time Maude is implemented in Maude [7] as an extension of Full Maude. The current version is a complete redesign of a prototype developed in 2000 [8] and emphasizes new analysis techniques, user-friendliness, and performance. Since most symbolic simulation, search, and model checking commands are implemented by translating them into corresponding Maude commands [9], Real-Time Maude's performance is in essence that of Maude; in particular, the model checking performance is comparable to that of SPIN [10].

The tool – together with a user manual, related papers, and executable example specifications – is available free of charge at `http://maude.cs.uiuc.edu/`.

Specification and Analysis in Real-Time Maude

Real-Time Maude extends the rewriting logic language Maude [7] to specify *real-time rewrite theories* [11]. A Maude module specifies a rewrite theory (Σ, E, R) where (Σ, E) is an *equational theory* specifying the system's state structure and R is a collection of *conditional rewrite rules* specifying the concurrent transitions of the system. A Real-Time Maude specification is a Maude specification together with a specification of a sort `Time` for the time domain, and a set of *tick rules* which model the time elapse in the system and have the form

$\{t\}$ `=>` $\{t'\}$ `in time` u `if` *cond*

where u is a term, possibly containing variables, of sort `Time` denoting the *duration* of the rule, and the terms t and t' are terms of a designated sort `System` denoting the system state. Rewrite rules that are not tick rules are *instantaneous* rules assumed to take zero time. The initial state should always have the form `{`t''`}`, for t'' a term of sort `System`, so that the form of the tick rules ensures that time elapses uniformly in all parts of the system.

The following very simple example models a "clock" which may be running (in which case the system is in state `{clock(`r`)}` for r the time shown by the clock) or which may have stopped (in which case the system is in state `{stopped-clock(`r`)}` for r the clock value when it stopped). When the clock shows `24` it must be reset to `0` immediately:

```
(tmod DENSE-CLOCK is protecting POSRAT-TIME-DOMAIN .
  ops clock stopped-clock : Time -> System [ctor] .
  vars R R' : Time .
  crl [tickWhenRunning] :
      {clock(R)} => {clock(R + R')} in time R'  if  R' <= 24 - R [nonexec] .
  rl [tickWhenStopped] :
     {stopped-clock(R)} => {stopped-clock(R)} in time R' [nonexec] .
  rl [reset] :   clock(24) => clock(0) .
  rl [batteryDies] :   clock(R) => stopped-clock(R) .
endtm)
```

The built-in module `POSRAT-TIME-DOMAIN` defines the time domain to be the nonnegative rational numbers. The two tick rules model the effect of time elapse on a system by increasing the clock value of a running clock according to the time elapsed, and by leaving a stopped clock unchanged. Time may elapse by *any* amount of time less than `24 -` r from a state `{clock(`r`)}`, and by any amount of time from a state `{stopped-clock(`r`)}`. There is no requirement that the specification be "non-Zeno." To execute the specification we should first specify a time sampling regime, for example by giving the command `(set tick def 1 .)` which says that time should advance by increments of `1` in each application of a tick rule. The command `(trew {clock(0)} in time <= 100 .)` then simulates one behavior of the system up to a total duration `100`. The command `(tsearch [1] {clock(0)} =>* {clock(25)} in time <= 100 .)` checks whether the state `{clock(25)}` can be reached from the state `{clock(0)}` in time less than or equal to `100`.

Time-bounded search and model checking are crucial for analyzing systems where the set of reachable states – relative to the chosen strategy for advancing time – is infinite, since the set of states reachable *within the time bound* should be finite. Such a time bound is not needed when the reachable state space, again relative to the chosen "time sampling" strategy, is finite, as in our example. For these cases, Real-Time Maude offers *untime(bounde)d* search and model checking commands, which apply the selected strategy for advancing time but where the search/model checking is not bounded by a time value.

Real-Time Maude's model checker extends Maude's high-performance linear temporal logic explicit-state model checker [10]. Temporal formulas are formed by user-defined atomic propositions and operators such as `/\` (conjunction), `\/` (disjunction), `~` (negation), `[]` ("always"), `<>` ("eventually"), `U` ("until"), etc. Atomic propositions, possibly parameterized, are terms of sort `Prop` and their semantics is defined by stating for which states a property holds. Propositions may be *clocked* in that they also take the elapsed time into account. A module defining the propositions should import the built-in module `TIMED-MODEL-CHECKER` and the module to be analyzed. The following module defines the unclocked propositions `clock-dead` (which holds for all stopped clocks) and `clock-is(`r`)` (which holds if a *running* clock shows r), and the *clocked* proposition `clockEqualsTime` (which holds if the running clock shows the time elapsed in the system):

```
(tmod MODEL-CHECK-DENSE-CLOCK is including TIMED-MODEL-CHECKER .
  protecting DENSE-CLOCK .
  ops clock-dead clockEqualsTime : -> Prop [ctor] .
  op clock-is : Time -> Prop [ctor] .
  vars  R R' : Time .
  eq {stopped-clock(R)}     |=   clock-dead = true .
  eq {clock(R)}             |=   clock-is(R') = (R == R') .
  eq {clock(R)} in time R'  |=   clockEqualsTime = (R == R') .
endtm)
```

The model checking command `(mc {clock(0)} |=u [] ~ clock-is(25).)` checks whether the clock is different from `25` in each computation (relative to the chosen time sampling strategy). The timed model checking command `(mc {clock(0)} |=t clockEqualsTime U (clock-is(24) \/ clock-dead) in time <= 1000 .)` checks whether the clock always shows the correct time, when started from `{clock(0)}`, until it shows `24` or is stopped.

While Real-Time Maude model checking is not complete in general, it *is* a decision procedure for time-bounded temporal properties when the time domain is discrete (which would be case if the imported module `POSRAT-TIME-DOMAIN` were replaced by `NAT-TIME-DOMAIN`) and the instantaneous rules terminate.

References

1. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. Int. Journal on Software Tools for Technology Transfer **1** (1997) 134–152
2. Henzinger, T.A., Ho, P.H., Wong-Toi, H.: HyTech: A model checker for hybrid systems. Software Tools for Technology Transfer **1** (1997) 110–122
3. Yovine, S.: Kronos: A verification tool for real-time systems. Software Tools for Technology Transfer **1** (1997) 123–133
4. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science **126** (1994) 183–235

5. Ölveczky, P.C., Keaton, M., Meseguer, J., Talcott, C., Zabele, S.: Specification and analysis of the AER/NCA active network protocol suite in Real-Time Maude. In: FASE 2001. Volume 2029 of Lecture Notes in Computer Science., Springer (2001) 333–347
6. Ding, H., Zheng, C., Agha, G., Sha, L.: Automated verification of the dependability of object-oriented real-time systems. In: 9th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'03), IEEE Computer Society Press (2003)
7. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Quesada, J.F.: Maude: Specification and programming in rewriting logic. Theoretical Computer Science **285** (2002) 187–243
8. Ölveczky, P.C., Meseguer, J.: Real-Time Maude: A tool for simulating and analyzing real-time and hybrid systems. In: Third International Workshop on Rewriting Logic and its Applications. Volume 36 of Electronic Notes in Theoretical Computer Science., Elsevier (2000) `http://www.elsevier.nl/locate/entcs/volume36.html`.
9. Ölveczky, P.C., Meseguer, J.: Real-Time Maude 2.0. Submitted for publication. `http://heim.ifi.uio.no/~peterol/RealTimeMaude/rtm-papers.html` (2004)
10. Eker, S., Meseguer, J., Sridharanarayanan, A.: The Maude LTL model checker. In: Fourth International Workshop on Rewriting Logic and its Applications. Volume 71 of Electronic Notes in Theoretical Computer Science., Elsevier (2002)
11. Ölveczky, P.C., Meseguer, J.: Specification of real-time and hybrid systems in rewriting logic. Theoretical Computer Science **285** (2002) 359–405