Abstraction and Completeness for Real-Time Maude

Peter Csaba Ölveczky [a,b] and José Meseguer [b]

[a] *Department of Informatics, University of Oslo*

[b] *Department of Computer Science, University of Illinois at Urbana-Champaign*

Abstract

This paper presents criteria that guarantee completeness of Real-Time Maude search and temporal logic model checking analyses, under the maximal time sampling strategy, for a large class of real-time systems. As a special case, we characterize simple conditions for such completeness for object-oriented real-time systems, and show that these conditions can often be easily proved even for large and complex systems, such as advanced wireless sensor network algorithms and active network multicast protocols. Our results provide completeness and decidability of time-bounded search and model checking for a large and useful class of dense-time non-Zeno real-time systems far beyond the class of automaton-based real-time systems for which well known decision procedures exist. For discrete time, our results justify abstractions that can drastically reduce the state space to make search and model checking analyses feasible.

Key words: Rewriting logic, real-time systems, object-oriented specification, formal analysis, abstraction, completeness

1 Introduction

Real-Time Maude [15,12] occupies a useful middle ground between automaton-based real-time formal tools providing decision procedures, such as Uppaal [7,1], Kronos [18], and HyTech [6], and general modeling and simulation tools which can be applied to much broader classes of systems, clearly beyond the pale of the above decision procedures, but that have quite limited analytic features.

In terms of expressiveness and the generality of the systems that can be specified, Real-Time Maude is clearly in the same category as modeling and simulation tools. But in terms of analytic power, it is much closer to the above-mentioned automaton-based tools, although with some limitations. The limitations in question have to do with the fact that, since we are dealing with general classes of infinite-state real-time systems for which no decision procedures are known, some of the formal analyses are *incomplete*.

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

We shall call an analysis method (for example, Real-Time Maude's (timed) breadth-first search, or LTL model checking) *sound* if any counterexample found using such a method is a real counterexample in the system; that is, if the method does not yield *spurious* counterexamples. In this precise sense, all the formal analysis methods supported by Real-Time Maude are indeed sound. We call an analysis method *complete* if the fact that a counterexample is never found using the method actually means that no such counterexamples exist for the analysis in question. For example, (timed) breadth-first search for a violation of an invariant property will be complete if the fact that the search never finds any such violation (assuming an idealized machine) actually means that no such violations exist. Similarly, the LTL model checking of a time-bounded property φ will be complete if the fact that a model checker responds with the answer *true* actually means that φ holds in the system, and therefore that no counterexamples violating φ exist.

For *discrete time* systems, completeness can be purchased, but at a very heavy price. The point is that, if time is discrete, an analysis can exhaustively visit *all* time instants. This makes breadth-fist search for the violation of an invariant complete. For general real-time systems outside the scope of decision procedures, unbounded time leads to infinite state spaces that cannot be model checked with the standard algorithms. However, under very reasonable assumptions satisfied by practically all discrete-time systems of interest, *time-bounded* LTL properties do have *finite* state spaces that can indeed be model checked, yielding a complete decision procedure for such properties. The heavy price of achieving completeness this way has to do with the fact that visiting all discrete times typically leads to a state space explosion that renders many formal analyses unfeasible.

For *dense time* systems, achieving completeness by visiting all times is indeed quite hopeless. The problem, of course, is that if time advances from, say, time r to time $r + r'$ with $r' > 0$ there is an *infinite* set of intermediate times r'' with $r \leq r'' \leq r + r'$ that will *not* be visited if the clock ticks by the positive amount r'. Real-Time Maude deals with this problem by making all analyses relative to a *time sampling strategy*. That is, only those times chosen by the strategy are used to tick the time; and only those behaviors where the states are those corresponding to the chosen times are analyzed. This has several important advantages. First, such time sampling strategies make "tick" rewrite rules that advance time *executable*, whereas if time can tick by any intermediate amount tick rules typically become nondeterministic. Second, under very reasonable assumptions about the time sampling strategy and about the system, it becomes possible for a timed breadth-first search, that only visits states at the chosen times, to examine all such states to see if an invariant is violated. Similarly, even though the state space of even time-bounded LTL properties is now infinite, the subspace obtained by restricting the times to those chosen by the strategy is typically finite, and can indeed be model checked. Of course, since only states with the chosen times are visited, these formal analyses, though sound, are in general incomplete.

The question that this paper raises and provides practical answers to is: under what conditions on the real-time system and on the time sampling strategy can *completeness* be guaranteed? That is, under what conditions does breadth first search become a complete semi-decision procedure for violations of invariants, and does LTL model checking of time-bounded properties (excluding the next operator $\bigcirc$) become a complete decision procedure for such properties even when time is dense?

Our experience in specifying and analyzing a substantial collection of real-time systems has guided our search for criteria guaranteeing completeness. Indeed, as we further explain in this paper, many practical and nontrivial real-time systems, including many of the systems that we had previously analyzed, satisfy the requirements that we present. If they used dense time, this shows in hindsight that many of our analyses were in fact complete. But even for discrete time this provides new completeness guarantees, since to avoid state explosions most of our analyses were made relative to a time sampling strategy also in that case.

The key insight is that many real-time systems are "time-robust" in a sense that we make mathematically precise in Section 3.1. A typical example of a time-robust system is one where each instantaneous transition is triggered by the expiration of a timer or by the arrival of a message with a given transmission delay. The typical time sampling strategy used for such systems is a *maximum time elapse* (*mte*) strategy, that advances time as much as possible to reach the next time at which an instantaneous transition will become enabled. We give simple conditions for time robustness, and also for what we call "tick-stabilizing" state properties that do not change arbitrarily in between *mte* time ticks. Our main result is that for time-robust systems and tick-stabilizing properties the *mte* time sampling strategy is indeed complete.

We prove this property using basic concepts such as *abstraction* and *stuttering bisimulation*. The point is that there are two real-time rewrite theories (and therefore two Kripke structures) involved: the original one, and the one in which time advance is restricted by the *mte* strategy. The behaviors of the restricted theory are of course a *subset* of those of the original theory. We can view the restricted theory as providing an *abstraction* of the original system, similar in nature to those considered in partial order reduction methods [3]. As in partial order reduction, the key point is to show that (after excluding pathological Zeno behaviors) the restricted, more abstract system is *stuttering bisimilar* to the original one. This result (the proof of which is given in [13]) yields as a direct corollary the fact that both systems satisfy the same LTL properties (excluding the next operator $\bigcirc$). We also show how this result can be naturally restricted to *time bounded* LTL properties. This ensures all of our desired completeness guarantees.

Of course, to guarantee completeness one has to check the time robustness of the specification and the tick-stabilizing nature of the relevant state properties. That is, a complete analysis decomposes into two tasks: (1) a standard formal analysis in Real-Time Maude under the *mte* time sampling strategy; and (2) the checking of appropriate proof obligations ensuring time robustness and tick-stabilization. We address the pragmatic question of finding simple and easy-to-check proof obligations to accomplish task (2). Specifically, we show that for a very large class of systems, namely, real-time object-oriented systems made up of objects that can communicate asynchronously by message passing, if one follows the specification methodology advocated in [15], there are indeed quite simple proof obligations that, if met, discharge task (2). We illustrate the ease of checking such proof obligations by means of several nontrivial examples. Finally, since some of the systems that we have analyzed in the past involve the use of probabilistic algorithms, we also include a discussion of how our results can be interpreted for such systems.

The paper is organized as follows: Section 2 gives some background on Real-Time Maude and stuttering simulations. Section 3 defines time-robustness, timed fair behaviors, tick-stabilization and tick-invariance of properties, and proves that unbounded and time-bounded LTL $\setminus\{\bigcirc\}$ model checking using the *mte* time sampling strategy is complete for systems satisfying the above requirements. Section 4 shows how proving those requirements reduces to proving very simple properties for object-based Real-Time Maude specifications, and shows that these properties can easily be proved for our large Real-Time Maude applications.

2 Preliminaries on Real-Time Maude and Stuttering Simulations

2.1 Rewrite Theories

Membership equational logic (**MEL**) [9] is a typed equational logic in which data are first classified by *kinds* and then further classified by *sorts*, with each kind k having an associated set S_k of *sorts*, so that a datum having a kind but not a sort is understood as an *error* or *undefined* element. Given a **MEL** signature Σ, we write $\mathbb{T}_{\Sigma,k}$ and $\mathbb{T}_{\Sigma}(X)_k$ to denote respectively the set of ground Σ-terms of kind k and of Σ-terms of kind k over variables in X, where $X = \{x_1 : k_1, \ldots, x_n : k_n\}$ is a set of kinded variables. *Atomic formulae* have either the form $t = t'$ (Σ-equation) or $t : s$ (Σ-membership) with $t, t' \in \mathbb{T}_{\Sigma}(X)_k$ and $s \in S_k$; and Σ-*sentences* are universally quantified Horn clauses on such atomic formulae. A **MEL** *theory* is then a pair (Σ, E) with E a set of Σ-sentences. Each such theory has an initial algebra $\mathbb{T}_{\Sigma/E}$ whose elements are equivalence classes of ground terms modulo provable equality.

In the general version of rewrite theories over **MEL** theories defined in [2], a *rewrite theory* is a tuple $\mathcal{R} = (\Sigma, E, \varphi, R)$ consisting of: (i) a **MEL** theory (Σ, E); (ii) a function $\varphi \colon \Sigma \to \wp_{\mathrm{f}}(\mathbb{N})$ assigning to each function symbol $f \colon k_1 \cdots k_n \to k$ in Σ a set $\varphi(f) \subseteq \{1, \ldots, n\}$ of *frozen argument positions*; (iii) a set R of (universally quantified) labeled conditional rewrite rules r having the general form

$$(\forall X)\ \ r:\ t \longrightarrow t'\ \ \textbf{if}\ \ \bigwedge_{i\in I} p_i = q_i\ \wedge\ \bigwedge_{j\in J} w_j : s_j\ \wedge\ \bigwedge_{l\in L} t_l \longrightarrow t'_l$$

where, for appropriate kinds k and k_l in K, $t, t' \in \mathbb{T}_\Sigma(X)_k$ and $t_l, t'_l \in \mathbb{T}_\Sigma(X)_{k_l}$ for $l \in L$.

The function φ specifies which arguments of a function symbol f *cannot be rewritten*, which are called *frozen positions*. Given a rewrite theory $\mathcal{R} = (\Sigma, E, \varphi, R)$, a *sequent* of $\mathcal{R}$ is a pair of (universally quantified) terms of the same kind t, t', denoted $(\forall X)\ t \longrightarrow t'$ with $X = \{x_1 : k_1, \ldots, x_n : k_n\}$ a set of kinded variables and $t, t' \in \mathbb{T}_\Sigma(X)_k$ for some k. We say that $\mathcal{R}$ *entails* the sequent $(\forall X)\ t \longrightarrow t'$, and write $\mathcal{R} \vdash (\forall X)\ t \longrightarrow t'$, if the sequent $(\forall X)\ t \longrightarrow t'$ can be obtained by means of the inference rules of reflexivity, transitivity, congruence, and nested replacement given in [2].

2.2 Kripke Structures and Stuttering Simulations

A *transition system* is a pair $\mathcal{A} = (A, \rightarrow_\mathcal{A})$ with A a set (of states) and $\rightarrow_\mathcal{A} \subseteq A \times A$ the *transition relation.* Given a fixed set Π of *atomic propositions*, a *Kripke structure* is a triple $\mathcal{A} = (A, \rightarrow_\mathcal{A}, L_\mathcal{A})$, where $\mathcal{A} = (A, \rightarrow_\mathcal{A})$ is a transition system with $\rightarrow_\mathcal{A}$ a *total* relation, and $L_\mathcal{A} : A \rightarrow \wp(\Pi)$ is a labeling function associating to each state the set of atomic propositions that hold in it. We write $\rightarrow^\bullet$ for the total relation that extends a relation $\rightarrow$ by adding a pair (a, a) for each a such that there is no b with $a \rightarrow b$.

To a rewrite theory $\mathcal{R} = (\Sigma, E, \varphi, R)$ we can associate a Kripke structure $\mathcal{K}(\mathcal{R}, k)_{L_\Pi} = (\mathbb{T}_{\Sigma/E,k}, (\xrightarrow[1]{}_{\mathcal{R},k})^\bullet, L_\Pi)$ in a natural way provided we: (i) specify a kind k in Σ so that the set of *states* is defined as $\mathbb{T}_{\Sigma/E,k}$, and (ii) define a set Π of (possibly parametric) *atomic propositions* on those states; such propositions can be defined equationally in a protecting extension $(\Sigma \cup \Pi, E \cup D) \supseteq (\Sigma, E)$, and give rise to a *labeling function* L_Π on the set of states $\mathbb{T}_{\Sigma/E,k}$ in the obvious way. The *transition relation* of $\mathcal{K}(\mathcal{R}, k)_{L_\Pi}$ is the one-step rewriting relation of $\mathcal{R}$, to which a self-loop is added for each deadlocked state. The semantics of linear-time temporal logic (LTL) formulas is defined for Kripke structures in the well-known way (e.g., [3,5]). In particular, for any LTL formula ψ on the atomic propositions Π and an initial state $[t]$, we have a satisfaction relation $\mathcal{K}(\mathcal{R}, k)_{L_\Pi}, [t] \models \psi$ which can be model checked, provided the number of states reachable from $[t]$ is finite. Maude 2.1 [5] provides an explicit-state LTL model checker precisely for this purpose.

In [8] the notion of *stuttering simulations*, which is used to relate Kripke structures, is introduced. For $\mathcal{A} = (A, \rightarrow_\mathcal{A})$ and $\mathcal{B} = (B, \rightarrow_\mathcal{B})$ transition systems and $H \subseteq A \times B$ a relation, a path ρ in $\mathcal{B}$ *H-matches* a path π in $\mathcal{A}$ if there are strictly increasing functions $\alpha, \beta : \mathbb{N} \rightarrow \mathbb{N}$ with $\alpha(0) = \beta(0) = 0$ such that, for all $i, j, k \in \mathbb{N}$, if $\alpha(i) \leq j < \alpha(i+1)$ and $\beta(i) \leq k < \beta(i+1)$, it holds that $\pi(j)\ H\ \rho(k)$. A *stuttering simulation of transition systems* $H : \mathcal{A} \rightarrow \mathcal{B}$

is a binary relation $H \subseteq A \times B$ such that if $a\,H\,b$, then for each path π in $\mathcal{A}$ there is a path ρ in $\mathcal{B}$ that H-matches π. Given Kripke structures $\mathcal{A} = (A, \rightarrow_{\mathcal{A}}, L_{\mathcal{A}})$ and $\mathcal{B} = (B, \rightarrow_{\mathcal{B}}, L_{\mathcal{B}})$ over a set of atomic propositions Π, a *stuttering* Π*-simulation* $H : \mathcal{A} \rightarrow \mathcal{B}$ is a stuttering simulation of transition systems $H : (A, \rightarrow_{\mathcal{A}}) \rightarrow (B, \rightarrow_{\mathcal{B}})$ such that if $a\,H\,b$ then $L_{\mathcal{B}}(b) \subseteq L_{\mathcal{A}}(a)$. We call H a *stuttering* Π*-bisimulation* if H and H^{-1} are stuttering Π-simulations, and we call H *strict* if $a\,H\,b$ implies $L_{\mathcal{B}}(b) = L_{\mathcal{A}}(a)$. A strict stuttering simulation $H : \mathcal{A} \rightarrow \mathcal{B}$ *reflects* satisfaction of $ACTL^*$ formulas without the next operator $\bigcirc$ as explained in [8], where $ACTL^*$ is the restriction of CTL^* to those formulas whose negation-normal forms do not contain any existential path quantifiers [3]. In particular, $ACTL^*$ contains LTL as a special case.

2.3 Real-Time Maude

The references [15,14] describe Real-Time Maude and its semantics in detail. A *real-time rewrite theory* $\mathcal{R}$ is a tuple $(\Sigma, E, \varphi, R, \phi, \tau)$, where (Σ, E, φ, R) is a (generalized) rewrite theory, such that

- ϕ is an equational theory morphism $\phi : \mathit{TIME} \rightarrow (\Sigma, E)$ from the theory TIME [11] which defines time abstractly as an ordered commutative monoid $(\mathit{Time}, 0, +, <)$ with additional operators such as $\dot{-}$ ("monus") and $\leq$;
- (Σ, E) contains a sort `System` (denoting the state of the system), and a specific sort `GlobalSystem` with no subsorts and supersorts and with an operator `{_}` : `System` $\rightarrow$ `GlobalSystem` so that each ground term t of sort `GlobalSystem` reduces to a term of the form `{`u`}` using the equations E; furthermore, the sort `GlobalSystem` does not appear in the arity of any function symbol in Σ;
- τ is an assignment of a term τ_l of sort $\phi(\mathit{Time})$ to every rewrite rule l : `{`t`}` $\longrightarrow$ `{`t'`}` **if** *cond* involving terms of sort `GlobalSystem`[1]; if $\tau_l \neq \phi(0)$ we call the rule a *tick rule* and write r : `{`t`}` $\xrightarrow{\tau_l}$ `{`t'`}` **if** *cond*. Rewrite rules that are not tick rules are called *instantaneous* rules and are supposed to take zero time.

The state of the system should have the form `{`u`}`, in which case the form of the tick rules ensures that time advances uniformly in all parts of the system. The total time elapse $\tau(\alpha)$ of a rewrite α : `{`t`}` $\longrightarrow$ `{`t'`}` of sort `GlobalSystem` is the sum of the times elapsed in each tick rule application [11]. We write $\mathcal{R} \vdash$ `{`t`}` $\xrightarrow{r}$ `{`t'`}` if there is proof α : `{`t`}` $\longrightarrow$ `{`t'`}` in $\mathcal{R}$ with $\tau(\alpha) = r$. We write *Time*, 0, ..., for $\phi(\mathit{Time})$, $\phi(0)$, etc.

Real-Time Maude extends Full Maude [5] to support the specification of real-time rewrite theories as *timed modules* and *object-oriented timed modules*. Although Real-Time Maude is parametric in the time domain, which may be discrete or dense, it provides a "skeleton" sort `Time` and a supersort `TimeInf`

[1] All rules involving terms of sort `GlobalSystem` are assumed to have different labels.

which adds an infinity element `INF`. Tick rules should (in particular for dense time) have one of the forms

```
crl [l] : {t} => {t'} in time x if cond /\ x <= u /\ cond' [nonexec] .   (†),
crl [l] : {t} => {t'} in time x if cond [nonexec] .                      (∗), or
 rl [l] : {t} => {t'} in time x [nonexec] .                              (§),
```

where x is a variable of sort `Time` which does not occur in `{`t`}` and which is not initialized in the condition. The term u denotes the maximum amount by which time can advance in one tick step. The (possibly empty) conditions *cond* and *cond′* should not further constrain x (except possibly by adding the condition $x \neq 0$). Rules of these forms are in general not executable since it is not clear what value to assign to x in a rewrite step; our tool deals with such rules by offering a choice of different "time sampling" strategies for setting the value of x. For example, the *maximal* time sampling strategy advances time by the maximum possible time elapse u in rules of the form (†) (unless u equals `INF`), and tries to advance time by a user-given time value r in tick rules having other forms. All applications of time-nondeterministic tick rules—be it for rewriting, search, or model checking—are performed using the selected time sampling strategy. This means that some behaviors in the system, namely those obtained by applying the tick rules differently, are not analyzed.

The real-time rewrite theory $\mathcal{R}^{maxDef(r),nz}$ denotes the real-time rewrite theory $\mathcal{R}$ where the tick rules are applied according to the maximal time sampling strategy, and where tick steps which advance time by 0 are not applied. The "untimed" rewrite theory $\mathcal{R}^C$ is obtained from $\mathcal{R}$ by adding a "clock" component to the state, so that the global state is a term of the form `{`t`} in time` r of sort `ClockedSystem`, where r is the duration of the rewrite leading to state `{`t`}`. When the maximal time sampling strategy is chosen, Real-Time Maude executes a command by internally transforming the real-time rewrite theory $\mathcal{R}$ and the command to the theory $(\mathcal{R}^{maxDef(r),nz})^C$, (or to an extension of this, depending on the command to be executed) and executes the corresponding command in Maude [15].

In this paper, we focus on Real-Time Maude's *unbounded* and *time-bounded* search and LTL model checking commands. In unbounded model checking under the maximal time sampling strategy, we check the LTL formula w.r.t. each path in $(\mathcal{R}^{maxDef(r),nz})^C$ starting with the state t_0 `in time` 0. For *time-bounded* model checking with time bound Δ, we only consider the set of paths $Paths(\mathcal{R}^{maxDef(r),nz})^{\leq\Delta}_{t_0}$ of $\mathcal{R}^C$-states "chopped off" at the time limit Δ as explained in [15], where we also define the unbounded and time-bounded satisfaction ($\mathcal{R}, L_\Pi, t_0 \models \Phi$ and $\mathcal{R}, L_\Pi, t_0 \models_{\leq\Delta} \Phi$, respectively) in the expected way.

3 Soundness and Completeness of the Maximal Time Sampling Strategy in Time-Robust Systems

In this section, we define *time-robust* real-time rewrite theories and show that unbounded and time-bounded search and model checking analyses using the *maximal* time sampling strategy are sound and complete with respect to the *timed fair* paths of a time-robust theory, given that the atomic propositions satisfy certain "stability" requirements with respect to tick steps.

A time-robust system is one where:

(i) From any given state time can advance either by (i) *any* amount, by (ii) any amount *up to and including* a specific instant in time, or (iii) not at all.

(ii) Advancing time does not affect the above property, unless time is advanced all the way to the specific bound in time in case 1-(ii) above.

(iii) An instantaneous rewrite rule can only be applied at specific times, namely, when the system has advanced time by the maximal possible amount.

A typical example of such time-robust systems is one where each instantaneous transition is triggered by the expiration of a timer or by the arrival of a message with a given transmission delay. Our experience indicates that many large systems are indeed time-robust.

A time-robust system may have *Zeno* paths, where the sum of the durations of an infinite number of tick steps is bounded. We differentiate between Zeno paths forced on the system by the specification (this could indicate a flaw in the system design) and Zeno paths that are due to bad "choices" in the tick increments. Intuitively, the second type of Zeno behavior does not reflect realistic behaviors in the system and therefore is not simulated by the maximal time sampling strategy. We therefore call *timed fair paths* those paths of the system that do not exhibit this second, unrealistic kind of Zeno behavior.

In Section 3.4 we state that there is a *stuttering bisimulation*, as defined in [8][2], between the set of timed fair paths in a time-robust real-time rewrite theory[3] $\mathcal{R}$ and the theory $\mathcal{R}^{maxDef(r),nz}$, which defines the system where the tick rules are applied according to the maximal time sampling strategy. The full proof of this theorem, together with other proofs, is given in [13]. This main result is proved by proving that for each timed fair path π in (the Kripke structure associated with) $\mathcal{R}$, there is a corresponding path ρ in (the Kripke structure associated with) $\mathcal{R}^{maxDef(r),nz}$ which *matches* π as explained in [8], and vice versa. Such a stuttering bisimulation means that each infinite path can be appropriately simulated to analyze *unbounded* properties. Section 3.5

[2] Although with the slight difference that we work on sets of paths to treat timed fair paths.

[3] In addition, the set of propositions considered must satisfy some properties with respect to tick steps.

gives some requirements which are sufficient to prove that each time-bounded prefix of a timed fair path can be simulated by a time-bounded path obtained by using the maximal time sampling strategy, so that we get completeness also for the analysis of *time-bounded* formulas.

3.1 Time-Robust Real-Time Rewrite Theories

There are two different kinds of tick rule applications that the maximal strategy can treat:

- ticks from states from which time can only advance up to a certain maximal time, and
- ticks from states from which time can advance by *any* amount.

The maximal time sampling strategy handles the first kind of tick steps by advancing time as much as possible, and handles the second kind by advancing time by a user-given time value r.

Definition 3.1 *A one-step rewrite $t \xrightarrow[1]{r} t'$ using a tick rule and having duration r is:*

- *a* maximal tick step*, written $t \xrightarrow[max]{r} t'$, if there is no time value $r' > r$ such that $t \xrightarrow[1]{r'} t''$ for some t'';*
- *an ∞* tick step*, written $t \xrightarrow[\infty]{r} t'$, if for each time value $r' > 0$, there is a tick rewrite step $t \xrightarrow[1]{r'} t''$; and*
- *a* non-maximal tick step *if there is a* maximal *tick step $t \xrightarrow[max]{r'} t''$ for $r' > r$.*

Example 3.2 The following specification models a dense-time *retrograde* clock, where the term `{clock(24)}` should always be explicitly reset to `{clock(0)}`. Just before or after this reset operation, the clock may check whether it has sufficient battery capacity left or whether it must stop immediately by going to a state `{clock-dead(...)}`.

```
(tmod SIMPLIFIED-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 monus R [nonexec] .
  rl [tickWhenStopped] :   {stopped-clock(R)} => {stopped-clock(R)}
                              in time R' [nonexec] .
  rl [reset] :             clock(24) => clock(0) .
  rl [batteryDies1] :      clock(0) => stopped-clock(0) .
  rl [batteryDies2] :      clock(24) => stopped-clock(24) .
endtm)
```

From a state `{clock(5)}` there is an infinity of *non-maximal* tick steps, e.g., to `{clock(10 + 2/7)}`, as well as a *maximal* tick step to `{clock(24)}`. From `{stopped-clock(24)}` there are ∞ tick steps from the term to itself in any amount of time.

The first requirement for a theory $\mathcal{R}$ to be time-robust is that each tick step is either a maximal, a non-maximal, or an ∞ tick step. This excludes specifications with dense time domains where time can advance by any amount strictly smaller than some value Δ from some state, but where time cannot advance by time Δ or more from the same state. Specifications with tick rules of the form †, $*$, and § where the conditions *cond* and *cond′* do not further constrain x will satisfy this requirement.

The next set of assumptions concerns the ability to cover all behaviors without performing non-maximal ticks. We have to make sure that a sequence of non-maximal ticks followed by a maximal tick can be simulated by just one maximal tick step, and that performing a non-maximal tick cannot lead to behaviors (deadlocks, other tick possibilities, taking instantaneous rules, etc.) different from those that can be covered with a maximal tick. Therefore, we add the following requirements: If $t \xrightarrow[max]{r+r'} t''$ is a maximal tick step and $t \xrightarrow[1]{r} t'$ is a non-maximal tick step, then there should be a maximal tick step $t' \xrightarrow[max]{r'} t'''$ for some t'''. Likewise, if $t \xrightarrow[1]{r} t'$ is a non-maximal tick step and $t' \xrightarrow[max]{r'} t''$ is a maximal tick step, then there must be a maximal tick step $t \xrightarrow[max]{r+r'} t''$. Finally, to ensure that no instantaneous rule is ignored by ticking maximally, if $t \xrightarrow[1]{r} t'$ is a non-maximal tick step with $r > 0$, then there is no instantaneous one-step rewrite $t' \xrightarrow[1]{inst} t''$.

We next consider ∞ tick steps. For the system to be time-robust, performing an ∞ tick step should not lead to the possibility of applying an instantaneous rule, and should not preclude the possibility of taking further ∞ steps. Therefore, if $t \xrightarrow[\infty]{r} t'$ is an ∞ tick step with $r > 0$, then there is also an ∞ tick step from t', and there can be no instantaneous step $t' \xrightarrow[1]{inst} t''$. These criteria by themselves only ensure that, once we have performed an ∞ tick step, all the remaining steps will be ∞ tick steps, and that we can have a sequence of ∞ tick steps where time is advanced by r in each step. For our desired bisimulation result, the key idea is that each infinite sequence of ∞ tick steps is simulated by *another* such sequence. However, there need not be any relationship between the states and the durations of these two sequences of ∞ steps. To have a proper stuttering simulation, we must also take the propositions into account.

Real-Time Maude assumes that zero-time tick steps do not change the state of the system; therefore the tool does not apply a tick rule to advance time by 0. Consequently, we require that if $t \xrightarrow[1]{0} t'$ is a tick step that advances time by 0 time units for ground terms t and t', then t and t' must be equivalent in the underlying equational theory of the system.

We summarize the requirements for time-robustness as follows:

Definition 3.3 *A real-time rewrite theory $\mathcal{R}$ is* time-robust *if the following requirements TR1 to TR6 hold for all ground terms t, t', t'' of sort* `GlobalSystem`*,* *and ground terms r, r', r'' of sort* `Time`*:*

TR1. Each one-step rewrite using a tick rule is either a maximal, a non-maximal, or an ∞ tick step.

TR2. $t \xrightarrow[max]{r+r'} t''$ and a non-maximal tick step $t \xrightarrow[1]{r} t'$ imply that there is a maximal tick step $t' \xrightarrow[max]{r'} t'''$ for some t'''.

TR3. For $t \xrightarrow[1]{r} t'$ a non-maximal tick step, $t' \xrightarrow[max]{r'} t''$ implies that there is a maximal step $t \xrightarrow[max]{r+r'} t''$.

TR4. If $t \xrightarrow{r} t'$ is a tick step with $r > 0$, and $t' \xrightarrow[1]{inst} t''$ is an instantaneous one-step rewrite, then $t \xrightarrow{r} t'$ is a maximal *tick step.*

TR5. $t \xrightarrow[\infty]{r} t'$ implies that there are t'', r' such that $t' \xrightarrow[\infty]{r'} t''$.

TR6. $t = t'$ holds in the underlying equational theory for any 0-time tick step $t \xrightarrow[1]{0} t'$.

3.2 Zeno Behaviors and Timed Fairness

For dense time, the form of typical tick rules makes it possible to have "Zeno" behaviors in the system in which an infinite number of tick applications only advances the total time in the system by a finite amount. For analysis purposes it seems important to differentiate between Zeno behaviors caused by "bad choices" about how much to increase time, and Zeno behaviors forced upon the system by the specification (which indicates a design error in the model). In the latter category we also include the case where there is an infinite sequence of applications of *instantaneous* rules. For example, the rewrite sequence

$$t_0 \xrightarrow{1} t_1 \xrightarrow{1/2} t_2 \xrightarrow{1/4} t_3 \xrightarrow{1/8} \cdots$$

is a Zeno behavior caused by bad choices of advancing time in a system which has a tick rule like

```
rl [tick] : {t} => {g(t, x)} in time x .
```

However, a Zeno rewrite sequence

$$\texttt{\{f(1)\}} \xrightarrow{1} \texttt{\{f(2)\}} \xrightarrow{1/2} \texttt{\{f(4)\}} \xrightarrow{1/4} \texttt{\{f(8)\}} \xrightarrow{1/8} \cdots$$

is "forced" by the tick rule

```
crl [tick] : {f(N)} => {f(2 * N)} in time x if x <= 1/N .
```

We will ignore, as *timed unfair*, all paths with an infinite sequence of tick steps where, at each step, time *could* have advanced to time r_0 or beyond, but where the total duration of a path never reaches time r_0.

Another timed unfairness issue deals with the fact that a system could continuously advance time by 0 in a tick rule. If this is the maximum amount by which time can advance from a state, then such bad sequences are not covered by the above non-Zeno requirement. In our clock example, a system could continuously apply the tick rule to the state `{clock(24)}` to reach the same state in the maximal possible time `0`. Given that 0-time ticking should not change the state of the system, we require that a "fair" path does not contain an infinite sequence consisting only of 0-time ticks as long as an instantaneous rule can be applied.

We define *timedFairPaths*$(\mathcal{R})$ to be the set of paths of a theory $\mathcal{R}$ that satisfy the above two requirements, and define unbounded satisfaction of LTL formulas with respect to such timed fair paths as follows:

Definition 3.4 *Given a real-time rewrite theory $\mathcal{R}$, a term t_0 of sort* `GlobalSystem`*, and a ground term r of sort Time, the set Paths$(\mathcal{R})_{t_0}$ is the set of all infinite sequences*

$$\pi = (t_0 \texttt{ in time } r_0 \longrightarrow t_1 \texttt{ in time } r_1 \longrightarrow \cdots \longrightarrow t_i \texttt{ in time } r_i \longrightarrow \cdots)$$

of $\mathcal{R}^C$-states, with $r_0 = 0$, such that either

- *for each i, $t_i \longrightarrow t_{i+1}$ is a one-step rewrite having duration r (which is 0 when an instantaneous rule is applied) with $r_i + r = r_{i+1}$; or*
- *there is a k such that there is no one-step rewrite from t_k in $\mathcal{R}$, and such that $t_k = t_j$ and $r_k = r_j$ for each $j > k$, and for all $i < k$, $t_i \longrightarrow t_{i+1}$ is a one-step rewrite having duration r with $r_i + r = r_{i+1}$.*

The set timedFairPaths$(\mathcal{R})_{t_0}$ is the subset of the paths π in Paths$(\mathcal{R})_{t_0}$ that satisfy the following conditions:

- *for any ground term Δ of sort Time, if there is a k such that for each $j > k$ there is a one-step tick rewrite $t_j \xrightarrow[1]{r} t'$ with $\Delta \leq r_j + r$, then there must be an l with $\Delta \leq r_l$;*
- *for each k, if for each $j > k$ both a maximal tick step with duration 0 and an instantaneous rule can be applied in t_j, then it must be the case that $t_l \xrightarrow[1]{inst} t_{l+1}$ is a one-step rewrite applying an instantaneous rule for some $l > k$.*

We extend the satisfaction relation in [15] to timed fair paths as follows:

Definition 3.5 *Given a real-time rewrite theory $\mathcal{R}$, a protecting extension L_Π of $\mathcal{R}^C$ defining the atomic state and clocked propositions Π, a term t_0 of sort* `GlobalSystem`*, and an LTL formula Φ, we define satisfaction, without time bound, with respect to timed fair paths as follows:*

$$\mathcal{R}, L_\Pi, t_0 \models^{tf} \Phi \quad\iff\quad \pi, L_\Pi^C \models \Phi \quad \textit{for all paths } \pi \in \textit{timedFairPaths}(\mathcal{R})_{t_0}.$$

3.3 Temporal Logic Propositions and Tick Steps

For model checking purposes, atomic propositions should satisfy certain "stability" requirements with respect to tick steps to enable the maximal time sampling strategy to simulate all timed fair paths. For *unbounded* model checking, we may allow the valuation of a set of temporal logic properties to change *once* in a sequence of tick applications. As explained in Section 3.5, for *time-bounded* model checking, properties should not change by tick steps that are not maximal tick steps. The following example shows that it is not necessary to require that a proposition is unchanged by a tick step for unbounded analysis:

Example 3.6 ***(Example 3.2 cont.)*** In our clock example, we could define a proposition `ge20` which holds if the clock shows a value greater or equal than `20`. This proposition is not invariant under ticks. Nevertheless, such a proposition should be allowed under the maximal time sampling strategy with unbounded analysis, since it changes only *once* in any tick sequence from `{clock(0)}` to `{clock(24)}`. The term `{clock(0)}` can therefore "simulate" all the stuttering steps from `{clock(0)}` to the last state in the tick sequence with a clock value less than `20`, and the term `{clock(24)}` could simulate the remaining steps. However, if we add another proposition `ge22`, the valuation of the two propositions could change *twice* in a tick sequence, and neither `{clock(0)}` nor `{clock(24)}` can represent, e.g., the state `{clock(21)}`. The maximal time sampling strategy would never find a behavior containing a state satisfying `ge20 /\ ~ ge22`. Likewise, the proposition `clockIs20` (which holds only for the state `{clock(20)}`) can neither be "simulated" by `{clock(0)}` nor by `{clock(24)}`.

For unbounded model checking we need to assume that the set of propositions under consideration is *tick-stabilizing*, in the sense that if

$$t_1 \xrightarrow[1]{r_1} t_2 \xrightarrow[1]{r_2} \cdots \xrightarrow[1]{r_{n-2}} t_{n-1} \xrightarrow[max]{r_{n-1}} t_n$$

is a sequence of non-maximal tick steps followed by a maximal tick step, then there is an $i < n$ such that $t_1, \ldots, t_i$ can all be simulated by t_1, and $t_{i+1}, \ldots, t_n$ can simulated by t_n:

Definition 3.7 *Let $P \subseteq AP$ be a set of atomic propositions (or a property corresponding to a search pattern). For ground terms t, t', we write $t \simeq_P t'$ (or just $t \simeq t'$ when P is implicit) if t and t' satisfy exactly the same set of propositions from P; that is, $L(t) \cap P = L(t') \cap P$ for the labeling function L. Such a set of propositions P is* tick-stabilizing *if and only if:*

- *For each sequence $t_1 \xrightarrow[1]{r_1} t_2 \xrightarrow[1]{r_2} \cdots \xrightarrow[1]{r_{n-2}} t_{n-1} \xrightarrow[max]{r_{n-1}} t_n$ of non-maximal tick steps followed by a maximal tick step, there is a $k < n$ such that $t_1 \simeq_P t_j$ for each $j \leq k$, and such that $t_l \simeq_P t_n$ for each $l > k$.*
- *$t \xrightarrow[\infty]{r} t'$ and $t' \xrightarrow[\infty]{r'} t''$ implies $t' \simeq_P t''$ when $r > 0$. In addition, $t \xrightarrow[\infty]{r} t'$ and $t \xrightarrow[\infty]{r''} t'''$ implies $t' \simeq_P t'''$ for $r, r'' > 0$.*

For *clocked* propositions[4], the tick steps $t \xrightarrow{r} t'$ should be understood as

[4] Whether or not a state satisfies a *clocked* proposition depends not only on the state, but

their equivalent clocked rewrites t `in time` $r' \longrightarrow t'$ `in time` $r' + r$ for each r', so that by "$t \xrightarrow{r} t'$ implies $t \simeq_{\{p\}} t'$ " with p a clocked proposition we mean that, for each time value r', the proposition p must hold for the clocked state t `in time` r' if and only it holds for the clocked state t' `in time` $r' + r$.

3.4 Completeness of Unbounded Model Checking

In this section we prove that $\mathcal{R}, L_\Pi, t_0 \models^{tf} \Phi$ holds if and only if it is the case that $\mathcal{R}^{maxDef(r),nz}, L_\Pi, t_0 \models \Phi$ holds, for $\mathcal{R}$ a time-robust real-time rewrite theory, Φ an $LTL \setminus \{\bigcirc\}$ formula, and L_Π a labeling function extending $\mathcal{R}^C$ such that the set of atomic propositions occurring in Φ satisfies the tick-stabilization requirement. This equivalence is proved by showing that $\simeq_P$ is a strict stuttering bisimulation between the Kripke structure $\mathcal{K}(\mathcal{R}^C, \texttt{[ClockedSystem]})_{L_\Pi^C}$, restricted to its timed fair paths, and $\mathcal{K}((\mathcal{R}^{maxDef(r),nz})^C, \texttt{[ClockedSystem]})_{L_\Pi^C}$. Furthermore, the bisimulation is *time-preserving* as explained in the proof, which is given in [13].

Lemma 3.8 *Let $\mathcal{R}$ be a time-robust real-time rewrite theory, r a time value in $\mathcal{R}$ greater than 0, L_Π a protecting extension of $\mathcal{R}^C$ which defines the propositions Π, and $P \subseteq \Pi$ a set of tick-stabilizing atomic propositions (some of which could be clocked, and some unclocked). Then,*

$$\simeq_P : \mathcal{K}(\mathcal{R}^C, \texttt{[ClockedSystem]})_{L_\Pi^C} \;\rightarrow\; \mathcal{K}((\mathcal{R}^{maxDef(r),nz})^C, \texttt{[ClockedSystem]})_{L_\Pi^C}$$

is a strict stuttering P-simulation when $\mathcal{K}(\mathcal{R}^C, \texttt{[ClockedSystem]})_{L_\Pi^C}$ is restricted to timed fair paths.

The converse of the above lemma also holds (the proof is given in [13]):

Lemma 3.9 *The relation $\simeq_P$ is a strict stuttering P-simulation*

$$\simeq_P : \mathcal{K}((\mathcal{R}^{maxDef(r),nz})^C, \texttt{[ClockedSystem]})_{L_\Pi^C} \;\rightarrow\; \mathcal{K}(\mathcal{R}^C, \texttt{[ClockedSystem]})_{L_\Pi^C}$$

when $\mathcal{K}(\mathcal{R}^C, \texttt{[ClockedSystem]})_{L_\Pi^C}$ is restricted to timed fair paths.

Theorem 3.10 *Let $\mathcal{R}$ be a time-robust real-time rewrite theory, r a time value in $\mathcal{R}$ greater than 0, L_Π a protecting extension of $\mathcal{R}^C$ which defines the propositions Π, and $P \subseteq \Pi$ a set of tick-stabilizing atomic propositions (some of which could be clocked, and some unclocked). Then $\simeq_P$ is a* strict stuttering P-bisimulation *between $\mathcal{K}(\mathcal{R}^C, \texttt{[ClockedSystem]})_{L_\Pi^C}$,* restricted to its timed fair paths*, and $\mathcal{K}((\mathcal{R}^{maxDef(r),nz})^C, \texttt{[ClockedSystem]})_{L_\Pi^C}$.*

Proof. Follows directly from Lemmas 3.8 and 3.9, since $\simeq_P$ equals $(\simeq_P)^{-1}$ because $\simeq_P$ is symmetric. □

also on the total time it has taken to reach the state.

The following main result, whose proof is also presented in [13], shows that maximal time sampling analysis is sound and complete:

Corollary 3.11 *Let $\mathcal{R}$, t_0, r, L_Π, and P be as in Theorem 3.10, and let Φ be an LTL $\setminus \{\bigcirc\}$ formula whose atomic propositions are contained in P. Then*

$$\mathcal{R}, L_\Pi, t_0 \models^{tf} \Phi \qquad \textit{if and only if} \qquad \mathcal{R}^{maxDef(r),nz}, L_\Pi, t_0 \models \Phi.$$

3.5 Soundness and Completeness of Time-Bounded Model Checking using the Maximal Time Sampling Strategy

Real-Time Maude's *time-bounded* model checking features have proved very useful for analyzing large applications [17,16]. Not only are time-bounded properties interesting *per se*, but model checking analyses terminate for time-bounded properties in non-Zeno specifications when using one of Real-Time Maude's time sampling strategies. Furthermore, even when the reachable state space is finite, using the maximal time sampling strategy can drastically reduce the state space to make time-bounded model checking feasible.

We defined in [15] the semantics of time-bounded properties. Essentially, a time-bounded formula is interpreted over all possible paths "chopped off" at the time limit, and with self-loops added to states from which time *could* advance beyond the time limit in one tick step. In this semantics, the time-bounded property "`<> ge20 in time <= 22`" would *not* hold from `{clock(0)}` in our clock example, since there is, e.g., a tick step from the clocked state `{clock(0)} in time 0` to the state `{clock(24)} in time 24` in *one* step. However, the property *does* hold with time bound 24 according to our semantics.

We denote by $\mathit{timedFairPaths}(\mathcal{R})^{\leq\Delta}_{t_0}$ the subset of $\mathit{Paths}(\mathcal{R})^{\leq\Delta}_{t_0}$ (see [15]) which contains all the timed fair paths starting in state t_0 that are chopped off at time Δ as explained in [15]. The satisfaction relation $\models^{tf}$ over timed fair paths is extended to time-bounded satisfaction in the obvious way:

Definition 3.12 $\mathcal{R}, L_\Pi, t_0 \models^{tf}_{\leq\Delta} \Phi$ *holds if and only if* $\pi, L_\Pi \models \Phi$ *holds for all* $\pi \in \mathit{timedFairPaths}(\mathcal{R})^{\leq\Delta}_{t_0}$.

Since when we have time bounds a sequence of non-maximal tick steps followed by a maximal tick step can be chopped off before the maximal tick is taken, it is no longer sufficient to require tick stabilization of the atomic propositions. Instead, we now must require *tick-invariance*, which means that only a *maximal* tick step may change the valuation of the propositions:

Definition 3.13 *A time-robust specification is* tick-invariant *with respect to a set P of propositions if and only if it is the case that $t \simeq_P t'$ holds for each* non-maximal *or* ∞ *tick step $t \xrightarrow{r} t'$.*

Fact 3.14 *Tick-invariance implies tick-stabilization.*

Our main result, proved in [13], is that the maximal time sampling strategy is sound and complete for time-robust systems when the atomic propositions are tick-invariant:

Theorem 3.15 *Given a time-robust real-time rewrite theory $\mathcal{R}$, a protecting extension L_Π of $\mathcal{R}^C$ defining the atomic state and clocked propositions Π, a tick-invariant subset $P \subseteq \Pi$, an initial state t_0 of sort* `GlobalSystem`*, a* `Time` *value Δ, and an LTL$\backslash\{\bigcirc\}$ formula Φ whose atomic propositions are contained in P. Then,*

$$\mathcal{R}, L_P, t_0 \models^{tf}_{\leq\Delta} \Phi \quad \textit{if and only if} \quad \mathcal{R}^{maxDef(r),nz}, L_P, t_0 \models_{\leq\Delta} \Phi.$$

The completeness results in Corollary 3.11 and Theorem 3.15 carry over directly to unbounded and time-bounded search without lower timer bounds. Therefore, unbounded and time-bounded search using the maximal time sampling strategy become, respectively, a complete semi-decision procedure and a complete decision procedure for the reachability problem for non-Zeno specifications.

In [13] we prove that our clock specification is time-robust and that the unbounded and time-bounded search and model checking analyses in [15] are complete when using maximal time sampling.

4 Completeness for Object-Based Systems

In Real-Time Maude we usually specify “flat” object-based real-time systems by means of functions

```
op δ : Configuration Time -> Configuration [frozen (1)] .
op mte : Configuration -> TimeInf [frozen (1)] .
```

that define, respectively, the effect of time elapse on a configuration, and the *m*aximum *t*ime *e*lapse (mte) possible from a configuration. We let these functions distribute over the objects and messages in a configuration according generic equations of the form:

```
vars NeC NeC' : NEConfiguration .     var R : Time .
eq δ(none, R) = none .
eq δ(NeC NeC', R) = δ(NeC, R) δ(NeC', R) .

eq mte(none) = INF .
eq mte(NeC NeC') = min(mte(NeC), mte(NeC')) .
```

together with domain-specific equations defining the functions δ and *mte* on *individual* objects and messages. There is usually only one tick rule in such systems. That tick rule has the form

```
crl [tick] : {C:Configuration}
           =>
            {δ(C:Configuration, R:Time)} in time R:Time
           if R:Time ≤ mte(C:Configuration) [nonexec] .
```

This specification technique has been used in most of the larger Real-Time Maude applications, such as in the AER/NCA protocol suite[5] [16], the OGDC algorithm [17], the round trip protocols in [15], etc. Furthermore, in these examples, no instantaneous rule is enabled when a tick rule can advance time by a non-zero amount of time. In such systems, it is fairly simple to prove time-robustness:

Theorem 4.1 *Let $\mathcal{R}$ be a flat object-oriented specification with a tick rule as defined above, let the infinity element* `INF` *be the only element in* `TimeInf` *which is not a normal time value, and let the time domain be* linear *(see [11]). Then, $\mathcal{R}$ is time-robust if the following conditions are satisfied for all appropriate ground terms t and r, r':*

OO1. $mte(\delta(t, r)) = mte(t) \mathbin{\dot{-}} r$, *for all* $r \leq mte(t)$.

OO2. $\delta(t, 0) = t$.

OO3. $\delta(\delta(t, r), r') = \delta(t, r + r')$, *for* $r + r' \leq mte(t)$.

OO4. $mte(\sigma(l)) = 0$ *for each ground instance* $\sigma(l)$ *of a left-hand side of an instantaneous rewrite rule.*

Furthermore, it is sufficient to consider OO1, OO2, and OO3 for t consisting of a single object or message.

The proof of Theorem 4.1 is given in [13]. This theorem simplifies the proof obligations for time-robustness for typical object-based systems to very simple requirements, which should be easy to check mechanically using a theorem prover such as Maude's ITP [4]. Furthermore, proving tick-invariance in such systems amounts to proving `{`t`}` $\simeq_P$ `{`$\delta(t, r)$`}` for all t, r with $r < mte(t)$.

As mentioned above, it is our experience that a large class of real-time specifications satisfy the above criteria. One such useful class is the class of systems where the precise transmission time of each message can be computed when the message is sent, and where each instantaneous rule is triggered either by the arrival of a message or by the expiration of a "timer." In what follows we illustrate the general applicability of our results with some practical examples.

4.1 A Small Network Protocol Example

In [15] we presented some versions of a very simple protocol for computing the *round trip time* of message transmission between pairs of nodes in a network. In one of those versions, it takes each message *exactly* time `MIN-TRANS-TIME` to travel from its source to its destination. The system is specified according to the specification techniques defined above (with time specified by the built-in module `NAT-TIME-DOMAIN-WITH-INF`, with `delta` for δ, etc.) and is given in Appendix A. We prove below that the specification satisfies the simplified requirements for time-robustness:

[5] Tick rules only applied to `ObjectConfigurations` in AER/NCA. This is equivalent to the above setting when the *mte* of a message is 0.

OO1: We must prove $\texttt{mte(delta(}C, r\texttt{))} = \texttt{mte(}C\texttt{)}\ \texttt{monus}\ r$, for C being either a (delayed) message or an object. Using the equations in Appendix A we have:

- $\texttt{mte(delta(dly(}m, r'\texttt{), }\ r\texttt{))} = \texttt{mte(dly(}m, r'\ \texttt{monus}\ r\texttt{))} = r'\ \texttt{monus}\ r = \texttt{mte(dly(}m, r'\texttt{))}\ \texttt{monus}\ r$.
- $\texttt{mte(delta(< }O\ \texttt{: Node | clock : }r'\texttt{, timer : }TI\ \texttt{>, }r\texttt{))} =$ $\texttt{mte(< }O\ \texttt{: Node | clock : }r'\ \texttt{+}\ r\texttt{, timer : }TI\ \texttt{monus}\ r\ \texttt{>)} =$ $TI\ \texttt{monus}\ r =$ $\texttt{mte(< }O\ \texttt{: Node | clock : }r'\texttt{, timer : }TI\ \texttt{>)}\ \texttt{monus}\ r$.

OO2 follows trivially from the fact that $r + 0$ equals r for all r, and that TI monus 0 equals TI for all TI of sort TimeInf. *OO3* holds since + is associative and (t monus r) monus r' equals t monus $(r + r')$. Finally, we show that mte of the left-hand side of any instance of an instantaneous rule is 0. For example, for the rule rttResponse, we have

```
mte(rttReq(O, O', R)  < O : Node | >) =
min(mte(rttReq(O, O', R)), mte(< O : Node | >)) =
min(mte(dly(rttReq(O, O', R), 0)), mte(< O : Node | >)) =
min(0, ...) = 0 .
```

The same happens with each rule whose left-hand side contains a message. The only remaining instantaneous rule is tryagain, whose left-hand side has timer value 0, and therefore also has mte 0.

Finally, we prove tick-invariance of the propositions and search patterns in the analysis commands in Appendix A. The search patterns do not mention any attribute which is changed by delta, so they are tick-invariant. The proposition superfluousMsg is defined as follows:

```
(tomod MC-RTT is including TIMED-MODEL-CHECKER .  protecting RTT-I .
  op superfluousMsg : -> Prop [ctor] .
  vars REST : Configuration .    vars O O' : Oid .
  vars R R' R'' : Time .

  ceq {REST < O : Node | rtt : R, clock : R' >  rttReq(O', O, R'')}
          |= superfluousMsg  =  true     if R'' + MAX-DELAY > R' .
  ceq {REST < O : Node | rtt : R, clock : R' >  rttResp(O, O', R'')}
          |= superfluousMsg  =  true     if R'' + MAX-DELAY > R' .
endtom)
```

Satisfaction of this proposition depends on the value of the clock attribute of the Node object and could therefore potentially be vulnerable to change by a tick. However, if superfluousMsg holds in t, then t contains a message and hence has mte 0, and therefore tick-invariance is vacuously true. If t does not contain a rttReq or rttResp message, ticking will not create such a message, so that superfluousMsg holds neither before nor after the tick step. The last option is that t contains a rttReq with dly r. But then $\texttt{mte(}t\texttt{)}$ is less than or equal to r, so superfluousMsg will not hold for $\{\delta(t, r')\}$ for any $r' < r$, which means that the system is tick-invariant with respect to superfluousMsg.

Therefore, all the analyses performed in [15] on this system are sound and complete when using the maximal time sampling strategy.

4.2 The AER/NCA Case Study

The AER/NCA active network protocol suite is the largest Real-Time Maude application analyzed so far [16,10]. Our specification of AER/NCA follows the above guidelines for object-based specification with one insignificant difference: The variable in the tick rule has sort ObjectConfiguration instead of Configuration, which means that the tick rule cannot be applied to states which contain messages at the "outermost level" (i.e., messages that are not inside link objects) in the configuration. This specification is equivalent to one with a tick rule of the form given in page 16 and where the mte of a message is 0.

Since delta is defined to have co-arity ObjectConfiguration, it is sufficient to prove Requirements *OO1*, *OO2*, and *OO3* for ObjectConfigurations. Furthermore, since the initial states only contain objects of the "smallest" subclasses, it is sufficient to prove the requirements for objects of such subclasses. The time-dependent behavior is straightforward: each object may have a clock and a set of timers. The function delta increases the value of the clocks and decreases the values of the timers according to the elapsed time. mte is defined to be the time remaining until the next timer expires. For example, delta and mte are defined as follows for the objects in the *round trip time* component of AER/NCA, where sender objects have a clock and receiver objects have both a clock and a timer:

```
vars R R' : Time .     var TI : TimeInf .     var O : Oid .

eq delta(< O : RTTsenderAlone | clock : R >, R') =
        < O : RTTsenderAlone | clock : R + R' > .
eq delta(< O : RTTreceivableAlone | clock : R, getRTTResendTimer : TI >, R')
     =  < O : RTTreceivableAlone | clock : R + R',
                                   getRTTResendTimer : TI monus R' > .

eq mte(< O : RTTsenderAlone | >) = INF .
eq mte(< O : RTTreceivableAlone | getRTTResendTimer : TI >) = TI .
```

This is the same definition of delta and mte as for the Node class in the simpler RTT example given in Section 4.1, and satisfaction of the Requirements *OO1*, *OO2*, and *OO3* can be proved in exactly the same way. When an object contains a *set* of timers, satisfaction of the requirements can be easily proved using straightforward induction techniques.

Regarding Requirement *OO4*, it is enough to prove mte($\sigma(l_i)$) = 0 for instances of left-hand sides that do not contain messages. Instantaneous rules with messages in their left-hand sides satisfy the requirements for time-robustness, because the application of such a rule cannot follow directly after a tick step, since a tick step takes a configuration without messages into another configuration without messages. It is easy to inspect each of the 60+ instantaneous rules in AER/NCA to see that either the left-hand side contains a message, or that each instance of a left-hand side of such a rule has `mte` 0.

Finally, we must prove tick-invariance of the search patterns and the atomic propositions that occur in the analysis commands in [16]. Remarkably, *none* of the search patterns or atomic propositions depend on the class attributes that are modified by `delta`, which implies that all the *unclocked* search patterns and propositions are tick-invariant, since $t \simeq \texttt{delta}(t, r)$ holds. However, the *clocked* propositions `nomineeIsBefore` and `nomineeIsAfter` are *not* tick-invariant.

To summarize, it is very easy to prove that our specification of the large and sophisticated AER/NCA suite, as given in [16], and all but one of the analysis commands in [16], satisfy the requirements for the maximal time sampling analysis to be sound and complete. AER/NCA is essentially parametric in the time domain. If time were dense, our analyses would have provided complete model checking procedures also for dense time. When the time domain is discrete, all behaviors can also be covered by the strategy that always advances time by one time unit. However, we gain a lot in efficiency by advancing time as much as possible. For example, using the maximal time sampling strategy, it took Real-Time Maude 1.5 seconds to find the bug in *nominee selection* component, while the same search took 160 seconds, i.e., about 100 times longer, when time was always increased by one time unit.

4.2.1 Probabilistic Behavior and Completeness

The AER/NCA suite contains some components with probabilistic behaviors. For example, one use case in the informal specification of the RTT component sets a timer as follows:

```
... Each receiver or repair server starts a  Get-RTT resend timer
with a duration of a random variate, uniformly distributed between
0 and 1.0, times implosionSuppressionInterval.
```

Although Real-Time Maude at present does not provide explicit support for probabilistic systems, it is however possible to *simulate* such systems using a pseudo-random number generator `random`, and setting the timer to the value `random(N) rem 31`, for some ever-changing seed `N`:

```
rl [R2rs] :
   ...  =>
   < O : RTTrepairserverAlone | getRTTResendTimer : random(N) rem 31, ... >  ...
```

For probabilistic systems completeness of our analyses becomes relative to the probabilistic choices. For example, in the specification of AER/NCA we can only analyze those behaviors that can be reached using the particular random number generator and initial seed value. For the purpose of specifying all possible behaviors, we should replace the above rule with a rule of the form

```
crl [R2rs] :
    ...  =>
    < O : RTTrepairserverAlone | getRTTResendTimer : X, ... >  ...
    if X >= 0 /\ X <= 30 .
```

for a new variable `X` of the appropriate sort. In this way absolute completeness could be regained. Alternatively, if the time domain is discrete, we could force time to stop at each moment in time that is within the desired time interval, and have a nondeterministic choice of whether or not to let the timer expire at that moment. Both of these alternatives would lead to time-robust specifications, so that analyzing these versions with the maximal time sampling strategy would really be complete for all possible behaviors of the AER/NCA protocol. Finally, it is worth mentioning that the *rate control* component of AER/NCA does *not* exhibit any probabilistic features, so that the maximal time sampling strategy analyses really cover all possible behaviors of this component.

4.3 The OGDC Wireless Sensor Network Algorithm

We have recently modeled and analyzed the sophisticated OGDC density control algorithm [19] for wireless sensor networks in Real-Time Maude [17]. Our object-based specification of OGDC uses the specification techniques given above. Given the complexity of the specification, it is remarkable how easy it is to prove time-robustness by proving *OO1* to *OO4*. For example, it follows directly that the `mte` of an instance of the left-hand side of an instantaneous rule is `0`, and proving the other requirements amounts to proving properties like $(x$ `monus` $y)$ `monus` $z = x$ `monus` $(y + z)$. Tick-invariance of the propositions and search patterns used in the analyses in [17] follows trivially, since `delta` does not modify the attributes that define these patterns and propositions.

Besides the fact that proving completeness of the analysis using maximal time sampling is almost trivial, we gain much by using maximal time sampling, since time is measured in *milliseconds* in this algorithm, while one round of the algorithm lasts for 1,000 *seconds*. An analysis based on visiting each moment in time would be unfeasible for such systems. Indeed, none of the analysis commands in [17] could terminate after several hours when we used the default time sampling strategy which increases time by one millisecond in each tick step.

OGDC is a probabilistic algorithm, and we have specified it for simulation purposes, so that the completeness of our analysis is again relative to the treatment of probabilistic behavior. Nevertheless, if we modify our specification by modeling probabilistic features by "pure nondeterminism" as indicated in Section 4.2, we would still have a time-robust specification whose analysis using maximal time sampling would be complete for all possible timed fair behaviors of the OGDC algorithm.

5 Conclusions

We have presented general criteria that, under the maximal time sampling strategy, guarantee completeness of Real-Time Maude formal analyses for a real-time system. We have considered large classes of systems, including many object-oriented systems of interest, to which our criteria can be applied; and we have characterized simple conditions under which our criteria can be checked for such systems.

The practical value of our results is that they apply to many systems outside the scope of the known automaton-based decision procedures; and that they greatly increases the level of assurance that can be attached to formal analyses performed in Real-Time Maude for such systems. Indeed, for systems meeting our criteria, our results yield a complete semi-decision procedure for violations of invariants; and a complete decision procedure for satisfaction of bounded LTL properties without the $\bigcirc$ operator.

The following research directions suggested by this work seem worth investigating:

- further generalizing our criteria to encompass an even broader class of systems for which completeness can be guaranteed;
- development of new abstraction techniques for real-time systems that extend or complement those presented here;
- mechanization of the process of checking proof obligations ensuring our correctness criteria; and
- development of additional case studies to experiment with our techniques and to further improve and simplify their foundations and their tool support.

Acknowledgments. We thank the anonymous referees for helpful comments on earlier versions of this paper. Partial support of this research by ONR Grant N00014-02-1-0715 and by NSF Grant CNS 05-24516 is gratefully acknowledged.

References

[1] G. Behrmann, A. David, and K. G. Larsen. A tutorial on UPPAAL. In M. Bernardo and F. Corradini, editors, *Proc. Formal Methods for the Design of Real-Time Systems (SFM-RT 2004)*, volume 3185 of *Lecture Notes in Computer Science*, pages 200–236. Springer, 2004.

[2] R. Bruni and J. Meseguer. Generalized rewrite theories. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *Proc. 30th International Colloquium on Automata, Languages and Programming (ICALP 2003)*, volume 2719 of *Lecture Notes in Computer Science*, pages 252–266. Springer, 2003.

[3] E. Clarke, O. Grumberg, and D. A. Peled. *Model Checking.* MIT Press, 1999.

[4] M. Clavel. The ITP tool home page. http://maude.sip.ucm.es/itp/.

[5] M. Clavel, F. Dúran, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *Maude Manual (Version 2.1.1)*, April 2005. maude.cs.uiuc.edu.

[6] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122, 1997.

[7] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.

[8] N. Martí-Oliet, J. Meseguer, and M. Palomino. Theoroidal maps as algebraic simulations. In J. L. Fiadeiro, P. Mosses, and F. Orejas, editors, *Proc. WADT 2004*, volume 3423 of *Lecture Notes in Computer Science*, pages 126–143. Springer, 2005.

[9] J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, *Proc. WADT'97*, volume 1376 of *Lecture Notes in Computer Science*, pages 18–61. Springer, 1998.

[10] P. C. Ölveczky, M. Keaton, J. Meseguer, C. Talcott, and S. Zabele. Specification and analysis of the AER/NCA active network protocol suite in Real-Time Maude. In H. Hussmann, editor, *Fundamental Approaches to Software Engineering (FASE 2001)*, volume 2029 of *Lecture Notes in Computer Science*, pages 333–347. Springer, 2001.

[11] P. C. Ölveczky and J. Meseguer. Specification of real-time and hybrid systems in rewriting logic. *Theoretical Computer Science*, 285:359–405, 2002.

[12] P. C. Ölveczky and J. Meseguer. Specification and analysis of real-time systems using Real-Time Maude. In T. Margaria and M. Wermelinger, editors, *Fundamental Approaches to Software Engineering (FASE 2004)*, volume 2984 of *Lecture Notes in Computer Science*, pages 354–358. Springer, 2004.

[13] P. C. Ölveczky and J. Meseguer. Abstraction and completeness for Real-Time Maude (extended version). Manuscript. Available at http://www.ifi.uio.no/RealTimeMaude/abstr-paper.pdf, 2005.

[14] P. C. Ölveczky and J. Meseguer. Real-Time Maude 2.1. In N. Martí-Oliet, editor, *Proc. Fifth International Workshop on Rewriting Logic and its Applications (WRLA 2004)*, volume 117 of *Electronic Notes in Theoretical Computer Science*, pages 285–314. Elsevier, 2005.

[15] P. C. Ölveczky and J. Meseguer. Semantics and pragmatics of Real-Time Maude. *Higher-Order and Symbolic Computation*, 2006. To appear.

[16] P. C. Ölveczky, J. Meseguer, and C. L. Talcott. Specification and analysis of the AER/NCA active network protocol suite in Real-Time Maude. Technical Report UIUCDCS-R-2004-2467, Department of Computer Science, University of Illinois at Urbana-Champaign, 2004. Available at http://www.ifi.uio.no/RealTimeMaude.

[17] S. Thorvaldsen and P. C. Ölveczky. Formal modeling and analysis of the OGDC wireless sensor network algorithm in Real-Time Maude. Manuscript. http://www.ifi.uio.no/RealTimeMaude/OGDC, October 2005.

[18] S. Yovine. Kronos: A verification tool for real-time systems. *Software Tools for Technology Transfer*, 1(1–2):123–133, 1997.

[19] H. Zhang and J. C. Hou. Maintaining sensing coverage and connectivity in large sensor networks. *Wireless Ad Hoc and Sensor Networks: An International Journal*, 1, 2005.

A The Real-Time Maude Specification of a Simple Round Trip Time Protocol

This appendix presents the Real-Time Maude specification and the analysis commands of the simple protocol for finding the round trip time between pairs of nodes in a network that was described in [15]. The specification below treats the case when it takes a message exactly time MIN-TRANS-TIME to travel from source to destination.

```
(tomod RTT is protecting NAT-TIME-DOMAIN-WITH-INF .
  op MAX-DELAY : -> Time .       eq MAX-DELAY = 4 .
  op MIN-TRANS-TIME : -> Time .  eq MIN-TRANS-TIME = 1 .

  class Node | clock : Time, rtt : TimeInf,
               nbr : Oid, timer : TimeInf .

  msgs rttReq rttResp : Oid Oid Time -> Msg .
  msg  findRtt : Oid -> Msg .                  --- start a run

  --- Dly message wrappers:
  sort DlyMsg .
  subsorts Msg < DlyMsg < NEConfiguration .
  op dly : Msg Time -> DlyMsg [ctor right id: 0] .

  vars O O' : Oid .   vars R R' : Time .   var TI : TimeInf .

  --- start a session, and set timer:
  rl [startSession] :
     findRtt(O)  < O : Node | clock : R, nbr : O' >  =>
                 < O : Node | timer : MAX-DELAY >
                 dly(rttReq(O', O, R), MIN-TRANS-TIME) .

  --- respond to request:
  rl [rttResponse] :
     rttReq(O, O', R)  < O : Node | >  =>
                       < O : Node | >  dly(rttResp(O', O, R), MIN-TRANS-TIME) .

  --- received resp within time MAX-DELAY;
  ---   record rtt value and turn off timer:
  crl [treatRttResp] :
    rttResp(O, O', R)  < O : Node | clock : R' >  =>
                       < O : Node | rtt : (R' monus R), timer : INF >
    if (R' monus R) < MAX-DELAY .

  --- ignore and discard too old message:
  crl [ignoreOldResp] :
    rttResp(O, O', R)  < O : Node | clock : R' >  =>  < O : Node | >
    if (R' monus R) >= MAX-DELAY .

  --- start new round and reset timer when timer expires:
  rl [tryAgain] :
     < O : Node | timer : 0, clock : R, nbr : O' >  =>
     < O : Node | timer : MAX-DELAY >
     dly(rttReq(O', O, R), MIN-TRANS-TIME) .

  --- tick rule should not advance time beyond expiration of a timer:
  crl [tick] :
      {C:Configuration}  =>  {delta(C:Configuration, R)} in time R
      if R <= mte(C:Configuration) [nonexec] .

  --- the functions mte and delta:
  op delta : Configuration Time -> Configuration [frozen (1)] .
  eq delta(none, R) = none .
  eq delta(NEC:NEConfiguration NEC':NEConfiguration, R) =
       delta(NEC:NEConfiguration, R) delta(NEC':NEConfiguration, R) .
  eq delta(< O : Node | clock : R, timer : TI >, R') =
          < O : Node | clock : R + R', timer : TI monus R' > .
  eq delta(dly(M:Msg, R'), R) = dly(M:Msg, R' monus R) .

  op mte : Configuration -> TimeInf [frozen (1)] .
  eq mte(none) = INF .
  eq mte(NEC:NEConfiguration NEC':NEConfiguration) =
        min(mte(NEC:NEConfiguration), mte(NEC':NEConfiguration)) .
  eq mte(< O : Node | timer : TI >) = TI .
  eq mte(dly(M:Msg, R)) = R .
endtom)


--- Define an initial state with three nodes:

(tomod RTT-I is including RTT .
  ops n1 n2 n3 : -> Oid .
  op initState : -> GlobalSystem .
  eq initState =
       {findRtt(n1) findRtt(n2) findRtt(n3)
        < n1 : Node | clock : 0, timer : INF, nbr : n2, rtt : INF >
        < n2 : Node | clock : 0, timer : INF, nbr : n3, rtt : INF >
        < n3 : Node | clock : 0, timer : INF, nbr : n1, rtt : INF >} .
endtom)

(set tick max def 10 .)     --- use maximal time sampling strategy

(tsearch [1]
      initState  =>* {C:Configuration
                      < O:Oid : Node | rtt : X:Time,
                                       ATTS:AttributeSet >}
         such that X:Time >= 4
            in time <= 100 .)

(tsearch [1]
    initState =>* {C:Configuration
                  < n1 : Node | rtt : 2, ATTS:AttributeSet >
                  < n2 : Node | rtt : 2, ATTS':AttributeSet >}
       in time <= 100 .)

(tomod MC-RTT is including TIMED-MODEL-CHECKER .  protecting RTT-I .
  op superfluousMsg : -> Prop [ctor] .
  vars REST : Configuration .
  vars O O' : Oid .
  vars R R' R'' R''' : Time .
  ceq {REST < O : Node | rtt : R, clock : R' >  dly(rttReq(O', O, R''), R''')}
          |= superfluousMsg  =  true     if R'' + MAX-DELAY > R' .
  ceq {REST < O : Node | rtt : R, clock : R' >  dly(rttResp(O, O', R''), R''')}
          |= superfluousMsg  =  true     if R'' + MAX-DELAY > R' .
endtom)

(mc initState |=t [] ~ superfluousMsg in time <= 100 .)
```