Completeness of Real-Time Maude Analysis (Extended Version) ⋆

Peter Csaba Ölveczky [a], 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 stuttering-bisimilar abstractions that can drastically reduce the state space to make search and model checking analyses feasible.

1 Introduction

Users of formal tools face a choice between expressiveness and generality of the modeling formalism on the one hand, and the availability of decidable and complete analysis methods on the other. For real-time systems, tools based on timed and linear hybrid automata, such as UPPAAL [1] and HyTech [2], have been successful in modeling and analyzing an impressive collection of systems. However, 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 data types, communication models, and advanced object-oriented features.

⋆ This report, *except for the two appendices*, has been submitted for publication.

Real-Time Maude [3,4] is a high-performance tool that extends the rewriting logic-based Maude system [5,6] to support the formal specification and analysis of real-time systems. Real-Time Maude emphasizes expressiveness and ease of specification over algorithmic decidability of key properties. In Real-Time Maude, the data types of a system are defined by *equational specifications*, instantaneous transitions are defined by *rewrite rules*, and time elapse is defined by *"tick" rewrite rules*. Real-Time Maude supports the specification of distributed *object-oriented* real-time systems in a natural way.

Real-Time Maude specifications are *executable*. Our tool offers a wide spectrum of efficient analysis methods. Timed *rewriting* can simulate *one* of the many possible fair concurrent behaviors of the system. Timed breadth-first *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 up to a certain duration, the set of reachable states is restricted to a finite set (unless the system exhibits pathological Zeno behaviors) that can be subjected to model checking.

The expressiveness and generality of Real-Time Maude's formalism and analysis methods have allowed us to model and analyze state-of-the-art systems in very different application areas, including:

- The CASH *scheduling algorithm* [7]. This algorithm has advanced capacity sharing features for reusing unused execution times.
- The OGDC algorithm for density control in wireless sensor networks [8,9]. This algorithm poses many challenges to its modeling and analysis, including modeling novel communication features (area broadcast with delays) and spatial features such as locations, distances, and coverage areas. To the best of our knowledge, our work on OGDC represents the first attempt at using a formal tool on advanced wireless sensor network algorithms.
- The AER/NCA protocol suite for reliable multicast in active networks [10].

These large and sophisticated systems, with their need to model different data types, unbounded data structures, advanced forms of communication, etc., are clearly beyond the pale of timed automaton-based tools.

Given this expressiveness, formal analysis in Real-Time Maude is generally *incomplete*. We call an analysis method (for example, Real-Time Maude's LTL model checking) *sound* if any counterexample found using such a method is a real counterexample in the system. 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, 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.

The question this paper addresses is: Under what conditions are analyses in Real-Time Maude complete? 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?

For *discrete time* systems, completeness can be purchased, at a very heavy price, by exhaustively visiting *all* time instants; however, this 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 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 $r < r'' < r + r'$ that will *not* be visited. 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. 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.

In this paper, we study analysis using the *maximal time sampling strategy*, that advances time as much as possible to reach in one tick step the next time at which an instantaneous transition will become enabled. In Section 4 we give very general criteria (namely, that the system is "time robust," and that state properties do not change arbitrarily in-between maximal ticks) that ensure that maximal time sampling analyses are complete. We prove this property using the concept of *stuttering bisimulation* [11,12]. The point is that there are two real-time rewrite theories involved: the original one, and the one in which time advance is restricted by the maximal time sampling strategy. The behaviors of the restricted theory are of course a *subset* of those of the original theory. The key point is to show that (after excluding pathological Zeno behaviors) the restricted system is *stuttering bisimilar* to the original one. We prove this result, which 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 restricted to *time bounded* LTL properties.

We then need to ask: how useful are our criteria for completeness?; do inter-

esting classes of systems satisfy them?; and how easy it is to prove that the criteria are satisfied? In [4] we suggest some techniques for specifying real-time systems in an object-oriented way. In Section 5 we show how the general requirements for completeness of maximal time sampling analysis specialize to simple and easily checkable requirements for such object-oriented specifications. We then explain that both the AER/NCA and the OGDC specifications, as well as systems where each instantaneous transition is triggered by the expiration of a timer or by the arrival of a message with a given transmission delay, are time-robust, and that the state properties occurring in the analysis commands in [9,10] satisfy the desired tick-invariance requirement. This shows in hindsight that our analyses were in fact complete. Moreover, we show that even for large and complex systems such as AER/NCA and OGDC, it is fairly easy to prove that the requirements ensuring completeness are satisfied. 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 rewriting logic and stuttering simulations. Section 3 introduces Real-Time Maude. Section 4 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 maximal time sampling strategy is complete for systems satisfying the above requirements. Section 5 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. This paper is an extended and revised version of [13].

2 Preliminaries on Rewriting Logic and Stuttering Simulations

Since Real-Time Maude extends the high-performance Maude tool [6,5] and its underlying rewriting logic formalism, we present in Section 2.1 some background on rewriting logic. Section 2.2 gives some background on Kripke structures and on stuttering simulations [12,14].

2.1 Rewrite Theories

Membership equational logic (**MEL**) [15] 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 formulas* 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 formulas. 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 [16], a *rewrite theory* is a tuple $\mathcal{R} = (\Sigma, E, \varphi, R)$ consisting of: (i) a **MEL** theory (Σ, E); (ii) a function $\varphi \colon \Sigma \to \mathcal{P}_{\mathrm{f}}(\mathbb{N})$ 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

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

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

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 rewriting logic inference rules of reflexivity, transitivity, congruence, and nested replacement given in [16].

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 $L_{\mathcal{A}} : A \to \wp(\Pi)$ 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 $\mathcal{K}(\mathcal{R}, k)_{L_\Pi} = (\mathbb{T}_{\Sigma/E,k}, (\underset{1}{\longrightarrow}_{\mathcal{R},k})^{\bullet}, L_\Pi)$ 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 (Σ, 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., [17,6]). 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 [6] provides an explicit-state LTL model checker for this purpose.

In [12,11] 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 [12], where $ACTL^*$ is the restriction of CTL^* to those formulas whose negation-normal forms do not contain any existential path quantifiers [17]. In particular, $ACTL^*$ contains LTL as a special case. Furthermore, by generalizing the results in [12], it follows from results in [11,18] that stuttering *bisimulations* preserve satisfaction of $CTL^* \setminus \{\bigcirc\}$ formulas.

3 Real-Time Maude

Real-Time Maude [7,4] extends Maude [6,5] to support the formal specification and analysis of real-time systems defined as *real-time rewrite theories* [19]. The references [4,20] describe Real-Time Maude and its semantics in detail. A real-time rewrite theory is a rewrite theory where some rules, called *tick rules*, model time elapse in a system, while "ordinary" rewrite rules model instantaneous change:

Definition 1 *A real-time rewrite theory $\mathcal{R}$ is a tuple $(\Sigma, E, \varphi, R, \phi, \tau)$, where (Σ, E, φ, R) is a rewrite theory, such that*

- *ϕ is an equational theory morphism $\phi : TIME \rightarrow (\Sigma, E)$ from the theory TIME [19] which defines time abstractly as an ordered commutative monoid*

$(Time, 0, +, <)$ *with additional operators such as* $\dot{-}$ *("monus," defined by* $x \dot{-} y = \max(x - y, 0)$*) 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* $\phi(Time)$ *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 [19]. We write $\mathcal{R} \vdash$ `{`t`}` $\xrightarrow{r}$ `{`t'`}` (or just `{`t`}` $\xrightarrow{r}$ `{`t'`}`) if there is proof α : `{`t`}` $\longrightarrow$ `{`t'`}` in $\mathcal{R}$ with $\tau(\alpha) = r$. We write $Time$, 0, ..., for $\phi(Time)$, $\phi(0)$, etc. As proved in [4], and further discussed in Section 3.1, any real-time rewrite theory can be *desugared* into a semantically equivalent ordinary rewritwe theory.

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` which adds an infinity element `INF`. Tick rules should, in particular for dense time, have one of the forms (given in the intuitive Real-Time Maude syntax):

```
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. Rules of these forms are in general not executable (`[nonexec]`) 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 (see [4]). This paper focuses on the *maximal time sampling strategy* which advances time by the maximum possible time elapse u in rules of the form (†) (unless u equals `INF`, in which case time is advanced by a user-given time value r), and tries to advance time by the user-given time value r in tick rules having other forms.

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

Example 2 *The following Real-Time Maude module specifies a dense-time* retrograde *clock, where the term* `{clock(24)}` *should always be explicitly reset to* `{clock(0)}`*. Just before this reset operation, the clock may check whether it has sufficient power left or otherwise must stop immediately by going to a state* `{stopped-clock(24)}`*. The imported predefined module* `POSRAT-TIME-DOMAIN` *defines the time domain to be the nonnegative rational numbers.*

```
(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 [batteryDies] :        clock(24) => stopped-clock(24) .
endtm)
```

The two tick rules are nonexecutable, since time can increase by any *value (up to total clock time* `24` *for rule* `tickWhenRunning`*). Furthermore, the specification admits "Zeno" rewrite paths such as* `{clock(0)}` $\longrightarrow$ `{clock(1/2)}` $\longrightarrow$ `{clock(3/4)}` $\longrightarrow$ `{clock(7/8)}` $\longrightarrow \cdots$*. For execution purposes, we can tell the tool to use the maximal time sampling strategy with default time increment* `10`*. Then, applying the rule* `tickWhenRunning` *on a term* `{clock(`r`)}` *advances time by* `24 monus` r*, resulting in the state* `{clock(24)}`*, and applying the rule* `tickWhenStopped` *on a term* `{stopped-clock(`r`)}` *will advance time by 10 time units although the state will remain* `{stopped-clock(`r`)}`*.*

3.1 Formal Analysis in Real-Time Maude

Real-Time Maude modules are executable under reasonable assumptions after a time sampling strategy has been selected. Our tool extends Maude's analysis commands to provide a range of formal analysis capabilities, including timed rewriting for simulation, time-bounded and unbounded search for reachability analysis, time-bounded and unbounded LTL model checking, and some time-specific analysis commands [4,21]. This paper focuses on LTL model checking, and, for the purposes of this paper, we regard search as a special case.

Real-Time Maude's search command uses explicit-state breadth-first search to analyze *all* behaviors *relative to the selected time sampling strategy*, by investigating whether a term matching a given *pattern* can be reached from the given initial state. The search can be either *time-bounded*, in which case we are only interested in states that can be reached from the initial state within a user-given maximum duration Δ, or can be *unbounded* in the sense that there is no restriction on the duration of the rewrite sequences from the initial state.

We can also analyze all *behaviors* of a system from a given initial state, again relative to the chosen time sampling strategy, using Real-Time Maude's *propositional linear temporal logic model checker*. Temporal formulas are formed exactly as in Maude, that is, as terms constructed by user-defined atomic propositions and operators such as `/\` (conjunction), `\/` (disjunction), `->` (implication), `~` (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. That is, whether a clocked proposition holds for a certain state depends not only on the state, but also on the total duration of the rewrite sequence leading up to the state. As for search, we have both *time-bounded* model checking, where we consider rewrite sequences up to a given duration, and *unbounded* model checking, where there is no such restricion on the behaviors. We refer to [20,4] for a precise definition of the semantics of timed search and model checking.

Both search and model checking are guaranteed to terminate if only a finite set of states can be reached from the initial state. This implies that *time-bounded* search and model checking are guaranteed to terminate if the system, with the chosen time sampling strategy, does not exhibit "Zeno" behaviors, since then only a finite number of states can be reached in a finite time interval.

Since all applications of nondeterministic tick rules are performed using the selected time sampling strategy, those behaviors obtained by applying the tick rules differently are not analyzed. Therefore, the model checker in general cannot *prove* a formula correct in the presence of such tick rules. However, if the tool finds a counterexample, it is a valid counterexample which proves that the formula does not hold. This paper investigates under what conditions the converse also holds: if the analysis does not find a counterexample (or a desired state) using the maximal time sampling strategy, then the formula does indeed hold for all possible behaviors (or a desired state cannot be reached).

Example 3 *Using the maximal time sampling strategy in our clock example, unbounded and time-bounded search (with duration greater than or equal to 24) will find that the state* `{clock(24)}` *is reachable from* `{clock(0)}`*. However, the state* `{clock(3)}` *will not be found by such search.*

3.2 Some Theory Transformations

Real-Time Maude executes a command by performing a series of theory transformations on the given timed module, and then using the underlying Maude facilities to efficiently execute the command [4].

Given a real-time rewrite theory $\mathcal{R}$ and a time value r in $\mathcal{R}$, there is a theory

transformation taking $\mathcal{R}$ to the real-time rewrite theory $\mathcal{R}^{maxDef(r),nz}$, 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. For example, the tick rules in the theory SIMPLIFIED-DENSE-CLOCK$^{maxDef(10),nz}$ are

```
crl [tickWhenRunning] :
    {clock(R)} => {clock(R + R')} in time R' if R' := 24 monus R /\ R' =/= 0 .
crl [tickWhenStopped] :
    {stopped-clock(R)} => {stopped-clock(R)} in time R' if R' := 10 .
```

Given a real-time rewrite theory $\mathcal{R}$, there is a semantics-preserving theory transformation $(\_)^C$ taking $\mathcal{R}$ into an ordinary rewrite theory $\mathcal{R}^C$ by adding a "clock" component to the state. The global state is a term of the form `{`t`}` `in time` d of sort `ClockedSystem`, where d is the duration of the rewrite leading to state `{`t`}`. When the maximal time sampling strategy (with "default" time increment r) 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 [4].

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 [4], where we also define 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.

4 Completeness of Maximal Time Sampling Analyses

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:

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

(3) An instantaneous rewrite rule cannot be applied after a tick step, unless that tick step 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 4.4 we prove that there is a *stuttering bisimulation*, as defined in [12] [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. This is done by proving that for each timed fair path π in (the Kripke structure associated to) $\mathcal{R}$, there is a corresponding path ρ in (the Kripke structure associated to) $\mathcal{R}^{maxDef(r),nz}$ which *matches* π as explained in [12], and vice versa. Such a stuttering bisimulation means that each infinite path can be appropriately simulated to analyze *unbounded* properties. Section 4.5 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.

4.1 Time-Robust Real-Time Rewrite Theories

The maximal strategy can treat two different kinds of tick rule applications:

- 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.

[2] 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.

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 "default" time value r.

Definition 4 *A rewrite step $t \xrightarrow[1]{r} t'$ 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 5 *(Example 2 cont.) 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 $\dagger$, $*$, and $\S$ where the conditions *cond* and *cond*$'$ do not further constrain x 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 at 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.

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 provably equal in the underlying equational theory of the system.

We summarize the requirements for time-robustness as follows:

Definition 6 *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 $t \xrightarrow[1]{0} t'$.

4.2 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 7 *Given a real-time rewrite theory $\mathcal{R}$ and a term t_0 of sort* `GlobalSystem`*, 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 rewrite applying an instantaneous rule for some $l > k$.*

We extend the satisfaction relation in [4] to timed fair paths as follows:

Definition 8 *Given a real-time rewrite theory $\mathcal{R}$, a labeling function L_Π 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:*[4]

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

4.3 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 4.5, for *time-bounded* model checking, properties should not change by tick steps that are not maximal. The following example shows that it is not necessary to require that a proposition is unchanged by a tick step for unbounded analysis:

Example 9 *(Example 2 cont.) In our clock example, we could define a proposition* `gt20` *which holds if the clock shows a value greater 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 a proposition* `gt22`*, the valuation of the two propositions could change* twice *in a tick sequence, and the maximal time sampling strategy would never find a behavior containing a state satisfying* `gt20 /\ ~ gt22`*.*

For unbounded model checking we need to assume that the set of propositions 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

[4] L_Π^C extends the labeling function L_Π to clocked states so that any *state* proposition that holds in state t also holds in t `in time` r for all r.

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 be simulated by t_n:

Definition 10 *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, the tick steps $t \xrightarrow{r} t'$ should be understood as 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` $r' + r$.

4.4 Completeness of Unbounded Model Checking

In this section we prove that $\mathcal{R}, L_\Pi, t_0 \models^{tf} \Phi$ if and only if $\mathcal{R}^{maxDef(r),nz}, L_\Pi, t_0 \models \Phi$ 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 $\mathcal{K}(\mathcal{R}^C, \texttt{[ClockedSystem]})_{L^C_\Pi}$, restricted to its timed fair paths, and $\mathcal{K}((\mathcal{R}^{maxDef(r),nz})^C, \texttt{[ClockedSystem]})_{L^C_\Pi}$. Furthermore, the bisimulation is *time-preserving* as explained in the proof.

Proposition 11 *Let $\mathcal{R}$ be a time-robust real-time rewrite theory, r a time value in $\mathcal{R}$ greater than 0, L_Π a labeling function 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, it is the case that $\simeq_P$: $\mathcal{K}(\mathcal{R}^C, \texttt{[ClockedSystem]})_{L^C_\Pi} \rightarrow \mathcal{K}((\mathcal{R}^{maxDef(r),nz})^C, \texttt{[ClockedSystem]})_{L^C_\Pi}$ is a strict stuttering P-simulation when $\mathcal{K}(\mathcal{R}^C, \texttt{[ClockedSystem]})_{L^C_\Pi}$ is restricted to timed fair paths.*

PROOF. For any timed fair path π in $\mathcal{K}(\mathcal{R}^C, \texttt{[ClockedSystem]})_{L^C_\Pi}$ we construct a corresponding path ρ in $\mathcal{K}((\mathcal{R}^{maxDef(r),nz})^C, \texttt{[ClockedSystem]})_{L^C_\Pi}$ that

$\simeq_P$-matches π as follows:[5]

- Any finite sequence of non-maximal tick steps followed by a maximal tick step, where the duration of the sequence is greater than 0, is replaced by the corresponding maximal tick. That is, the sequence $t_i \xrightarrow[1]{r_1} t_{i+1} \xrightarrow[1]{r_2} \cdots \xrightarrow[max]{r_n} t_{i+n}$ is simulated by the maximal tick $t_i \xrightarrow[max]{\sum_{j=1}^{n} r_j} t_{i+n}$.
- 0-time ticks, which should be identity rewrites, are removed.
- Any (infinite) sequence of ∞-ticks is replaced with an "arbitrary" infinite sequence of ∞-steps from the same "starting" term.

For the full proof, we define the path ρ and the "stuttering" function α inductively as follows (the function β is just the identity function). We need an additional function γ which gives the index of the "current" state in π. For example, if $\pi = t_0 \xrightarrow[1]{r_1} t_1 \xrightarrow[1]{r_2} \cdots \xrightarrow[1]{r_i} t_i \xrightarrow[1]{r_{i+1}} t_{i+1} \xrightarrow[1]{r_{i+2}} \cdots t_{k-1} \xrightarrow[max]{r_k} t_k \cdots$ starts with a sequence of non-maximal tick steps followed by a maximal tick step $t_{k-1} \xrightarrow[max]{r_k} t_k$, then this is simulated, as mentioned, by $t_0 \xrightarrow[max]{\sum r_j} t_k$ in ρ. Assume now that for each $j \leq i$ it is the case that $t_0 \simeq_P t_j$, and that for each $i < l \leq k$ it is the case that $t_l \simeq_P t_k$. Then, the elements $t_0, \ldots, t_i$ should be simulated by t_0, and $t_{i+1}, \ldots, t_k$ should be simulated by t_k, so that $\alpha(0) = 0$ and $\alpha(1)$ is *not* k, but $i + 1$. The function γ is used to relate the corresponding states in π and ρ; in this case $\gamma(1)$ would be k.

The stronger property we prove by induction (on i and $\gamma(i)$) is that either ρ and α are fully defined, or that they, together with γ, are defined "up to" i such that $\gamma(i) \geq \alpha(i)$, $\rho(i) = \pi(\gamma(i))$, and that ρ and α provide a $\simeq_P$-mapping "up to" i. In addition, all $\pi(j)$ for $\alpha(i) \leq j \leq \gamma(i)$ are $\simeq_P$-equal to $\rho(i)$. In particular, the duration up to $\pi(\gamma(i))$ equals the duration of ρ up to $\rho(i)$.

Base case: We define $\alpha(0) = \gamma(0) = 0$ and define $\rho(0)$ to be $\pi(0)$. It is obvious that the property we want to prove holds so far.

Induction step: We assume our property up to i. That is, $\pi(\gamma(i))$ equals $\rho(i)$. The next step $\pi(\gamma(i)) \longrightarrow \pi(\gamma(i) + 1)$ in π could be either of the following:

(1) Application of an instantaneous rule.
(2) Deadlock.
(3) Maximal tick step with duration > 0.
(4) Maximal tick step with duration 0.
(5) An ∞ tick step.
(6) A non-maximal tick step.

[5] Each term in the paths π and ρ is a clocked term of the form t `in time` r. For readability purposes, we write $t \xrightarrow{r'} t'$ for t `in time` $r \longrightarrow t'$ `in time` $r + r'$ for r some appropriate time value.

1. Instantaneous rule application: The same rule can be applied in $\mathcal{R}^{maxDef(r),nz}$ since the the instantaneous rules in $\mathcal{R}$ are not modified in $\mathcal{R}^{maxDef(r),nz}$. Therefore, $\rho(i+1)$ is defined to be $\pi(\gamma(i)+1)$, both $\alpha(i+1)$ and $\gamma(i+1)$ are defined to be $\gamma(i)+1$. The resulting ρ and α still provide a stuttering simulation up to $i+1$, since each $\pi(j)$ between $\pi(\alpha(i))$ and $\pi(\gamma(i))$ are $\simeq_P$-equal to $\rho(i)$.

2. Deadlock: Same as above. If there is a deadlock from $\pi(\gamma(i))$ in $\mathcal{R}$, then there is also a deadlock in the restricted version $\mathcal{R}^{maxDef(r),nz}$.

3. Maximal step with duration > 0: Same as above. This step can also be performed in $\mathcal{R}^{maxDef(r),nz}$.

4. Maximal tick step with duration 0: According to *TR6*, this is the identity rewrite, which is not performed in $\mathcal{R}^{maxDef(r),nz}$. In this case, $\gamma(i)$ is increased by 1, to mark that we have moved forward on the π side. However, we must make sure that $\rho(i+1)$ will eventually be defined. There are two possibilities in state $\pi(\gamma(i))$: either some instantaneous rule is enabled in $\pi(\gamma(i))$, or no instantaneous rule is enabled in $\pi(\gamma(i))$. If an instantaneous rule is enabled in $\pi(\gamma(i))$, then, since a 0-tick does not change the state, it will continue to be enabled as long as a maximal 0-time tick is performed. Since our definition of timed fair paths explicitly forbids an infinite sequence of 0-time maximal ticks when an instantaneous rule is enabled, this means that, eventually, some instantaneous rule must be applied, and $\rho(i+1)$ is defined according to Case 1 above. If no instantaneous rewrite rule can be applied, then, again due to the identity of 0-time ticks, the system $\mathcal{R}$ will forever only be able to perform 0-time ticks. Of course, in this case no rule can be applied in $\mathcal{R}^{maxDef(r),nz}$. In the Kripke structure $\mathcal{K}((\mathcal{R}^{maxDef(r),nz})^C, \texttt{[ClockedSystem]})_{L^C_\Pi}$, the path will therefore be extended to an infinite path with repeated occurrences of $\rho(i)$-terms. This infinite $\rho(i)$-loop is the same as the infinite $\pi(\gamma(i))$-loop caused by the repeated application of the maximal 0-time tick step. In this case, α can be defined by $\alpha(i+1) = \gamma(i)+1$ and $\alpha(j+1) = \alpha(j)+1$ for $j > i$. It is easy to see that the resulting path indeed gives a stuttering simulation.

5. ∞ tick step: In this case we know that the remainder of π consists of an infinite sequence of ∞ tick steps, since by assumption *TR5* in Definition 6, any ∞ step can be followed by an ∞ step, and by assumption *TR4*, it cannot be followed by anything else. We define the rest of ρ to be *some* infinite sequence of ∞ tick steps. Since we have assumed tick-stabilization w.r.t. P, it follows that each term $\pi(j)$ for $j > \gamma(i)$ is $\simeq_P$-equal to $\pi(\gamma(i)+1)$ (with the possible modification if the first ∞ ticks are 0-time ticks as explained below). Since by the definition of an ∞ tick step, $t \xrightarrow[\infty]{r'} t'$ implies that $t \xrightarrow[\infty]{r} t''$ for some t'', we can just construct the rest of ρ by adding ∞ tick steps with duration r. We also define $\alpha(i+1) = \gamma(i)+1$, and $\alpha(j+1) = \alpha(j)+1$ for all $j > i$. It follows from the fact that all terms reachable from $\pi(\gamma(i))$ in ∞ tick steps are $\simeq_P$-equivalent, so that we are left with a $\simeq_P$-simulation. It may be that the

first ∞ tick steps from $\pi(\gamma(i))$ have duration 0. These are just ignored; that is, $\gamma(i)$ is just moved forward and $\alpha(i+1)$ is defined to be equal to the index of the first element reached either by applying an ∞ tick step with duration greater than 0 or by applying an instantaneous rule. Timed fairness ensures that there can be no infinite sequence of 0-time ∞ tick steps.

6. Non-maximal tick step $t \xrightarrow[1]{r} t'$: We consider the next sequence of steps in π. These following steps are either: (1) an infinite sequence of non-maximal tick steps, or (2) a sequence of non-maximal tick steps followed by either: (a) a maximal tick step, (b) an ∞ tick step, (c) an application of an instantaneous rule, or (d) a deadlock. We first consider case (1). Assume that the total duration associated with t $(= \pi(\gamma(i)))$ is Δ, and assume that the maximal time elapse possible from $\pi(\gamma(i))$ is r_m. Then, the system *could* have gone to a state with total duration $\Delta + r_m$. It follows from assumption *TR2* in Definition 6 that if $t \xrightarrow[1]{r} t'$ is a non-maximal tick, then there is a *maximal* tick step from t' with duration r' for $r + r' = r_m$, so that time could again have advanced to a total duration $\Delta + r_m$ in the second step. And so on. But then we have an infinite sequence of ticks where, at each step, a total duration of $\Delta + r_m$ could have been reached, but is never reached (since reaching it is done with a maximal step). But this is impossible, since it would break the non-Zeno-ness of timed fair paths. Case (2a): a sequence of non-maximal steps followed by a maximal tick. The total duration of this sequence is greater than 0, since otherwise all steps would be maximal tick steps. Let $\pi(\gamma(i)) = t = t_0 \xrightarrow[1]{r_1} t_1 \xrightarrow[1]{r_2} \cdots t_{n-2} \xrightarrow[1]{r_{n-1}} t_{n-1} \xrightarrow[max]{r_n} t_n$. By assumption *TR3*, there is a maximal tick $t_{n-2} \xrightarrow[max]{r_{n-1}+r_n} t_n$. Continuing in this way, there is a maximal tick $t_0 \xrightarrow[max]{\sum_{k=1}^{n} r_k} t_n$. The above sequence of ticks is represented by the latter tick step in ρ. That is, $\rho(i+1) = t_n$ and $\gamma(i+1) = \gamma(i) + n$. As for $\alpha(i+1)$, the tick-stability assumption states that there is a $k < n$ such that $t_1, \ldots, t_k$ are $\simeq_P$-equivalent to t_0, and the rest of the states are $\simeq_P$-equivalent to t_n. $\alpha(i+1)$ is defined accordingly to be the first position in the above chain where the term t_l is $\simeq_P$-equivalent to t_n. Then, the system up to $i+1$ is a $\simeq_P$-simulation. Case (2b) is impossible. Time-robustness implies that if $t \xrightarrow[1]{r} t'$ is a non-maximal tick step, then there is a maximal step from t', and then there cannot be a ∞-step from t'. The same argument also disqualifies case (2d). Finally, case (2c) is impossible due to the assumption that an instantaneous rule should never be applicable after non-maximal tick steps.

This ends the proof that constructs the path ρ and shows that $\simeq_P$ is a stuttering simulation from the timed fair paths in $\mathcal{K}(\mathcal{R}^C, \texttt{[ClockedSystem]})_{L^C_\Pi}$ to $\mathcal{K}((\mathcal{R}^{maxDef(r),nz})^C, \texttt{[ClockedSystem]})_{L^C_\Pi}$. That $\simeq_P$ is also a *strict* stuttering P-simulation follows directly from Definition 10, where $t \simeq_P t'$ is defined to hold if and only if $L(t) \cap P = L(t') \cap P$. $\square$

The converse of the above proposition also holds:

Proposition 12 *The relation* $\simeq_P$ *is a strict stuttering* P*-simulation* $\simeq_P : \mathcal{K}((\mathcal{R}^{maxDef(r),nz})^C, \texttt{[ClockedSystem]})_{L^C_\Pi} \to \mathcal{K}(\mathcal{R}^C, \texttt{[ClockedSystem]})_{L^C_\Pi}$ *when* $\mathcal{K}(\mathcal{R}^C, \texttt{[ClockedSystem]})_{L^C_\Pi}$ *is restricted to timed fair paths.*

PROOF. We prove that each path π in $\mathcal{K}((\mathcal{R}^{maxDef(r),nz})^C, \texttt{[ClockedSystem]})_{L^C_\Pi}$ can be stuttering $\simeq_P$-simulated by a *timed fair* path ρ in $\mathcal{K}(\mathcal{R}^C, \texttt{[ClockedSystem]})_{L^C_\Pi}$.

The corresponding path ρ is π itself. Since $\mathcal{R}^{maxDef(r),nz}$ is a restricted version of $\mathcal{R}$, it is only necessary to make sure that deadlocks in $\mathcal{R}^{maxDef(r),nz}$ can be "simulated" in $\mathcal{R}$, and that each π represents a *timed fair* path. The restrictions in $\mathcal{R}^{maxDef(r),nz}$ are: (i) no 0-time ticks; (ii) always increase time by time r in an ∞ tick step; and (iii) always advance time maximally if possible.

Assume that there is deadlock caused by the inability to perform 0-time ticks. Since this leads to a deadlock from the given state, it means that no instantaneous rules could be used on the state. This setting can be simulated in $\mathcal{R}$ either by the same deadlock, or, if a 0-time tick is a possibility in $\mathcal{R}$, then ρ can consist of an infinite sequence of such ticks. Given that a 0-time tick should not change the state, a sequence of 0-time ticks is essentially the same as an infinite sequence of identity rewrites caused by a deadlock. Furthermore, this infinite 0-time tick chain does *not* break the timed fairness requirement of $\mathcal{R}$-paths, since no instantaneous rule could be applied.

The fact that $\mathcal{R}^{maxDef(r),nz}$ advances time maximally does not lead to additional deadlocks in $\mathcal{R}^{maxDef(r),nz}$, since, by assumption *TR3* in Def. 6, a maximal tick can be taken if a non-maximal step can be taken. It is also easy to see that maximal ticks do not cause Zeno-ness which destroys timed fairness. Finally, an ∞ tick step can always be performed in $\mathcal{R}^{maxDef(r),nz}$ if it can be performed in $\mathcal{R}$, due to the definition of ∞ tick steps given in Definition 4. □

Theorem 13 *Let* $\mathcal{R}$ *be a time-robust real-time rewrite theory,* r *a time value in* $\mathcal{R}$ *greater than* 0*,* L_Π *a labeling function 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* $\mathcal{K}(\mathcal{R}^C, \texttt{[ClockedSystem]})_{L^C_\Pi}$*,* restricted to its timed fair paths*, and* $\mathcal{K}((\mathcal{R}^{maxDef(r),nz})^C, \texttt{[ClockedSystem]})_{L^C_\Pi}$.

PROOF. Follows directly from Propositions 11 and 12, since $\simeq_P$ is the same as $(\simeq_P)^{-1}$ because $\simeq_P$ is symmetric. □

Our main result is that maximal time sampling analysis is sound and complete:

Corollary 14 *Let $\mathcal{R}$, t_0, r, L_Π, and P be as in Theorem 13, and let Φ be an $LTL \setminus \{\bigcirc\}$ formula whose atomic propositions are contained in P. Then*

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

PROOF. Theorem 1 in [12] states that strict simulations reflect satisfaction of $ACTL^* \setminus \{\bigcirc\}$ formulas. The logic $LTL \setminus \{\bigcirc\}$ is a sublogic of $ACTL^* \setminus \{\bigcirc\}$, so that Φ holds for all timed fair paths in $\mathcal{K}(\mathcal{R}^C, \texttt{[ClockedSystem]})_{L_\Pi^C}$ starting from t_0 if and only if $\mathcal{K}((\mathcal{R}^{maxDef(r),nz})^C, \texttt{[ClockedSystem]})_{L_\Pi^C}, [t_0] \models \Phi$. This latter property is equivalent to $(\mathcal{R}^{maxDef(r),nz})^C, L_\Pi, t_0 \models \Phi$ according to Fact 5.15 (and the following statement about unbounded model checking) in [4]. The first property is the same as $\mathcal{R}, L_\Pi, t_0 \models^{tf} \Phi$ by Definition 8 as long as the set $timedFairPaths(\mathcal{R})_{t_0}$ is the same as $timedFairPaths(\mathcal{K}(\mathcal{R}^C, \texttt{[ClockedSystem]})_{L_\Pi^C})_{t_0}$. This latter property follows again from the fact that for each path in $paths(\mathcal{R})$ there is a corresponding path in $\mathcal{K}(\mathcal{R}^C, \texttt{[ClockedSystem]})_{L_\Pi^C}$ as shown in Fact 5.15 in [4]. Finally, the fact that we have proved bisimulation only with respect to a subset $P \subseteq \Pi$ should not cause problems as long as Φ does not contain any proposition from $\Pi \setminus P$. $\square$

4.5 Completeness of Time-Bounded Model Checking

Real-Time Maude's *time-bounded* model checking features have proved very useful for analyzing large applications [9,10]. 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. We defined in [4] 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 "`<> gt20 in time <= 22`" would *not* hold 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` in our semantics.

We denote by $timedFairPaths(\mathcal{R})_{t_0}^{\leq\Delta}$ the subset of $Paths(\mathcal{R})_{t_0}^{\leq\Delta}$ (see [4]) which contains all the timed fair paths starting in state t_0 that are chopped off at time Δ as explained in [4]. The satisfaction relation $\models^{tf}$ over timed fair paths is extended to time-bounded satisfaction in the obvious way:

Definition 15 $\mathcal{R}, L_\Pi, t_0 \models_{\leq\Delta}^{tf} \Phi$ *holds if and only if* $\pi, L_\Pi \models \Phi$ *holds for all* $\pi \in timedFairPaths(\mathcal{R})_{t_0}^{\leq\Delta}$.

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 16 *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 17 *Tick-invariance implies tick-stabilization.*

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

Theorem 18 *Given a time-robust real-time rewrite theory $\mathcal{R}$, a function L_Π 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 $LTL \setminus \{\bigcirc\}$ formula Φ whose atomic propositions are contained in P. Then,*

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

Proof (sketch) We sketch how the proof of Corollary 14 can be modified to allow time bounds by outlining how a path π in $timedFairPaths(\mathcal{R})^{\leq\Delta}_{t_0}$ can be $\simeq_P$-matched by a path ρ in $Paths(\mathcal{R}^{maxDef(r),nz})^{\leq\Delta}_{t_0}$. In particular, we use from Lemma 11 the fact that $\pi(\gamma(i))$ equals $\rho(i)$ as far as γ is defined.[6] Therefore, there is a appropriate simulation until the "last" maximal tick step or the last instantaneous rewrite step taken before the time bound. Let $\rho(k) = \pi(\gamma(k))$ denote this last "common" state. There are two interesting cases to consider for the next steps in ρ and π:

(1) ∞ tick steps can be taken from $\pi(\gamma(k))$, and either: (a) π does not contain further steps (because the first ∞ step could advance beyond the bound while ρ contains further (∞) steps (since $\pi(\gamma(k))$ has associated duration less than the time bound minus r), or (b) the other way around.
(2) π contains a sequence of non-maximal tick steps that ρ cannot perform.

In case (1a), there is a self-loop after $\pi(\gamma(k))$ in π and some more ∞-ticks in ρ. Since $\simeq_P$ is unchanged by ∞-steps, each $\rho(l)$, for $l > k$ is $\simeq_P$-equivalent to $\rho(k) = \pi(\gamma(k)) = \pi(\gamma(k)+1) = \ldots$, and we have the desired simulation. Case (1b) has the same simulation (in the "opposite" direction). In case (2), there

[6] $\rho(i)$ is a *clocked* term, meaning that that not only are the state parts of $\pi(\gamma(i))$ of $\rho(i)$ equivalent, but also the duration of the rewrite sequences leading to this state in π and ρ are also equivalent.

is a possibility to have a self-loop in ρ after $\rho(k)$, since time could advance beyond the time bound. Again, we have that, due to tick-invariance w.r.t. non-maximal tick steps, that each $\pi(\gamma(k)+l)$ is $\simeq_P$-equivalent to $\rho(k)$ (and therefore to $\rho(k+l)$) and we have the desired simulation. $\Box$

The completeness results in Corollary 14 and Theorem 18 carry over directly to unbounded and time-bounded search without lower time 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 Appendix A we prove that our clock specification is time-robust and that the unbounded and time-bounded search and model checking analyses in [4] are complete when using maximal time sampling.

5 Completeness for Object-Based Systems

Real-Time Maude inherits Maude's object model and is particularly well suited to specify real-time systems in an object-oriented way. An object of class C in a given state is represented as a term $\texttt{<}\ O : C \mid att_1 : val_1\texttt{,}\ldots\texttt{,}\ att_n : val_n\ \texttt{>}$, where O is the objects' identifier, and val_i the current value of the attribute att_i. A message is a term of the built-in sort `Msg`. If m is such a message, the term $\texttt{dly}(m, r)$ defines a "delayed" message which will be "ripe" in time r. In a concurrent object-oriented system, the state is a term of the built-in sort `Configuration` and is a *multiset* of objects and (delayed and ripe) messages. Multiset union for configurations is denoted by a juxtaposition operator (empty syntax) that is declared associative and commutative and having the `none` multiset as its identity element, so that order and parentheses do not matter, and so that rewriting is *multiset rewriting* supported directly in Maude. The dynamic behavior of object systems is axiomatized by specifying each of its transition patterns by a rewrite rule. For example, the instantaneous rule

```
rl [l] : m(O,w) < O : C | a1 : x, a2 : O' >  =>
                < O : C | a1 : x + w >  dly(m'(O'),x) .
```

defines a family of 0-time transitions in which a (ripe) message `m`, with parameters `O` and `w`, is read by an object `O` of class `C`, with the effect of altering the attribute `a1` of `O` and of sending a new message `m'(O')` with delay `x`.

In [4] we suggest to specify "flat" object-based real-time systems [7] by functions

[7] A "flat" object-oriented system is one where the object attributes do not contain configurations (or other "dynamic" elements).

```
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: [8]

```
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, including the AER/NCA protocol suite [9] [10] and the OGDC algorithm [9,8]. 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 19 *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 time value, and let the time domain be* linear *(see [19]). Then, $\mathcal{R}$ is time-robust if the following conditions are satisfied for all appropriate ground terms t and r, r':*

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

---

[8] `NEConfiguration` is a subsort denoting *non-empty* configurations.

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

PROOF. Requirement *TR1* is satisfied by the form of the tick rule.

For *TR2* and *TR3*, consider a maximal tick step $\{t\} \xrightarrow[max]{mte(t)} \{\delta(t, mte(t))\}$ and a non-maximal tick step $\{t\} \xrightarrow[1]{r} \{\delta(t,r)\}$. Since $r < mte(t)$ and $mte(\delta(t,r)) = mte(t) \dot{-} r$, there is also a maximal tick step $\{\delta(t,r)\} \xrightarrow[max]{mte(t) \dot{-} r} \{\delta(\delta(t,r), mte(t) \dot{-} r)\}$. We need to prove that:

- The resulting durations are the same: $mte(t) = r + (mte(t) \dot{-} r)$.
- The resulting states $\{\delta(t, mte(t))\}$ and $\{\delta(\delta(t,r), mte(t) \dot{-} r)\}$ are the same.

Since $r < mte(t)$, it follows from the eighth equation of the module *TIME* [19] that $mte(t) = r + (mte(t) \dot{-} r)$. That the resulting states are equivalent follows from *OO3*, given that we have shown that the durations are equivalent.

TR4: Since the specification is assumed to be "flat," rewrites only take place at the outermost level of a configuration, so that for an instantaneous rule $l_i \longrightarrow l_i'$ a rewrite step has the form $\{\sigma(l_i)\ C'\} \xrightarrow{inst} \{\sigma(r_i)\ C'\}$. Assume that such a step follows a non-maximal tick step $\{C\} \xrightarrow[1]{r} \{\sigma(l_i)\ C'\}$. Since this is not a maximal step, there is a maximal step from $\{\sigma(l_i)\ C'\}$, which means that $mte(\sigma(l_i)\ C')$ cannot be 0. However, *OO4* implies that $mte(\sigma(l_i)\ C') = \mathtt{min}(mte(\sigma(l_i)), mte(C')) = \mathtt{min}(0, mte(C'))$, which equals 0 since $0 \leq mte(C')$.

TR5 follows from *OO1*: If $\{C\} \xrightarrow[1]{r} \{\delta(C,r)\}$ is an ∞ tick step, then $mte(C)$ must be `INF`, and then *OO1* implies that $mte(\delta(C,r)) = \mathtt{INF} \dot{-} r$, which is defined in [19, Module $TIME_\infty$] to be `INF`.

TR6 follows from *OO2*.

Finally, we show that it is sufficient to prove *OO1*, *OO2*, and *OO3* for *single* elements in a configuration.

For *OO1*, we must prove that $mte(C,r) = mte(C) \dot{-} r$ and $mte(C',r) = mte(C') \dot{-} r$ imply that $mte(\delta(C\ C', r)) = mte(C\ C') \dot{-} r$. Since *mte* requires *linear* time domains, we assume that $mte(C) \leq mte(C')$. Then, $mte(\delta(C\,C', r)) = mte(\delta(C,r)\ \delta(C',r)) = \mathtt{min}(mte(\delta(C,r)), mte(\delta(C',r))) = \mathtt{min}(mte(C) \dot{-} r, mte(C') \dot{-} r')$. According to the last equation in the theory *TIME*, $mte(C) \dot{-} r \leq mte(C') \dot{-} r$ when $mte(C) \leq mte(C')$, so that $\mathtt{min}(mte(C) \dot{-} r, mte(C') \dot{-} r')$ equals $mte(C) \dot{-} r$. The right-hand side $mte(C\ C') \dot{-} r$ equals $\mathtt{min}(mte(C), mte(C')) \dot{-} r$, which equals $mte(C) \dot{-} r$ when $mte(C) \leq mte(C')$.

For *OO2*, we must prove that $\delta(C,0) = C$ and $\delta(C',0) = C'$ imply that $\delta(C\ C', 0) = C\ C'$, which follows directly from the definition of δ.

For *OO3*, we must prove that $\delta(\delta(C,r), r') = \delta(C, r+r')$ and $\delta(\delta(C',r), r') = \delta(C', r+r')$ implies that $\delta(\delta(C\ C', r), r') = \delta(C\ C', r+r')$. This follows from

$\delta(\delta(C\ \ C', r), r') = \delta(\delta(C, r)\ \ \delta(C', r), r') = \delta(\delta(C, r), r')\ \ \delta(\delta(C', r), r') = \delta(C, r + r')\ \delta(C, r + r') = \delta(C\ \ C', r + r')$.

Finally, we must show that *OO1* to *OO3* always hold for the empty configuration `none`. For *OO1*, we must prove $mte(\delta(\texttt{none}, r)) = mte(\texttt{none}) \dot{-} r$, which holds since $mte(\delta(\texttt{none}, r)) = mte(\texttt{none}) = \texttt{INF} = \texttt{INF} \dot{-} r = mte(\texttt{none}) \dot{-} r$. It is easy to see that *OO2* and *OO3* also hold for `none`. □

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 [22]. 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.

5.1 A Small Network Protocol Example

In [4] we presented different variations of a very simple protocol for computing the *round trip time* of message transmission between pairs of nodes in a network. In one of the versions, it takes each message *exactly* time `MIN-TRANS-TIME` to travel from its source to its destination.[10] The system is specified as an object-oriented system according to the specification techniques above (with `delta` for δ, etc.). In Appendix B we first present the entire specification of the protocol and prove that the specification satisfies requirements *OO1* to *OO4* for time-robustness. We then consider the search and model checking commands in [4], and prove that the search patterns and atomic propositions occurring in them are tick-invariant. We can therefore conclude that all the analyses performed in [4] on this system are sound and complete when using the maximal time sampling strategy.

[10] Computing the round trip time in this setting is admittedly not very interesting, but the example is fairly small yet contains timers, messages delays, etc.

The AER/NCA active network multicast protocol suite is the largest Real-Time Maude application analyzed so far [10]. Our specification of AER/NCA follows the above specification guidelines 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 24 and where the `mte` of a message is 0.

Since `delta` is defined to have co-arity `ObjectConfiguration`, it is sufficient to prove *OO1*, *OO2*, and *OO3* for objects. 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 value of each timer 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 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 Appendix B and satisfaction of the Requirements *OO1*, *OO2*, and *OO3* can be proved in exactly the same way; that is, by using the equations for `mte` and `delta` and by proving the following trivial facts:

- ti `+` `0` $= ti$,
- ti `monus` `0` $= ti$,
- (ti `monus` r) `monus` $r' = ti$ `monus` (r `+` r'), and
- associativity of `+`

for ti a time value or `INF`, and for time values r, r'. 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 $\texttt{mte}(\sigma(l_i)) = 0$ for instances of left-hand sides that do not contain messages. It is easy to inspect each of the 76 instantaneous rules in

AER/NCA to see that the left-hand side contains a message or has `mte` 0.

Finally, we must prove tick-invariance of the search patterns and the atomic propositions in the analysis commands in [10]. Remarkably, *none* of the search patterns or 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 protocol suite and all but one of the analysis commands in [10] satisfy the requirements for the maximal time sampling analysis to be sound and complete.

AER/NCA is essentially parametric in the time domain. 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.

5.2.1 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 specifying probabilistic behavior, it is possible to simulate such systems using a pseudo-random number generator `random`, and setting the timer to a pseudo-random number between 0 and *implosionSuppressionInterval*.

For probabilistic systems, completeness of our analyses therefore 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 can model probabilistic behavior by (unquantified) nondeterministic behavior by letting the the "random" value be a new variable, only occurring in the right-hand side of the rule, which can be given *any* value in the desired interval.

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.

5.3 The OGDC Wireless Sensor Network Algorithm

We have recently used Real-Time Maude to model, simulate, and analyze the state-of-the-art OGDC density control algorithm for wireless sensor networks [8,9]. Our object-based specification of OGDC uses the specification techniques above. Given the complexity of the specification, it is remarkable how easy it is to prove time-robustness. 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 $(x \texttt{ monus } y) \texttt{ monus } z = x \texttt{ monus } (y + z)$. Tick-invariance of the propositions and search patterns used in the analyses in [9] follows trivially, since `delta` does not modify the attributes that define these patterns and propositions.

Besides the fact that proving completeness of maximal time sampling analysis 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 [9] terminated within several hours when we used the time sampling strategy which increases time by one millisecond in each tick step.

OGDC is also 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 5.2, 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.

6 Conclusions

We have presented general criteria that, under the maximal time sampling strategy, guarantee completeness of Real-Time Maude formal analyses. 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 real-time 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 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.

References

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

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

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

[4] P. C. Ölveczky, J. Meseguer, Semantics and pragmatics of Real-Time Maude, Higher-Order and Symbolic ComputationTo appear.

[5] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, J. F. Quesada, Maude: Specification and programming in rewriting logic, Theoretical

Computer Science 285 (2002) 187–243.

[6] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, C. Talcott, Maude Manual (Version 2.2), `http://maude.cs.uiuc.edu` (December 2005).

[7] P. C. Ölveczky, M. Caccamo, Formal simulation and analysis of the CASH scheduling algorithm in Real-Time Maude, in: L. Baresi, R. Heckel (Eds.), Fundamental Approaches to Software Engineering (FASE'06), Vol. 3922 of Lecture Notes in Computer Science, Springer, 2006, pp. 357–372.

[8] P. C. Ölveczky, S. Thorvaldsen, Formal modeling and analysis of wireless sensor network algorithms in Real-Time Maude, in: 20th International Parallel and Distributed Processing Symposium (IPDPS 2006), IEEE Computer Society Press, 2006.

[9] S. Thorvaldsen, 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).

[10] P. C. Ölveczky, J. Meseguer, C. L. Talcott, Specification and analysis of the AER/NCA active network protocol suite in Real-Time Maude, Formal Methods in System DesignTo appear.

[11] P. Manolios, Mechanical verification of reactive systems, Ph.D. thesis, University of Texas at Austin (2001).

[12] N. Martí-Oliet, J. Meseguer, M. Palomino, Theoroidal maps as algebraic simulations, in: J. L. Fiadeiro, P. Mosses, F. Orejas (Eds.), Proc. WADT 2004, Vol. 3423 of Lecture Notes in Computer Science, Springer, 2005, pp. 126–143.

[13] P. C. Ölveczky, J. Meseguer, Abstraction and completeness for Real-Time Maude, Electronic Notes in Theoretical Computer ScienceTo appear.

[14] M. Palomino, J. Meseguer, N. Martí-Oliet, A categorical approach to simulations, in: J. L. Fiadeiro, N. Harman, M. Roggenbach, J. J. M. M. Rutten (Eds.), CALCO 2005, Vol. 3629 of Lecture Notes in Computer Science, Springer, 2005, pp. 313–330.

[15] J. Meseguer, Membership algebra as a logical framework for equational specification, in: F. Parisi-Presicce (Ed.), Proc. WADT'97, Vol. 1376 of Lecture Notes in Computer Science, Springer, 1998, pp. 18–61.

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

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

[18] M. C. Browne, E. M. Clarke, O. Grumberg, Characterizing finite Kripke structures in propositional temporal logic, Theoretical Computer Science 59 (1988) 115–131.

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

[20] P. C. Ölveczky, J. Meseguer, Real-Time Maude 2.1, Electronic Notes in Theoretical Computer Science 117 (2005) 285–314.

[21] P. C. Ölveczky, Real-Time Maude 2.1 Manual, `http://www.ifi.uio.no/RealTimeMaude/` (2005).

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

A The Clock Example Revisited

We show in this section that the simple clock system in Example 2 is time-robust, and show that the propositions in the analysis commands used to analyze a similar clock in [4] are tick-invariant or tick-stabilizing. Therefore, using the maximal time sampling strategy will provide a complete analysis of this clock with dense time domain.

We first prove that our specification satisfies the requirements for time-robustness:

TR1: Consider a tick step $\texttt{\{clock(}r\texttt{)\}} \xrightarrow{r'} \texttt{\{clock(}r \texttt{ + } r'\texttt{)\}}$. This is a maximal step for $r' = 24 \texttt{ monus } r$, since if r' were greater than $24 \texttt{ monus } r$, then the tick could not be applied (the condition of the rule would not hold, since, for x, y rational numbers, $x > y \;\Rightarrow\; x \not\leq y$). If r' is smaller than $24 \texttt{ monus } r$, then it is by definition a non-maximal step. A tick step $\texttt{\{stopped-clock(}r\texttt{)\}} \xrightarrow{r'} \texttt{\{stopped-clock(}r\texttt{)\}}$ is an ∞ tick step, since it can be applied for any time value r'.

TR2: Let $\texttt{\{clock(}r\texttt{)\}} \xrightarrow[max]{24 \texttt{ monus } r} \texttt{\{clock(}r \texttt{ + (24 monus } r\texttt{))\}}$ be a maximal tick step, and let $\texttt{\{clock(}r\texttt{)\}} \xrightarrow[1]{r'} \texttt{\{clock(}r \texttt{ + } r'\texttt{)\}}$ be a non-maximal tick step; that is, $r' < 24 \texttt{ monus } r$. By the definition of the tick rule, there is a maximal tick rule $\texttt{\{clock(}r\texttt{+}r'\texttt{)\}} \xrightarrow[max]{24 \texttt{ monus } (r+r')} \texttt{\{clock(}r\texttt{+}r'\texttt{+ (24 monus(}r \texttt{ + } r'\texttt{)))\}}$. Then we just need to show that the durations $24 \texttt{ monus } r$ and $r'+(24 \texttt{ monus } (r+r'))$ are the same. This equivalence does not hold in general for `monus`. However, since $0 \leq r' < (24 \texttt{ monus } r)$, it follows that $r < 24$, and hence $24 \texttt{ monus } r$ equals $24 - r$. Likewise, since $r' < (24 - r)$ (assumed non-maximal step), we have that $r + r' < 24$. Hence, $24 \texttt{ monus } (r + r')$ equals $24 - (r + r')$. The desired equality therefore reduces to proving $r' + (24 - (r + r')) = 24 - r$, which clearly holds.

TR3: Let $\texttt{\{clock(}r\texttt{)\}} \xrightarrow[1]{r'} \texttt{\{clock(}r \texttt{ + } r'\texttt{)\}}$ be a non-maximal tick step, and let $\texttt{\{clock(}r\texttt{+}r'\texttt{)\}} \xrightarrow[max]{24 \texttt{ monus } (r+r')} \texttt{\{clock(}r\texttt{+}r'\texttt{+ (24 monus (}r \texttt{ + } r'\texttt{)))\}}$ be the following maximal tick step. Then we just need to show that the "composition"

$\texttt{\{clock(}r\texttt{)\}} \xrightarrow{r' + (24 \texttt{ monus } (r+r'))} \texttt{\{clock(}r \texttt{ + } r' \texttt{ + (24 monus (}r \texttt{ + } r'\texttt{)))\}}$ is the same as the maximal tick step $\texttt{\{clock(}r\texttt{)\}} \xrightarrow[max]{24 \texttt{ monus } r} \texttt{\{clock(}r \texttt{ + (24 monus } r\texttt{))\}}$. We proved for *TR2* that 24 `monus` r and $r' + (24 \texttt{ monus } (r + r'))$ are the same. It then follows from the congruence property of equational logic that $\texttt{\{clock(}r \texttt{ + (24 monus } r\texttt{))\}}$ and $\texttt{\{clock(}r \texttt{ + } r' \texttt{ + (24 monus (}r \texttt{ + } r'\texttt{)))\}}$ are the same.

TR4: An instantaneous step cannot follow a non-maximal or an ∞ tick step. That is, a non-maximal step or an ∞ step cannot lead to `{clock(24)}`, since a non-maximal tick $\texttt{\{clock(}r\texttt{)\}} \xrightarrow[1]{r'} \texttt{\{clock(}r \texttt{ + } r'\texttt{)\}}$ implies (as shown above) that $r' < 24 - r$, so that $r + r' < 24$, which should lead to $r + r' \neq 24$.

TR5: An ∞ step can always follow an ∞ step, since an ∞ step does not change the state.

TR6: 0-time ticks do not change the state, since $r + 0$ equals r for any time value r.

This concludes the proof that our specification is time-robust. We next prove time-invariance or time-stabilization of the sets of propositions used in the analysis commands given in [4]. The first such command is

```
(tsearch [1] {clock(0)} =>* {clock(X:Time)} such that X:Time > 24 in time <= 99 .)
```

Since this is a *time-bounded* command, we need to show tick-invariance: If $t \xrightarrow[1]{r} t'$ is a tick step that is not a maximal tick step, then t should match the search pattern if and only if t' does so. We have previously seen that if $\texttt{\{clock(}r\texttt{)\}} \xrightarrow[1]{r'} \texttt{\{clock(}r \texttt{ + } r'\texttt{)\}}$ is a non-maximal tick step, then $r + r'$ is smaller than 24, and neither the left-hand side nor the right-hand side matches the search pattern. Tick-invariance under ∞ ticks is trivial, since the ∞ ticks do not change the state of the system. Since tick-invariance implies tick-stabilization, we obtain that the maximal strategy is complete for the next search command:

```
(utsearch {clock(0)} =>* {clock(X:Time)} such that X:Time > 24 .)
```

We next consider LTL model checking, and define propositions `clock-dead` (which holds for stopped clocks), `clock-is(`r`)` (which holds if the clock is not stopped and shows time r), and `clockEqualsTime` (which holds if the clock is not stopped and shows the total elapsed time in the system) as follows [4]:

```
(tmod MODEL-CHECK-DENSE-CLOCK is including TIMED-MODEL-CHECKER .
  protecting SIMPLIFIED-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)
```

It is worth noting that `clockEqualsTime` is a *clocked* proposition. The property we analyze is:

```
(mc {clock(0)} |=t clockEqualsTime U (clock-is(24) \/ clock-dead) in time <= 1000 .)
```

Since this is a time-bounded property, we must prove that each proposition in the formula is tick-invariant. Tick-invariance is trivial for ∞ tick steps, since they do not change the state. For the clocked proposition `clockEqualsTime`, tick-invariance must be shown w.r.t. the clocked version `{clock(`r`)} in time` x $\xrightarrow[1]{r'}$ `{clock(`r `+` r'`)} in time` x `+` r' of a non-maximal tick step. That is, it is sufficient to prove $(r = x) = (r + r' = x + r')$, which clearly holds. This clocked proposition is also invariant with respect to ∞ tick steps, since `clockEqualsTime` never holds for any `stopped-clock` state. That the proposition `{clock(24)}` is tick-invariant follows from the fact that we have shown above that when `{clock(`r`)}` $\xrightarrow[1]{r'}$ `{clock(`r`+`r'`)}` is a non-maximal tick step, then both r and $r+r'$ are different from 24. Finally, tick-invariance of `clock-dead` is trivial. Therefore, the above model checking analysis is complete.

B Proving Completeness of Real-Time Maude Analysis of a Simple Round Trip Time Protocol

This appendix first presents an object-oriented Real-Time Maude specification of a very simple protocol for finding the round trip time between pairs of nodes in a network. The protocol is described in [4] and is "parametric" in the assumptions about communication. In this appendix, we consider the case where it takes a message *exactly* time `MIN-TRANS-DELAY` to travel from source to destination, but where any ripe message *could* be lost.

It is worth noticing that the `dly` operator for messages is declared to have right identity `0`; this makes `dly(`m`,0)` equivalent to the ripe message m. Since we define `mte` on messages by the equation

```
  eq mte(dly(M:Msg, R)) = R .
```

the `mte` of a state with a ripe message is `0`, and therefore time cannot elapse while such a message is present in the state, forcing the application of a rule that consumes the message or the rule `messageLoss` that models the loss of a message.

The protocol goes as follows: When an object reads a `findRtt` message (rule

startSession), it sends a rttReq message, with the current time (as given by its clock attribute), to its neighbor, which is given by the object's nbr attribute. When the neighbor reads the rttReq message (rule rttResponse), it sends back an rttResp message without changing the original time stamp. When the originator receives the rttResp message with its own original time stamp, it can easily compute the round trip time by comparing the time stamp in the message with its current clock value (rule treatRttResp). Since messages may get lost, the initiator repeats this process if it has not received an rttResp messages within time MAX-DELAY after initiating a run. Therefore, it has a timer which is set to MAX-DELAY when the rtt process starts and which is turned off (i.e., set to INF) when an rttResp message is read. When the timer expires (i.e., reaches 0), the process starts all over again (rule tryAgain).

The functions delta and mte are defined as expected: delta increases the clock value and decreases the timer value and the message delays according to the elapsed time. mte gives the time until a timer expires or until a message becomes ripe.

The entire specification can then be given as follows, where the module RTT-I defines a suitable initial state with three nodes:

```
(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] .

  --- A ripe message may be lost:
  rl [messageLoss] : M:Msg NeC:NEConfiguration => NeC:NEConfiguration .

  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)

(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)
```

The following command searches for an (undesired) state reachable in time 100 that contains an object with a recorded `rtt` value greater than or equal to 4:

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

The following command searches whether we can reach a desired state where both `n1` and `n2` have recorded the expected `rtt` values:

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

Finally, in [4] we use the temporal logic model checker to prove that there are no *superfluous* messages being sent around in the system after an `rtt` value has been found. That is, if an object o has found an `rtt` value, then there is no `rttReq(`o', o, r`)` or `rttResp(`o, o', r`)` message with $r + \texttt{MAX-DELAY} > c$, for c the value of o's `clock`. The following module defines the proposition `superfluousMsg`:

```
(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)
```

The command

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

can then prove that there are no superfluous messages in the system within time `100`.

We first prove that the requirements $OO1$ to $OO4$ for time-robustness are satisfied in our specification. For $OO1$ to $OO3$, we must prove the properties for single objects and messages. An object has the form

```
< o : Node | clock : r, timer : ti, nbr : o', rtt : ti' >
```

and a (delayed or ripe) message has the form `dly(`m`,`r`)` since a ripe message m is identical to `dly(`m`,0)` due to `dly` having right identity element `0`.

OO1: We must prove `mte(delta(`t`,`r`))` $=$ `mte(`t`) monus` r, for t a message or an object. Using the equations we get:

```
mte(delta(m, r)) = mte(dly(m, r' monus r)) = r' monus r =
mte(dly(m, r)) monus r.

mte(delta(< o : Node | clock : r', timer : ti, nbr : o', rtt : ti >, r)) =
mte(< o : Node | clock : r' + r, timer : ti monus r, nbr : o', rtt : ti >) =
ti monus r =
mte(< o : Node | clock : r', timer : ti, nbr : o', rtt : ti >) monus r.
```

OO2: $\delta(t, 0) = t$: For objects, the desired equality

```
delta(< o : Node | clock : r', timer : ti, nbr : o', rtt : ti >, 0) =
      < o : Node | clock : r', timer : ti, nbr : o', rtt : ti >
```

follows trivially from the fact that

```
delta(< o : Node | clock : r', timer : ti, nbr : o', rtt : ti >, 0) =
      < o : Node | clock : r' + 0, timer : ti monus 0, nbr : o', rtt : ti >
```

and that $r + 0$ equals r for all r, and that ti `monus` 0 equals ti for all ti of sort `TimeInf`. For messages, we must prove `delta(dly(`m`,`r`),0)`$=$ `dly(`m`,`r`)`, which holds since `delta(dly(`m`,`r`),0)` is defined to be `dly(`m`,`r `monus 0)`, which equals `dly(`m`,`r`)`.

OO3 holds since $+$ is associative and (t `monus` r) `monus` r' equals t `monus` ($r+$ r'). Finally, for $OO4$, 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
```

where the second equality follows from the fact that `dly` has right identity element `0`, so that `rttReq(O, O', R)` equals `dly(rttReq(O, O', R), 0)`.

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

We can therefore conclude that our specification is time-robust. We now prove tick-invariance of the propositions and search patterns in the analysis commands.

The search patterns do not mention any attribute which is changed by `delta`, so they are tick-invariant. As for the atomic proposition `superfluousMsg`, its satisfaction 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 `mte(`t`)` 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`.

Since we have shown both time-robustness of our specification and tick-invariance of the search patterns and propositions involved in our analysis, we can conclude that these analyses are sound and complete when using the maximal time sampling strategy.