Timed CTL Model Checking in Real-Time Maude[⋆]

Daniela Lepri[1], Erika Ábrahám[2], and Peter Csaba Ölveczky[1,3]

[1] University of Oslo, Norway
[2] RWTH Aachen University, Germany
[3] University of Illinois at Urbana-Champaign, USA

Abstract. This paper presents a timed CTL model checker for Real-Time Maude and its semantic foundations. In particular, we show that our timed CTL model checking procedure is sound and complete for a fairly large class of systems. An important benefit of our model checker is that it also automatically provides a timed CTL model checker for subsets of modeling languages, like Ptolemy II and (Synchronous) AADL, which have Real-Time Maude model checking integrated into their tool environments.

1 Introduction

Real-Time Maude [28] extends Maude [13] to support the formal modeling and analysis of real-time systems in rewriting logic. Real-Time Maude is characterized by its expressiveness and generality, natural model of object-based distributed real-time systems, the possibility to define any computable data type, and a range of automated formal analysis such as simulation, reachability and temporal logic model checking. This has made it possible to successfully apply the tool to a wide range of real-time systems, including advanced state-of-the-art wireless sensor network algorithms [16, 29], scheduling algorithms requiring unbounded queues [24], and routing protocols [30].

Real-Time Maude's expressiveness and generality also make it a suitable semantic framework and analysis tool for modeling languages for real-time systems [22]. For example, the tool has been used to formalize the industrial avionics modeling standard AADL [23], a synchronous version of AADL [6], (subsets of) Ptolemy II discrete-event (DE) models [7], the web orchestration language Orc [2], etc. Real-Time Maude formal analysis has been integrated into the tool environment of some of these languages, enabling a model engineering process that combines the convenience of an intuitive modeling language with formal analysis.

In Real-Time Maude, the data types of the system are defined by an algebraic equational specification, and the system's instantaneous transitions are modeled

[⋆] This work was partially supported by the Research Council of Norway through the Rhytm project, by the DAADppp HySmart project, and by AFOSR Grant FA8750-11-2-0084.

by (instantaneous) rewrite rules. Time advance is modeled explicitly by so-called *tick (rewrite) rules* of the form `{`t`} => {`t'`} in time` u `if` $cond$, where `{_}` is an operator that encloses the entire global state, and the term u denotes the *duration* of the rewrite. Real-Time Maude is parametric in the time domain, which may be discrete or dense. For dense time (in particular), tick rules typically have the form `{`t`} => {`t'`} in time` x `if` x `<=` d `/\` $cond$, where x is a new variable not occurring in t, d, or $cond$. This form of the tick rules ensures that any moment in time (within time d) *can* be visited, also for a dense time domain.

Real-Time Maude extends Maude's rewriting, search, and linear temporal logic model checking features to the timed case. For dense time, it is of course not possible to execute all possible rewrite sequences. The fairly restrictive timed automaton formalism [3] trades expressiveness for decidability of key properties for dense/continuous time, since the state space can be divided into a finite number of "clock regions" so that any two states in the same region satisfy the same properties. Such a quotient seems hard to achieve for the much more expressive real-time rewrite theories. Instead, the general approach taken in Real-Time Maude is to use *time sampling strategies* to instantiate the new variable x in the tick rules, and to analyze the resulting specification instead of the original one. One such strategy advances time by a fixed amount Δ in each application of any tick rule. The *maximal* time sampling strategy advances time as much as possible (i.e., by the given instance of d in the above tick rules) in each application of a tick rule. Although the fixed-increment strategy could cover all possible behaviors in the original system when the time domain is discrete, the maximal time sampling typically only analyzes a *proper subset* of all the possible behaviors. However, in [26], it is shown that for a fairly large set of real-time systems appearing in practice, the maximal time sampling strategy yields sound and complete analyses for untimed LTL properties. For example, systems where events are triggered by the arrival of messages or by the expiration of some "timer", and where time elapse does not change the valuation of the atomic propositions (this requirement almost always holds, since time elapse typically only changes timers and clocks, whose values are rarely relevant for temporal logic properties) satisfy the requirements for maximal time sampling analyses to be sound and complete. In particular, maximal time sampling analyses are sound and complete for most of the large applications and languages mentioned above, including the wireless sensor networks and the Ptolemy II models, that are systems that cannot be formalized by, say, timed automata.

Until recently, Real-Time Maude could only analyze *untimed* temporal logic properties, but not quantitative properties such as "the airbag must deploy within 5ms of a crash," or "the ventilator machine cannot be turned off more than once every 10 minutes." This paper presents a model checker for Real-Time Maude for a timed temporal logic, namely, TCTL [5], which is an extension of the branching time logic CTL in which the temporal operators are annotated with a time interval, so that, for example, the formula $E\ \varphi_1\ U_{[2,4]}\ \varphi_2$ holds if there is a path in which φ_2 holds after some time $2 \leq r \leq 4$ and where φ_1 holds in all states until then.

Going from untimed temporal logic to a timed temporal logic presents at least two significant challenges for Real-Time Maude:

1. What is the intended semantics of a Real-Time Maude specification with the above tick rule w.r.t. timed temporal logic properties? For example, given a tick rule `{`$f(y)$`} => {`$f(y+x)$`} in time` x `if` x `<=` $3-y$, should the property $AF_{[1,2]}$ `true` (for all paths, a state satisfying `true` will be reached in some time between 1 and 2) hold for initial state `{`$f(0)$`}`? There are paths (e.g., jumping directly from `{`$f(0)$`}` to `{`$f(3)$`}`) where no state is visited in the desired time interval. On the other hand, the above rule could be seen as a natural way to specify a *continuous* process from `{`$f(0)$`}` to `{`$f(3)$`}` in Real-Time Maude, so that the *intended* semantics should satisfy the above property. We address this problem by presenting two different semantics for the satisfaction of a TCTL formula in Real-Time Maude: the *pointwise* semantics takes all paths into account, including the one where we jump directly from time 0 to time 3, whereas the *continuous* semantics allows us to break up longer ticks in smaller steps.
2. The previous soundness and completeness results for maximal time sampling analyses no longer hold. In the example above, maximal time sampling would not satisfy the existential formula $E\,F_{[1,2]}$ `true`, although the original model satisfies it in both the continuous and the pointwise semantics.
   To achieve sound and complete time sampling analyses for cacca the continuous semantics and dense time, we always advance time by a time value $\frac{\bar{r}}{2}$, where $\bar{r}$ is the greatest common divisor of all the (non-zero) time values appearing in the annotations in the TCTL formula, as well as all the time values of the maximal tick steps reached from the initial state. For model checking w.r.t. the pointwise semantics, we also use this time increment, as well as any multiple of it. Notice that our approach requires the above mentioned greatest commond divisor $\bar{r}$ to exist on the considered dense time domain, which is, in turn, required to satisfy the functional theory `GCD-TIME-DOMAIN` introduced in Section 4. This, in general, does not rule out $\mathbb{R}$ as a possible time domain, as long as all the values used in the computation of $\bar{r}$ are not irrational numbers (i.e. all instantaneous transitions happen at rational time points and all time bounds in the TCTL formula are rational numbers).

This paper describes our model checker, its semantic foundations, and its implementation in Maude. Most importantly, we prove that our model checking approach provides sound and complete model checking analysis for the original theory.

An important benefit of our work is that a TCTL model checker for Real-Time Maude also gives us a TCTL model checker *for free* for Ptolemy II DE models, synchronous AADL models, and other modeling languages for which Real-Time Maude code can be generated. Indeed, as shown in this paper, our model checker has already been integrated into the Ptolemy II tool, allowing the user to model check TCTL properties of Ptolemy II models from within Ptolemy II. In particular, in Section 6 we illustrate the use of the model checker on a Ptolemy II DE model of a hierarchical traffic light system.

Related Work. The tools Kronos [34], Redlib [32], and TSMV [20] implement TCTL model checkers for, respectively, timed automata, linear hybrid automata, and timed Kripke structures. The tool Uppaal [9] is also based on timed automata and provides an efficient *symbolic* model checking procedure for a subset of *non-nested* TCTL properties. The Roméo tool [15, 10], based on a timed extension of Petri nets [1, 21], has an integrated timed model checker for some *non-nested* TCTL modalities, with the addition of bounded response properties. These formalisms are significantly less expressive than real-time rewrite theories [25], which makes their model checking problems decidable. The first approaches to model checking timed temporal properties for Real-Time Maude are described in [19, 33] and analyze important *specific* classes of timed temporal logic formulae (time-bounded response, time-bounded safety, and minimum separation), but only for *flat* object-based specifications. Unlike in [19, 33], our new model checker is not limited to specific classes of temporal logic properties, but offers the *full* TCTL. The new model checker is also not limited to flat object-oriented systems, but can also analyze any (sensible) Real-Time Maude model.

Paper Structure. Section 2 gives an introduction to real-time rewrite theories and Real-Time Maude. Section 3 describes our model checker and its semantics. Section 4 presents our soundness and completeness results. We discuss our model checker implementation in Section 5, and demonstrate its applicability on a case study in Section 6. Finally, concluding remarks are given in Section 7.

2 Real-Time Rewrite Theories and Real-Time Maude

A *rewrite theory* is a tuple (Σ, E, R), where (Σ, E) is a *membership equational logic theory* [13] that defines the state space of a system as an algebraic data type, with Σ a *signature* declaring sorts, subsorts, and function symbols, and E a set of *conditional equations* and *membership axioms*, and where R is a set of *labeled conditional rewrite rules* of the form $[l] : \ t \longrightarrow t'$ **if** *cond*, where l is a label, t, t' are Σ-terms, and *cond* is a conjunction of *rewrite conditions* $u \longrightarrow u'$, *equational conditions* $v = v'$, and *membership conditions* $w : s$, where u, u', v, v', w are Σ-terms and s is a sort in Σ. A rule is implicitly universally quantified by the variables appearing in t, t' and *cond*, and specifies a set of local *one-step transitions* in the system. Rules are applied modulo the equations E. The set $\mathbb{T}_{\Sigma/E,\mathtt{GlobalSystem}}$ of *states* is defined by the E-equivalence classes of ground terms of sort `GlobalSystem`.

Real-time rewrite theories [25] are used to specify real-time systems in rewriting logic. Rules are divided into *tick rules*, that model time elapse in a system, and *instantaneous rules*, that model instantaneous change. Formally a *real-time rewrite theory* $\mathcal{R}$ is a tuple $(\Sigma, E, R, \phi, \tau)$ such that

- (Σ, E, R) is a rewrite theory, with a sort `System` and a sort `GlobalSystem` with no subsorts or supersorts and with only one operator `{_}` : `System` →

`GlobalSystem` which satisfies no non-trivial equations; furthermore, for any $f : s_1 \dots s_n \to s$ in Σ, the sort `GlobalSystem` does not appear in $s_1 \dots s_n$.
- $\phi : \mathit{TIME} \to (\Sigma, E)$ is an equational theory morphism which interprets *TIME* in $\mathcal{R}$; the theory *TIME* [25] defines time abstractly as an ordered commutative monoid $(\mathit{Time}, 0, +, <)$. We write $0, +, \dots$ instead of $\phi(0), \phi(+), \dots$ and use `Time` for $\phi(\mathit{Time})$.
- τ is an assignment of a term τ_l of sort `Time` to each rewrite rule in R of the form $[l] : \texttt{\{}t\texttt{\}} \longrightarrow \texttt{\{}t'\texttt{\}}$ **if** *cond*. Such a rule is called a *tick rule* if $\tau_l \neq 0$; in this case τ_l denotes the duration of the step. Rules that are not tick rules are called *instantaneous rules* and are assumed to take zero time. Since the initial state has the form $\texttt{\{}t\texttt{\}}$, the form of the tick rules ensures that time advances uniformly in the whole system.

We write $t \xrightarrow{r} t'$ when t can be rewritten into t' in time r by a *one-step rewrite*. For convenience, we also write $t \xrightarrow{inst} t'$ instead of $t \xrightarrow{0} t'$ for one-step rewrites applying an *instantaneous* rule.[4]

A tick step $t \xrightarrow{r} t'$ is *maximal* if there is no $r' > r$ with $t \xrightarrow{r'} t''$ for some t''. A *timed path* in $\mathcal{R}$ is an infinite sequence $\pi = t_0 \xrightarrow{r_0} t_1 \xrightarrow{r_1} t_2 \xrightarrow{r_2} \cdots$ such that

- for all $i \in \mathbb{N}$, $t_i \xrightarrow{r_i} t_{i+1}$ is a one-step rewrite in $\mathcal{R}$; or
- there exists a $k \in \mathbb{N}$ such that $t_i \xrightarrow{r_i} t_{i+1}$ is a one-step rewrite in $\mathcal{R}$ for all $0 \leq i < k$, there is no one-step rewrite from t_k in $\mathcal{R}$, and $t_j = t_k$ and $r_j = 0$ for each $j \geq k$.

For paths π of the above form we define $d^\pi_m = \sum_{i=0}^{m-1} r_i$, $t^\pi_m = t_m$ and $r^\pi_m = r_m$. We call the timed path $\pi = t_0 \xrightarrow{r_0} t_1 \xrightarrow{r_1} t_2 \xrightarrow{r_2} \cdots$ a *timed fair path* if

- 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 $t_j \xrightarrow{r} t$ with $\Delta \leq d^\pi_j + r$ then there is an l with $\Delta \leq d^\pi_l$, and
- 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 $t_l \xrightarrow{inst} t_{l+1}$ is a one-step rewrite applying an instantaneous rule for some $l > k$.

We denote the set of all timed fair paths of $\mathcal{R}$ starting in t_0 by $\mathit{tfPaths}_{\mathcal{R}}(t_0)$. A term t is *reachable* from t_0 in $\mathcal{R}$ in time r iff there is a path $\pi \in \mathit{tfPaths}_{\mathcal{R}}(t_0)$ with $t^\pi_k = t$ and $d^\pi_k = r$ for some k. A path π is *time-divergent* iff for each time value $r \in$ `Time` there is an $i \in \mathbb{N}$ such that $d^\pi_i > r$.

The Real-Time Maude tool [27] extends the Maude system [13] to support the specification, simulation, and analysis of real-time rewrite theories. Real-Time Maude is parametric in the time domain, which may be discrete or dense, and defines a supersort `TimeInf` of `Time` which adds the infinity element `INF`. To cover all time instances in the dense time domain, tick rules should have one of the forms

[4] Notice that $t \xrightarrow{0} t'$ could be either a 0-time tick step or an instantaneous one-step rewrite.

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

where x is a new variable of sort `Time` not occurring in `{`t`}` and *cond*. This ensures that the tick rules can advance time by *any* amount in rules of the form $(*)$ or (§) and *any* amount less than or equal to u in rules of the form (†). Rules of these forms are called *time-nondeterministic* and are not directly executable in general, since many choices are possible for instantiating the new variable x.

In contrast to, e.g., timed automata, where the restrictions in the formalism allow the abstraction of the dense time domain by "clock regions" containing bisimilar states [3], for the more complex systems expressible in Real-Time Maude there is not such a discrete "quotient". Instead, Real-Time Maude executes time-nondeterministic tick rules by offering a choice of different *time sampling strategies* [27], so that only some moments in the time domain are visited. For example, the *maximal* time sampling strategy advances time by the maximum possible time elapse u in rules of the form (†) (unless u equals `INF`), and tries to advance time by a user-given time value r in tick rules having other forms. In the *default* mode each application of a time-nondeterministic tick rule will try to advance time by a time value r.

The paper [27] explains the semantics of Real-Time Maude in more detail. In particular, given a real-time rewrite theory $\mathcal{R}$ and a time sampling strategy σ, there is a real-time rewrite theory $\mathcal{R}^{\sigma}$ that has been obtained from $\mathcal{R}$ by applying the theory transformation corresponding to using the time sampling strategy σ when executing the tick rules. In particular, the real-time rewrite theory $\mathcal{R}^{maxDef(r)}$ denotes the real-time rewrite theory $\mathcal{R}$ where the tick rules are applied according to the maximal time sampling strategy, while $\mathcal{R}^{def(r)}$ denotes $\mathcal{R}$ where the tick rules are applied according to the default time sampling strategy (tick steps which advance time by 0 are not applied).

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

- $t = t'$ holds in the underlying equational theory for any 0-time tick step $t \xrightarrow{0} t'$.
- $t \xrightarrow{r+r'} t''$ if and only if there is a t' of sort `Time` such that $t \xrightarrow{r} t'$ and $t' \xrightarrow{r'} t''$.
- If $t \xrightarrow{r} t'$ is a tick step with $r > 0$, and $t' \xrightarrow{inst} t''$ is an instantaneous one-step rewrite, then $t \xrightarrow{r} t''$ is a *maximal* tick step.
- for $M = \{r \mid \exists t'.\ t \xrightarrow{r} t'\}$ we have that either there is a maximal element in M or M is the whole domain of `Time`.

Real-Time Maude extends Maude's *linear temporal logic model checker* to check whether each behavior, possibly up to a certain time bound, satisfies an (untimed) LTL formula. *State propositions* are terms of sort `Prop`. The labeling of states with propositions can be specified by (possibly conditional) equations of the form

```
{statePattern} |= prop = b
```

for b a term of sort `Bool`, which defines the state proposition *prop* to evaluate to b in all states matching the given pattern. We say that a set of atomic propositions is *tick-invariant* in $\mathcal{R}$ if tick rules do not change their values.

Since the model checking commands execute time-nondeterministic tick rules according to a time sampling strategy, only a subset of all possible behaviors is analyzed. Therefore, Real-Time Maude analysis is in general *not sound and complete*. However, the reference [26] gives easily checkable sufficient conditions for soundness and completeness, which are satisfied by many large Real-Time Maude applications.

3 Timed CTL Model Checking for Real-Time Maude

In untimed temporal logics it is not possible to reason about the *duration* of/between events. There are many *timed* extensions of temporal logics [4, 31, 11]. In this paper we consider TCTL [5] with interval time constraints on temporal operators.

3.1 Timed CTL

In *computation tree logic (CTL)* [5], a *state formula* specifies a property over the computation tree corresponding to the system behavior rooted in a given state. State formulae are constructed by adding universal (A) and existential (E) path quantifiers in front of *path formulae* to specify whether the path formula must hold, respectively, on *each* path starting in the given state, or just on *some* path. Path formulae are built from state formulae using the temporal operators X ("next") and U ("until"), from which F ("finally") and G ("globally") can be derived.

Timed CTL (TCTL) is a quantitative extension of CTL [5], where the scope of the temporal operators can be limited in time by subscripting them with time constraints. In this paper we consider an interval-bound version of TCTL where the temporal operators are subscripted with a time interval. A *time interval* I is an interval of the form $[a, b]$, $(a, b]$, $[a, b_\infty)$ or (a, b_∞), where a and b are values of sort `Time` and b_∞ is a value of sort `TimeInf`.

Definition 1. *Given a set Π of atomic propositions, TCTL formulae are built using the following abstract syntax:*

$$\varphi \quad ::= \quad \mathit{true} \quad | \quad p \quad | \quad \neg\varphi \quad | \quad \varphi \wedge \varphi \quad | \quad E\,\varphi\,U_I\,\varphi \quad | \quad A\,\varphi\,U_I\,\varphi$$

where $p \in \Pi$ and I is a time interval.

We omit the bound $[0, \infty)$ as subscript and we write $\leq b$, $< b$, $\geq a$ and $> a$ for $[0, b]$, $[0, b)$, $[a, \infty)$ and (a, ∞), respectively. We denote by TCTL$_{cb}$ the fragment of TCTL where all time bounds are of the form $[a, b]$ with $a < b$, or $[a, \infty)$.

3.2 Timed Kripke Structures and TCTL Semantics

The semantics of TCTL formulae is defined on Kripke structures. A *Kripke structure* is a transition system with an associated labeling function, which maps each state in the transition system to the set of atomic propositions that hold in that state.

A *timed Kripke structure* is a Kripke structure where each transition has the form $s \xrightarrow{r} s'$, where r denotes the duration of the transition step.

Definition 2. *Given a set of atomic propositions Π and a time domain $\mathcal{T}$, a* timed Kripke structure *is a triple $TK = (S, \xrightarrow{\mathcal{T}}, L)$ where S is a set of states, $\xrightarrow{\mathcal{T}} \subseteq S \times \mathcal{T} \times S$ is a* transition relation with duration*, and L is a labeling function $L : S \to \mathcal{P}(\Pi)$. The transition relation $\xrightarrow{\mathcal{T}}$ is* total*,[5] i.e., for each $s \in S$ there exist $r \in \mathcal{T}$, $s' \in S$ such that $(s, r, s') \in \xrightarrow{\mathcal{T}}$. We write $s \xrightarrow{r} s'$ if $(s, r, s') \in \xrightarrow{\mathcal{T}}$.*

We use a similar notation for timed paths in a timed Kripke structure $\mathcal{TK}$ as for real-time rewrite theories. Thus, a *timed path* is written $\pi = t_0 \xrightarrow{r_0} t_1 \xrightarrow{r_1} \ldots$, we define $d^\pi_m = \sum_{i=0}^{m-1} r_i$, $t^\pi_m = t_m$ and $r^\pi_m = r_m$, and the set of all timed fair paths originating in state t is denoted by $\mathit{tfPaths}_{\mathcal{TK}}(t)$.

The semantics of TCTL formulae is defined as follows:

Definition 3. *For timed Kripke structures $\mathcal{TK} = (S, \xrightarrow{\mathcal{T}}, L)$, states $t \in S$, and TCTL formulae φ, the* pointwise satisfaction relation *$\mathcal{TK}, t \models_p \varphi$ is defined inductively as follows:*

$$
\begin{array}{lll}
\mathcal{TK}, t \models_p \mathit{true} & & \textit{always.} \\
\mathcal{TK}, t \models_p p & \textit{iff} & p \in L(t). \\
\mathcal{TK}, t \models_p \neg\varphi_1 & \textit{iff} & \mathcal{TK}, t \not\models_p \varphi_1. \\
\mathcal{TK}, t \models_p \varphi_1 \wedge \varphi_2 & \textit{iff} & \mathcal{TK}, t \models_p \varphi_1 \textit{ and } \mathcal{TK}, t \models_p \varphi_2. \\
\mathcal{TK}, t \models_p E\ \varphi_1\ U_I\ \varphi_2 & \textit{iff} & \textit{there exists } \pi \in \mathit{tfPaths}_{\mathcal{TK}}(t) \textit{ and an index } k \textit{ s.t.} \\
 & & d^\pi_k \in I,\ \mathcal{TK}, t^\pi_k \models_p \varphi_2, \textit{ and} \\
 & & \mathcal{TK}, t^\pi_l \models_p \varphi_1 \textit{ for all } 0 \le l < k. \\
\mathcal{TK}, t \models_p A\ \varphi_1\ U_I\ \varphi_2 & \textit{iff} & \textit{for each } \pi \in \mathit{tfPaths}_{\mathcal{TK}}(t) \textit{ there is an index } k \textit{ s.t.} \\
 & & d^\pi_k \in I,\ \mathcal{TK}, t^\pi_k \models_p \varphi_2, \textit{ and} \\
 & & \mathcal{TK}, t^\pi_l \models_p \varphi_1 \textit{ for all } 0 \le l < k.
\end{array}
$$

For a timed Kripke structure $\mathcal{TK} = (S, \xrightarrow{\mathcal{T}}, L)$, a state $t \in S$ and paths $\pi, \pi' \in \mathit{tfPaths}_{\mathcal{TK}}(t)$ we say that π' is a *simple time refinement* of π if either $\pi = \pi'$ or π' can be obtained from π by replacing a transition $t_k \xrightarrow{r_k} t_{k+1}$, $r_k > 0$, by a sequence $t_k \xrightarrow{r'_k} t \xrightarrow{r''_k} t_{k+1}$ of transitions for some $t \in S$ and time values $r'_k, r''_k > 0$ with $r'_k + r''_k = r_k$. A path π' is a *time refinement* of another path π if π' can be obtained from π by applying a (possibly infinite) number of time refinements. We also say that π is a *time abstraction* of π'.

[5] A transition relation $\xrightarrow{\mathcal{T}}$ can be made *total* by defining $(\xrightarrow{\mathcal{T}})^\bullet = \xrightarrow{\mathcal{T}} \cup \{(s, 0, s) \in S \times \mathcal{T} \times S \mid \neg\exists\, s' \in S, r \in \mathcal{T} \text{ s.t. } (s, r, s') \in \xrightarrow{\mathcal{T}}\}$.

Definition 4. *The* continuous-time satisfaction relation $\mathcal{TK}, t \models_c \varphi$ *is defined as the pointwise one for the first four cases; for the last two cases we have:*

$\mathcal{TK}, t \models_c E\ \varphi_1\ U_I\ \varphi_2$ *iff* *there is a path* $\pi \in \mathit{tfPaths}_{\mathcal{TK}}(t)$ *such that for each time refinement* $\pi' \in \mathit{tfPaths}_{\mathcal{TK}}(t)$ *of* π *there is an index* k *s.t.* $d_k^{\pi'} \in I$, $\mathcal{TK}, t_k^{\pi'} \models_c \varphi_2$, *and* $\mathcal{TK}, t_l^{\pi'} \models_c \varphi_1$ *for all* $0 \leq l < k$.

$\mathcal{TK}, t \models_c A\ \varphi_1\ U_I\ \varphi_2$ *iff* *for each path* $\pi \in \mathit{tfPaths}_{\mathcal{TK}}(t)$ *there is a time refinement* $\pi' \in \mathit{tfPaths}_{\mathcal{TK}}(t)$ *of* π *and an index* k *s.t.* $d_k^{\pi'} \in I$, $\mathcal{TK}, t_k^{\pi'} \models_c \varphi_2$, *and* $\mathcal{TK}, t_l^{\pi'} \models_c \varphi_1$ *for all* $0 \leq l < k$.

3.3 Associating Timed Kripke Structures to Real-Time Rewrite Theories

To each real-time rewrite theory we associate a timed Kripke structure as follows:

Definition 5. *Given a real-time rewrite theory* $\mathcal{R} = (\Sigma, E, R, \phi, \tau)$, *a set of atomic propositions* Π *and a protecting extension* $(\Sigma \cup \Pi, E \cup D) \supseteq (\Sigma, E)$, *we define the associated timed Kripke structure*

$$\mathcal{TK}(\mathcal{R})_\Pi = (\mathbb{T}_{\Sigma/E,\texttt{GlobalSystem}},\ (\xrightarrow{\mathcal{T}}_{\mathcal{R}})^\bullet,\ L_\Pi),$$

where $(\xrightarrow{\mathcal{T}}_{\mathcal{R}})^\bullet \subseteq \mathbb{T}_{\Sigma/E,\texttt{GlobalSystem}} \times \mathbb{T}_{\Sigma/E,\phi(\mathit{Time})} \times \mathbb{T}_{\Sigma/E,\texttt{GlobalSystem}}$ *contains all transitions of the kind* $t \xrightarrow{r} t'$ *which are also* one-step rewrites *in* $\mathcal{R}$ *and all transitions of the kind* $t \xrightarrow{0} t$ *for all those states* t *that cannot be further rewritten in* $\mathcal{R}$, *and for* $L_\Pi : \mathbb{T}_{\Sigma/E,\texttt{GlobalSystem}} \to \mathcal{P}(\Pi)$ *we have that* $p \in L_\Pi(t)$ *if and only if* $E \cup D \vdash (t\ \texttt{|=}\ p) = \texttt{true}$.

We use this transformation to define $\mathcal{R}, L_\Pi, t_0 \models_c \varphi$ as $\mathcal{TK}(\mathcal{R})_\Pi, t_0 \models_c \varphi$, and similarly for the pointwise semantics. The model checking problems $\mathcal{TK}(\mathcal{R})_\Pi, t_0 \models_p \varphi$ and $\mathcal{TK}(\mathcal{R})_\Pi, t_0 \models_c \varphi$ are decidable if

- the equational specification in $\mathcal{R}$ is *Church-Rosser* and *terminating*,
- the set of states reachable from t_0 in the rewrite theory $\mathcal{R}$ is *finite*, and
- given a pair of reachable states t and t', the number of one-step rewrites of the kind $t \xrightarrow{r} t'$ in $\mathcal{R}$ is *finite*.

As mentioned above, real-time rewrite theories generally contain a time-nondeterministic tick rule, but since Real-Time Maude executes such theories by applying a *time sampling strategy* σ, our model checker does not analyze $\mathcal{R}$ but the executable theory $\mathcal{R}^\sigma$ in which the time sampling strategy transformation has been applied. Thus, we associate a timed Kripke structure not to $\mathcal{R}$, but to $\mathcal{R}^\sigma$, and hence the third requirement is satisfied by all but the most esoteric cases; indeed, the tick rules in all Real-Time Maude applications we have seen are deterministic, in the sense that there is at most one one-step tick rewrite $t \xrightarrow{r} t'$ from any state, when the time sampling strategy is taken into account.

We denote by $\mathcal{TK}(\mathcal{R}, t_0)_\Pi$ the timed Kripke structure associated to $\mathcal{R}$ which is restricted to states reachable from t_0, and for states t reachable from t_0 we write $\mathcal{R}, L_\Pi, t \models \varphi$ for $\mathcal{TK}(\mathcal{R}, t_0)_\Pi, t \models \varphi$.

4 Sound and Complete TCTL Model Checking for Real-Time Maude

As mentioned above, at least for dense time domains, Real-Time Maude only analyzes a *subset* of all possible behaviors in a real-time rewrite theory with time-nondeterministic tick rules, namely, those behaviors obtained by applying the tick rules according to a selected time sampling strategy. The paper [26] specifies some conditions on a real-time rewrite theory $\mathcal{R}$ and on the atomic propositions that ensure that model checking $\mathcal{R}^{maxDef(r)}$, i.e., using the maximal time sampling strategy, is indeed a sound and complete model checking procedure to check whether all behaviors in the original model $\mathcal{R}$ satisfy an LTL formula without the next operator.

For example, if no application of a tick rule changes the valuation of the atomic propositions in a formula and instantaneous rewrite rules can only be applied after *maximal* tick steps or after applying an instantaneous rule, then model checking $\mathcal{R}^{maxDef(r)}$ gives a sound and complete model checking procedure for $\mathcal{R}$.[6] This result yields a feasible sound and complete model checking procedure for many useful (dense-time) systems, such as "discrete-event" systems, that includes many systems that cannot be modeled as, e.g., timed automata.

As explained in the introduction, this completeness result does not carry over to *timed* temporal logic properties. To keep the exposition simple, in the following we focus on dense time, since we can achieve sound and complete model checking for discrete time by exploring all possible tick steps in the pointwise semantics, and by advancing time by the smallest possible non-zero duration in the continuous semantics.[7] Furthermore, in this paper we restrict our formalisms to TCTL$_{cb}$.[8]

Our solution to achieve completeness for TCTL$_{cb}$ is to make sure that time progress "stops" at any time point when a time bound in the formula is reached. Since a TCTL formula must be recursively evaluated in each reachable state, we must make sure that from any reachable state we "stop" at a time relative to this state whenever a bound in the formula is reached. This condition is guaranteed if we split any tick step by an amount that divides all possible maximal tick durations and all possible finite non-zero time bounds in the formula. Let $\bar{r}$ be the greatest common divisor of the durations of all maximal tick steps in $\mathcal{R}^{maxDef(r)}$ reachable from the initial state and each finite non-zero time bound in the formula, then we divide each maximal tick step into smaller steps of duration $\frac{\bar{r}}{2}$.

[6] The requirements in [26] are weaker than the above.

[7] This condition is generally stronger than what needed, in fact we could abstract theories with discrete time domain in a way similar to what we do for dense time, i.e. by advancing time by some value smaller than the $\bar{r}$ explained in what follows, if possible.

[8] We are currently working on releasing the restriction to closed bounds. However, our proof for the completeness result cannot be directly extended to TCTL formulae with open bounds.

To formalize this notion, let us first consider the time domain. Real-time rewrite theories are parametric in their time domain; the time domain must only satisfy some abstract properties given in some *functional theory* defined in [25] that defines the time domain abstractly as a commutative monoid $(0, \leq, Time)$ with some additional operators. The following functional theory states that there exists a function `gcd` on the non-zero time values with the expected properties and a commutative function `middle` that states that given two non-equal times x `<` y there is always a third time in between, that is x `< middle(`x`,`y`) <` y.

```
fth GCD-TIME-DOMAIN is including LTIME-INF .
  sort NzTime .   subsort NzTime < Time .   cmb T:Time : NzTime if T:Time =/= 0 .
  op gcd : NzTime NzTime -> NzTime [assoc comm] .
  op _divides_ : NzTime NzTime -> Bool .
  op middle : Time Time -> NzTime [comm] .
  vars T1 T2 T3 : NzTime . vars T T’ : Time .
  eq T1 divides T1 = true .
  ceq T1 divides T2 = false if T2 < T1 .
  eq T1 divides (T1 + T2) =  T1 divides T2 .
  eq gcd(T1, T2) divides T1 = true .
  ceq gcd(T1, T2) >= T3  if T3 divides T1 /\ T3 divides T2 .
  ceq middle(T, T’) > T = true if T’ > T .
  ceq middle(T, T’) < T’ = true if T’ > T .
endfth
```

In the following we assume that all considered time domains satisfy the theory `GCD-TIME-DOMAIN`, and write *gcd* for the interpretation of `gcd`.

The real-time rewrite theory $\mathcal{R}^{gcd(t_0,r,\varphi)}$ is obtained from the tick-robust real-time rewrite theory $\mathcal{R}$, a state t_0 in $\mathcal{R}$, and a TCTL formula φ, by advancing time by the greatest common divisor of all the following values:

- all tick step durations appearing in paths from $\mathit{tfPaths}_{\mathcal{R}^{maxDef(r)}}(t_0)$ and
- all finite non-zero lower and upper bounds of all temporal operators in φ.

Definition 6. *For a real-time rewrite theory $\mathcal{R}$ whose time domain satisfies the theory* `GCD-TIME-DOMAIN`*, a non-zero time value r, a TCTL formula φ and a state t_0 of $\mathcal{R}$ we define*

$$\begin{aligned}
T_1(\mathcal{R}, t_0, r) &= \{r' \in \texttt{NzTime} \mid \exists \pi \in \mathit{tfPaths}_{\mathcal{R}^{maxDef(r)}}(t_0).\ \exists i \geq 0.\ r' = r_i^{\pi}\} \\
T_2(\varphi) &= \{r \in \texttt{NzTime} \mid \textit{there exists a subformula } E\ \varphi_1\ U_I\ \varphi_2 \textit{ or} \\
&\qquad\qquad A\ \varphi_1\ U_I\ \varphi_2 \textit{ of } \varphi \textit{ with } r \textit{ a non-zero finite} \\
&\qquad\qquad \textit{lower or upper bound in } I\} \\
GCD(\mathcal{R}, r, \varphi, t_0) &= gcd(T_1(\mathcal{R}, t_0, r) \cup T_2(\varphi))
\end{aligned}$$

where $gcd(T)$ for a non-empty set T of time points is recursively defined as $gcd(\{r\}) = r$ and $gcd(\{r_1, r_2\} \cup T') = gcd(\{gcd(r_1, r_2)\} \cup T')$.

If $T_1(\mathcal{R}, t_0, r)$ and $T_2(\varphi)$ are finite then the *GCD* value is well-defined and we can define the real-time rewrite theory $\mathcal{R}^{gcd(t_0,r,\varphi)}$ as follows:

Definition 7. *Given a real-time rewrite theory $\mathcal{R}$ whose time domain satisfies the theory* `GCD-TIME-DOMAIN`*, a non-zero time value r, a TCTL formula φ, a state t_0 of $\mathcal{R}$, and assume that $\bar{r} = GCD(\mathcal{R}, t_0, r, \varphi)$ is a defined non-zero time value. Then $\mathcal{R}^{gcd(t_0,r,\varphi)}$ is defined as $\mathcal{R}$ but where each tick rule of the forms (†), (∗), and (§) is replaced by the respective tick rule:*

```
crl [tick] : {t} => {t'} in time x if x := r̄/2 /\ cond [nonexec] .
crl [tick] : {t} => {t'} in time x if x := r̄/2 /\ cond [nonexec] .
crl [tick] : {t} => {t'} in time x if x := r̄/2 [nonexec] .
```

The following lemma states that the evaluation of the formula φ and its subformulae do not change inside tick steps of $\mathcal{R}^{gcd(t_0,r,\varphi)}$.

Lemma 1. *Assume a time-robust real-time rewrite theory $\mathcal{R}$ whose time domain satisfies the theory* `GCD-TIME-DOMAIN`*. Let Π be a set of tick-invariant atomic propositions, and assume a protecting extension of $\mathcal{R}$ defining the atomic propositions in Π and inducing a labeling function L_Π. Let t_0 be a state of $\mathcal{R}$, r a non-zero time value of sort* `Time`*, φ a $TCTL_{cb}$ formula over Π, and assume that $\bar{r} = GCD(\mathcal{R}, t_0, r, \varphi)$ is a defined non-zero time value.*

Then for each subformula φ' of φ, each time-divergent path $\pi \in tfPaths_{\mathcal{R}}(t_0)$ and for all tick step sequences $t_i^\pi \xrightarrow{r_i^\pi} \ldots \xrightarrow{r_{j-1}^\pi} t_j^\pi$ in π satisfying $n \cdot \bar{r} < d_i^\pi < d_j^\pi < (n+1) \cdot \bar{r}$ for some n we have that

$$\mathcal{R}, t_i^\pi \models_c \varphi' \quad \textit{iff} \quad \mathcal{R}, t_j^\pi \models_c \varphi' \ .$$

Proof. The proof by induction on the structure of φ' can be found in our technical report [18].

Based on the above lemma we gain our completeness result:

Theorem 1. *Let $\mathcal{R}$, L_Π, t_0, r, φ and $\bar{r}$ be as in Lemma 1. Then*

$$\mathcal{R}, L_P, t \models_c \varphi \quad \Longleftrightarrow \quad \mathcal{R}^{gcd(t_0,r,\varphi)}, L_P, t \models_p \varphi$$

for all states t reachable in $\mathcal{R}^{gcd(t_0,r,\varphi)}$ from t_0.

Proof. The proof by induction on the structure of φ' can be found in our technical report [18].

5 Implementation

Our model checker makes the natural and reasonable assumption that given a real-time rewrite theory $\mathcal{R}$, and an initial state t_0 on which we would like to check some TCTL formula φ, all behaviors starting from t_0 are *time-diverging* w.r.t. the selected time sampling strategy σ. Notice that this assumption also implies that the transition relation $\xrightarrow{\mathcal{T}}_{\mathcal{R}^\sigma}$, in the respective timed Kripke structure $\mathcal{TK}(\mathcal{R}^\sigma, t_0)_\Pi$, consisting of all the one-step rewrites $t \xrightarrow{r} t'$ in $\mathcal{R}^\sigma$ is *total.*

Also notice that checking that all behaviors from the given t_0 are time-diverging reduces, when the untimed reachable state space from t_0 is finite, to checking that the timed Kripke structure associated to $\mathcal{R}$ has no zero-time loops.

The current implementation of the model checker assumes that time values are either in `NAT-TIME-DOMAIN-WITH-INF` or `POSRAT-TIME-DOMAIN-WITH-INF`, and provides the user with two possible model-checking strategies:

(i) The *default* strategy, in the pointwise semantics, which applies the user-defined time sampling strategy.
(ii) The *gcd* strategy, which requires the maximal time sampling strategy and performs the model checking for the satisfaction problem $\mathcal{R}^{gcd(t_0,r,\varphi)}, t_0 \models_p \varphi$.

Soundness and completeness of the *gcd* strategy might come at the cost of a larger state space due to the application of the *gcd* transformation, e.g. if the maximal time sampling strategy leads to tick step whose durations vary from very "small" ones to very "big" ones. When the *gcd* strategy is impractical, the user can still perform model checking in the pointwise semantics for TCTL$_{cb}$ formulae with the generally faster default strategy, which does not increase the system state space and can still be very useful to discover potential bugs, as illustrated below.

Real-Time Maude, and hence our model checker, is implemented in Maude, making extensive use of Maude's meta-programming capabilities. The model checker first constructs the timed Kripke structure, according to the selected model checking strategy, by collecting all the reachable states and transitions. When the gcd strategy is selected, the timed Kripke structure is refined by "splitting" the transitions into smaller ones of duration equal to the computed greatest common divisor. Then, the satisfaction sets of each subformula are recursively computed. Since the meta-representation of the states can be fairly large[9], performing the rest of the model checking procedure on the generated timed Kripke structure is fairly inefficient. In our current implementation, we assign a unique natural number to each (meta-represented) state in the generated timed Kripke structure, and construct a more compact timed Kripke structure, where all the occurrences of these meta-represented states are replaced by their respective identifiers. We then perform the recursive computation of the satisfaction set of φ, on this compact representation. This optimization led to a large performance improvement and made it feasible to apply our model checker to the case studies in reasonable time, especially in the Ptolemy II case, whereas working directly on meta-represented terms made model checking unfeasible even for simple case studies.

Our implementation of the TCTL model checker is based on the *explicit-state* CTL model checking approach [8] that, starting with the atomic propositions, recursively computes for each subformula of the desired TCTL formula the set of satisfying reachable states. We implemented specific procedures for a basic set of

[9] For example, *each* state in the Maude representation of the Ptolemy II model in Section 6 "contains" the entire Ptolemy II model.

temporal modal operators and we expressed other formulae into this canonical form. The basic set consists of the CTL modal operators $E\ \varphi_1\ U\ \varphi_2$, $E\ G\ \varphi$, the TCTL$_{\leq\geq}$[10] modal operators $E\ \varphi_1\ U_{\sim r}\ \varphi_2$ with $\sim \in \{>, \geq\}$, $E\ \varphi_1\ U_{\sim r}\ \varphi_2$ with $\sim \in \{<, \leq\}$, $A\ \varphi_1\ U_{>0}\ \varphi_2$ and the TCTL$_{cb}$ modal operator $E\ \varphi_1\ U_{[a,b]}\ \varphi_2$. The procedures for CTL modalities follow the standard explicit algorithm [8]. For TCTL$_{\leq\geq}$ modalities, our implementation adapts the TCTL$_{\leq\geq}$ model checking procedure defined in [17] for time-interval structures and to timed Kripke structures with time-diverging paths.

The ease and flexibility of the Maude meta-level allowed us to implement the model checker reasonably quickly and easily. However, the convenience of operating at the meta-level comes at a certain cost in terms of computational efficiency, even with our optimizations. Therefore, the current Real-Time Maude model checker should be regarded as a *working prototype* for a C++ implementation that we plan to implement in the future.

Our model checker is available at `http://folk.uio.no/leprid/TCTL-RTM/` together with the technical report [18], the specifications and analysis commands of the case studies in this paper.

5.1 Using the Model Checker

In Real-Time Maude, the user is provided with two TCTL model checking commands, corresponding respectively to the default and gcd strategy, with syntax

```
(mc-tctl t |= φ .)     and     (mc-tctl-gcd t |= φ .)
```

for t the initial state and φ one of the possible TCTL formula. The syntax of TCTL formulae is fairly intuitive, with syntactic sugar for (untimed) CTL formulae, common abbreviations and boolean connectors such as `AF`, `EF`, `AG`, `EG`, `iff` and `implies`, etc. For example, $E\ \mathit{true}\ U_{\leq r}(\neg\varphi \wedge A\ G\ (E\ F_{[a,b]}\varphi'))$ is written as[11]:

```
E tt U[<= than r](not φ and AG (EF[c a, b c] φ'))
```

We do not support counter-example generation, since, in contrast to linear temporal logics, where counter-examples are just paths, it is generally more complex to generate counter-examples in branching-time temporal logics, where counterexamples are parts of computation trees (see, e.g. [12]). For example, a counter-example to the validity of the formula $E\ F\ p$, for p an atomic proposition, is the entire computation tree (where each state is a $\neg p$-state).

[10] We denote by TCTL$_{\leq\geq}$ the restricted TCTL logic with time constraints on the temporal modalities of the form $\sim r$, where $\sim \in \{<, \leq, \geq, >\}$,

[11] The model checker syntax for TCTL formulae supports also open bounds, e.g. the user could write `[c 0, b o]` for $[0, b)$, which would be internally reduced to `[< than b]`.

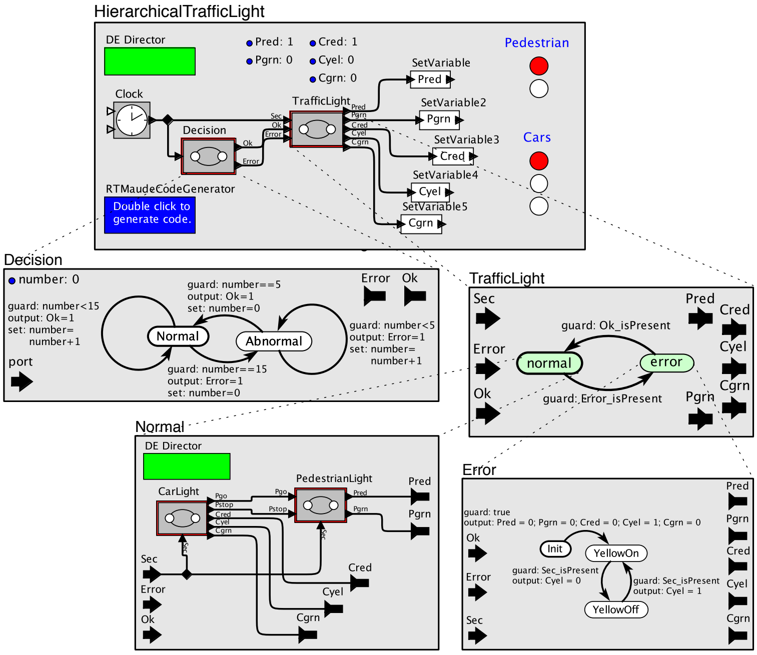


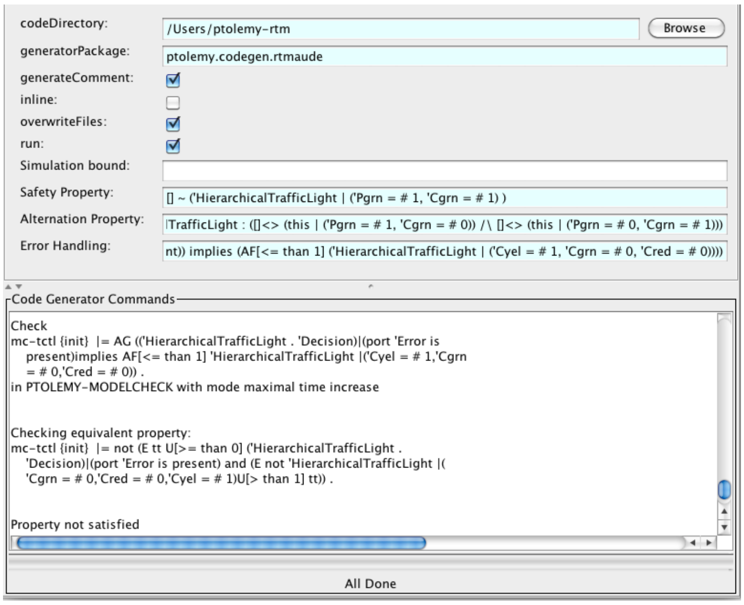
Fig. 1: A hierarchical fault-tolerant traffic light system in Ptolemy II.

6 Case Study: Verifying a Ptolemy II Discrete-Event Model

Real-Time Maude provides a formal analysis tool for a set of modeling languages for embedded systems, including Ptolemy II discrete-event (DE) models that cannot be formalized by, say, timed automata. Ptolemy II [14] is a well-established modeling and simulation tool used in industry that provides a powerful yet intuitive graphical modeling language. Our model checker has been integrated into Ptolemy II by Kyungmin Bae, so that we can now model check TCTL properties of Ptolemy II DE models *from within Ptolemy*[12]. We show the TCTL analysis of a Ptolemy II model of a hierarchical traffic light system, in which our model checker has uncovered a previously unknown flaw. Notice that Ptolemy II DE models satisfy the requirements for having a sound and complete analysis when using the gcd strategy. The analysis has been performed on a 2.4GHz Intel® Core 2 Duo processor with 4 GB of RAM.

Figure 1 shows a hierarchical Ptolemy II model of a fault-tolerant traffic light system at a pedestrian crossing, consisting of one car light and one pedestrian

[12] Real-Time Maude verification commands can be entered into the dialog box that pops up when the blue button in Fig. 1 is clicked.



codeDirectory: /Users/ptolemy-rtm

generatorPackage: ptolemy.codegen.rtmaude

generateComment: ☑

inline: □

overwriteFiles: ☑

run: ☑

Simulation bound:

Safety Property: [] ~ ('HierarchicalTrafficLight | ('Pgrn = # 1, 'Cgrn = # 1))

Alternation Property: TrafficLight : ([]<> (this | ('Pgrn = # 1, 'Cgrn = # 0)) /\ []<> (this | ('Pgrn = # 0, 'Cgrn = # 1)))

Error Handling: nt)) implies (AF[<= than 1] ('HierarchicalTrafficLight | ('Cyel = # 1, 'Cgrn = # 0, 'Cred = # 0))))

Code Generator Commands

```
Check
mc-tctl {init}  |= AG (('HierarchicalTrafficLight . 'Decision)|(port 'Error is
   present)implies AF[<= than 1] 'HierarchicalTrafficLight |('Cyel = # 1,'Cgrn
   = # 0,'Cred = # 0)) .
in PTOLEMY-MODELCHECK with mode maximal time increase


Checking equivalent property:
mc-tctl {init}  |= not (E tt U[>= than 0] ('HierarchicalTrafficLight .
   'Decision)|(port 'Error is present) and (E not 'HierarchicalTrafficLight |(
   'Cgrn = # 0,'Cred = # 0,'Cyel = # 1)U[> than 1] tt)) .


Property not satisfied
```

All Done

Fig. 2: Dialog window for the hierarchical traffic light code generation.

light. Each light is represented by a set of *set variable* actors (`Pred` and `Pgrn` represent the pedestrian light, and `Cred`, `Cyel` and `Cgrn` represent the car light). A light is *on* iff the corresponding variable equals 1. The FSM actor `Decision` "generates" failures and repairs by alternating between staying in location `Normal` for 15 time units and staying in location `Abnormal` for 5 time units. Whenever the model operates in `error` mode, all lights are turned off, except for the yellow light of the car light, which is blinking. We refer to [7] for a thorough explanation of the model.

An important fault tolerance property is that the car light will turn yellow, *and only yellow*, within 1 time unit of a failure. We can model check this bounded response property with the command:

```
Maude> (mc-tctl {init} |=
  AG(('HierarchicalTrafficLight . 'Decision | (port 'Error is present))
    implies AF[<= than 1] ('HierarchicalTrafficLight |
                           ('Cyel = # 1, 'Cgrn = # 0, 'Cred = # 0)))) .)
```

In about 15 secs, the command returns that the property is not satisfied. This model checking uncovered a previously unknown scenario, which shows that, after a failure, the car light may show red or green in addition to blinking yellow. Eleven of the 15 seconds used by the timed CTL model checker were used to generate the timed Kripke structure. Because of the large size of the system states in this case study, it was impossible to run the same analysis before implementing the optimization that mapped each state to an unique identifier. The same property can be model checked with the gcd strategy command mc-tks-gcd in about 22 secs.

Using the gcd strategy we can also determine a “minimal” time interval such that the above bounded response is satisfied in the system. In particular, we discovered that this interval is $[5, 12]$ by trying different values for a and b in the interval-bounded command

```
Maude> (mc-tctl {init} |=
  AG(('HierarchicalTrafficLight . 'Decision | (port 'Error is present))
    implies AF[c a, b c] ('HierarchicalTrafficLight |
                           ('Cyel = # 1, 'Cgrn = # 0, 'Cred = # 0)))) .)
```

Figure 2 shows the dialog window for the Real-Time Maude code generation of the hierarchical traffic light model: after entering the *error handling* property, a simple click on the Generate button will display the result of the model checking command execution in the “Code Generator Commands” box.

7 Conclusions and Future Work

We have described the semantic foundations of our TCTL model checker for Real-Time Maude. Our modeling formalism is more expressive than those of other timed model checkers, allowing us to analyze real-time systems which are beyond the scope of other verification tools. In particular, we have proved soundness and completeness of our model checker for a class of dense-time Real Time Maude specifications that contain many systems outside the scope of other real-time model checkers. Furthermore, the introduced TCTL model checker also provides for free a timed temporal logic model checker for interesting subsets of modeling languages widely used in industry, such as Ptolemy II and the avionics standard AADL.

In future work, the model checker should also provide counter-examples in a user-friendly way, when possible. We want to relax the restriction to closed bounds and even extend our model checker to *time-bounded* TCTL model checking to support the model checking of systems with infinite reachable state space. Finally, the current version of the tool is implemented at the Maude meta-level; for efficiency purposes, it should be implemented in C++ in the Maude engine.

References

1. Aalst, W.M.P.v.d.: Interval timed coloured Petri nets and their analysis. In: Application and Theory of Petri Nets 1993. LNCS, vol. 691, pp. 453–472. Springer (1993)
2. AlTurki, M., Meseguer, J.: Real-time rewriting semantics of Orc. In: Proc. PPDP'07. ACM (2007)
3. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
4. Alur, R., Henzinger, T.: Logics and models of real time: A survey. In: Real Time: Theory in Practice. LNCS, vol. 600. Springer (1992)

5. Alur, R., Courcoubetis, C., Dill, D.: Model-checking in dense real-time. Inf. Comput. 104, 2–34 (May 1993)
6. Bae, K., Ölveczky, P.C., Al-Nayeem, A., Meseguer, J.: Synchronous AADL and its formal analysis in Real-Time Maude. In: ICFEM'11. LNCS, vol. 6991. Springer (2011)
7. Bae, K., Ölveczky, P.C., Feng, T.H., Lee, E.A., Tripakis, S.: Verifying hierarchical Ptolemy II discrete-event models using Real-Time Maude. Science of Computer Programming (2011), to appear, `doi:10.1016/j.scico.2010.10.002`
8. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
9. Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: SFM-RT 2004. LNCS, vol. 3185. Springer (2004)
10. Boucheneb, H., Gardey, G., Roux, O.H.: TCTL Model Checking of Time Petri Nets. J Logic Computation 19(6), 1509–1540 (2009)
11. Bouyer, P.: Model-checking timed temporal logics. ENTCS 231, 323–341 (2009)
12. Clarke, E.M., Grumberg, O., McMillan, K.L., Zhao, X.: Efficient generation of counterexamples and witnesses in symbolic model checking. In: DAC '95 (1995)
13. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: All About Maude, LNCS, vol. 4350. Springer (2007)
14. Eker, J., Janneck, J.W., Lee, E.A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y.: Taming heterogeneity—the Ptolemy approach. Proceedings of the IEEE 91(2), 127–144 (2003)
15. Gardey, G., Lime, D., Magnin, M., Roux, O.H.: Roméo: A tool for analyzing time Petri nets. In: CAV'05. LNCS, vol. 3576. Springer, Edinburgh, Scotland, UK (2005)
16. Katelman, M., Meseguer, J., Hou, J.: Redesign of the LMST wireless sensor protocol through formal modeling and statistical model checking. In: Proc. FMOODS'08. LNCS, vol. 5051. Springer (2008)
17. Laroussinie, F., Markey, N., Schnoebelen, P.: Efficient timed model checking for discrete-time systems. Theor. Comput. Sci. 353, 249–271 (2006)
18. Lepri, D., Ölveczky, P.C., Ábrahám, E.: Timed CTL model checking in Real-Time Maude, `http://folk.uio.no/leprid/TCTL-RTM/tctl-rtm2011.pdf`
19. Lepri, D., Ölveczky, P.C., Ábrahám, E.: Model checking classes of metric LTL properties of object-oriented Real-Time Maude specifications. In: Proc. RTRTS'10. EPTCS, vol. 36, pp. 117–136 (2010)
20. Markey, N., Schnoebelen, P.: TSMV: A Symbolic Model Checker for Quantitative Analysis of Systems. In: QEST. IEEE Computer Society (2004)
21. Morasca, S., Pezzè, M., Trubian, M.: Timed high-level nets. The Journal of Real-Time Systems 3, 165–189 (1991)
22. Ölveczky, P.C.: Semantics, simulation, and formal analysis of modeling languages for embedded systems in Real-Time Maude. In: Formal Modeling: Actors, Open Systems, Biological Systems. LNCS, vol. 7000. Springer (2011)
23. Ölveczky, P.C., Boronat, A., Meseguer, J.: Formal semantics and analysis of behavioral AADL models in Real-Time Maude. In: Proc. FMOODS/FORTE'10. LNCS, vol. 6117. Springer (2010)
24. Ölveczky, P.C., Caccamo, M.: Formal simulation and analysis of the CASH scheduling algorithm in Real-Time Maude. In: Proc. FASE'06. LNCS, vol. 3922. Springer (2006)
25. Ölveczky, P.C., Meseguer, J.: Specification of real-time and hybrid systems in rewriting logic. Theoretical Computer Science 285, 359–405 (2002)
26. Ölveczky, P.C., Meseguer, J.: Abstraction and completeness for Real-Time Maude. Electronic Notes in Theoretical Computer Science 176(4), 5–27 (2007)

27. Ölveczky, P.C., Meseguer, J.: Semantics and pragmatics of Real-Time Maude. Higher-Order and Symbolic Computation 20(1-2), 161–196 (2007)
28. Ölveczky, P.C., Meseguer, J.: The Real-Time Maude tool. In: Proc. TACAS'08. LNCS, vol. 4963. Springer (2008)
29. Ölveczky, P.C., Thorvaldsen, S.: Formal modeling, performance estimation, and model checking of wireless sensor network algorithms in Real-Time Maude. Theoretical Computer Science 410(2-3), 254–280 (2009)
30. Riesco, A., Verdejo, A.: Implementing and analyzing in Maude the enhanced interior gateway routing protocol. Electr. Notes Theor. Comput. Sci. 238(3), 249–266 (2009)
31. Wang, F.: Formal verification of timed systems: A survey and perspective. Proceedings of the IEEE 92(8), 1283–1307 (2004)
32. Wang, F.: REDLIB for the formal verification of embedded systems. In: Proc. ISoLA'06. IEEE (2006)
33. Wirsing, M., Bauer, S.S., Schroeder, A.: Modeling and analyzing adaptive user-centric systems in Real-Time maude. In: Proc. RTRTS'10. EPTCS, vol. 36, pp. 1–25 (2010)
34. Yovine, S.: Kronos: A verification tool for real-time systems. Software Tools for Technology Transfer 1(1–2), 123–133 (1997)