Rewriting Logic Semantics and Symbolic Analysis for Parametric Timed Automata

Jaime Arias  
arias@lipn.univ-paris13.fr  
LIPN, CNRS UMR 7030,  
Université Sorbonne Paris Nord  
France

Kyungmin Bae  
kmbae@postech.ac.kr  
Pohang University of Science and Technology  
Korea

Carlos Olarte  
olarte@lipn.univ-paris13.fr  
LIPN, CNRS UMR 7030,  
Université Sorbonne Paris Nord  
France

Peter Csaba Ölveczky  
peterol@ifi.uio.no  
University of Oslo  
Norway

Laure Petrucci  
petrucci@lipn.univ-paris13.fr  
LIPN, CNRS UMR 7030,  
Université Sorbonne Paris Nord

Fredrik Rømming  
fredrrom@ifi.uio.no  
University of Oslo  
Norway

Abstract

This paper presents a rewriting logic semantics for parametric timed automata (PTAs) and shows that symbolic reachability analysis using Maude-with-SMT is sound and complete for the PTA reachability problem. We then refine standard Maude-with-SMT reachability analysis so that the analysis terminates when the symbolic state space of the PTA is finite. We show how we can synthesize parameters with our methods, and compare their performance with Imitator, a state-of-the-art tool for PTAs. The practical contributions are two-fold: providing new analysis methods for PTAs—e.g., allowing more general state properties in queries and supporting reachability analysis combined with user-defined execution strategies—not supported by Imitator, and developing symbolic analysis methods for real-time rewrite theories.

CCS Concepts: • Theory of computation → Timed and hybrid models: Rewrite systems.

Keywords: Timed automata, rewriting logic, symbolic analysis, parameter synthesis

1 Introduction

Many, if not most, safety-critical computer systems, e.g., in robotics, microelectronics circuits, avionics, and automotive and aerospace systems, are time-critical systems whose correctness depends on time and on the correct values of their parameters. Timed automata [2] are a popular formalism for modeling real-time systems, and the timed automaton tool UPPAAL [23] has been applied to a wide range of safety-critical applications, including automotive [30, 35], airborne [19], and fire fighting [53] systems.

Parametric timed automata (PTA) [3] extend timed automata to the case where the values of some system parameters are unknown. The formal modeling, parameter synthesis, and analysis of PTAs are supported by the state-of-the-art Imitator tool [5], which has been applied to a number of systems, including protocols [27–29], an asynchronous circuit commercialized by ST-Microelectronics [18], and a distributed architecture for the flight control system of spacecraft designed at ASTRIUM Space Transportation [25].

Timed automata are nevertheless a somewhat restricted formalism—to ensure that key properties are decidable—that does not support well features like unbounded data structures, user-defined data types, different forms of communications, dynamic object creation and deletion, and so on.

Rewriting logic [37] and its associated tool Maude [22] are on the other side of the expressiveness spectrum, and support the above features. The Real-Time Maude tool [48, 49] extends Maude to real-time systems and has been used to analyze a wide range of systems where the above features are needed. Such applications include state-of-the-art 50-page multicast and IETF protocols [33, 50], scheduling protocols with unbounded queues [45], state-of-the-art wireless sensor network protocols [51], MANET protocols [36], turning control algorithms for aircraft [11], human multitasking [17], large cloud-based transaction systems [16, 26], and so on (see [43] for a dated overview). In particular, thanks to its expressiveness, Real-Time Maude has been applied as a semantic framework and formal analysis backend in which a number of modeling languages, such as (subsets
of) Ptolemy II [13], AADL [44], a language developed at DOCOMO Labs [1], and others have been given a formal semantics and formal analysis capabilities [42].

However, Real-Time Maude only supports concrete execution of real-time systems, where time advances by a concrete value in each step. Many behaviors (those where time advances by other values) are therefore not analyzed in dense-time systems, and hence Real-Time Maude analysis is in general unsound [47]. One way to provide sound and complete formal analysis for real-time systems in (Real-Time) Maude is to perform symbolic execution that has recently been enabled by combining rewriting logic with SMT solving [52], and implemented in the Maude-SE tool [54].

In this paper we define a rewriting logic semantics for PTAs by mapping a PTA $\mathcal{A}$ into a rewriting logic theory $[[\mathcal{A}]]$, and showing that $\mathcal{A}$ and $[[\mathcal{A}]]$ are bisimilar (Section 3). More importantly, we show in Section 4 that symbolic execution with Maude-with-SMT gives us sound and complete reachability analysis methods for $[[\mathcal{A}]]$. However, straightforward Maude-with-SMT execution of $[[\mathcal{A}]]$ generates a new SMT variable whenever time advances, which leads to nontermination when the desired states are unreachable. We therefore show in Section 4 that “folding” symbolic states solves this problem, and implement a reachability analysis command for Maude-with-SMT that terminates whenever the parametric zone graph of the PTA is finite.

Section 5 shows how we can synthesize parameters that guarantee that a desired reachability property is satisfied. We also show how we can combine our methods and Maude’s strategy language to perform symbolic reachability analysis when the PTA execution follows a user-defined strategy.

In Section 6 we compare the performance of Imitator, “standard” Maude-with-SMT reachability analysis, and our new reachability command on a number of PTAs taken from the PTA benchmark library [7].

The contributions of this work are the following. First, it provides new analysis methods for PTA that are not provided by Imitator. For example, we can analyze PTAs that behave according to a certain execution strategy, defined using Maude’s strategy language, and we illustrate in this paper that this can be useful for PTAs. Our approach also allows us to tackle properties that Imitator cannot handle, by permitting state properties not only on the locations but also on the values of clocks and parameters. Second, Maude provides meta-programming facilities that allow us to quickly implement and prototype new analysis methods for PTAs, instead of having to hardcode them in a tool. Third, Maude provides full (explicit-state) LTL and LTLR model checking, and Real-Time Maude provides timed CTL model checking [32]; when these methods are extended to the symbolic case, we would get full (timed and untimed) temporal logic checking for PTA, which is not provided by either Imitator or Urpaal. Fourth, and maybe most important, this work is the first step investigating how real-time systems can be efficiently symbolically analyzed using Maude-with-SMT, with the goal of providing sound and complete symbolic analysis methods for (Real-Time) Maude. This would also automatically equip a number of modeling languages with such sound and complete formal analysis methods.

The proofs of all results can be found in the technical report [9]. The companion repository of this paper [8] contains the rewrite theories, examples, and benchmarks presented here, as well as a tool for translating Imitator files into Maude.

## 2 Preliminaries

This section gives background to bisimulations [20], parametric timed automata [3], rewriting logic [37], rewriting modulo SMT [52], and Maude [22] and its strategy language [21].

### Parametric Timed Automata (PTA)

Let $X$ be a set of real-valued clocks (e.g. $x, y$) and $P$ a set of rational-valued parameters (e.g. $p, q$). A linear term over parameters (plt) is an expression $(\Sigma_i \alpha_i p_i + \beta)$, where $p_i \in P$ and $\alpha_i, \beta \in \mathbb{Q}$. A (diagonal) inequality has the form $x_i - x_j \leq \text{plt}$, where $x_i, x_j \in X \cup \{0\}$, $\leq \in \{<, \leq, =, \geq, >\}$. Examples are $x - y \leq 2p + q$, $x > q - 1$ and $2 \leq p$. A (convex) constraint (or zone) is a conjunction of inequalities. We write $C$ for the set of zones.

A parametric timed automaton (PTA) $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, L_0, X, P, I, E)$, where $\Sigma$ is a finite set of actions, $L$ is a finite set of locations, $L_0 \in L$ is the initial location, $X$ is a set of clocks, and $P$ is a set of parameters. $I : L \rightarrow C$ denotes an invariant for each location and $E$ is a set of transitions of the form $(\ell, q, \sigma, R)$, with source $\ell \in L$, target $\ell' \in L$, guard $g \in C$, action $\sigma \in \Sigma$, and clock reset $R \subseteq X$

A clock valuation is a function $v : P \rightarrow \mathbb{Q}_{\geq 0}$ and a clock valuation is a function $w : X \rightarrow \mathbb{R}_{\geq 0}$. For $d \in \mathbb{R}_{\geq 0}$ the clock valuation $w + d$ is defined $(w + d)(x) := w(x) + d$. For a clock reset $R \subseteq X$ the clock valuation $w[R]$ is defined $w[R](x) := 0$ if $x \in R$ and $w(x)$ otherwise. We extend parameter valuations to linear terms. We write $\delta$ for the clock valuation s.t. $\forall x \in X : \delta(x) = 0$. We extend parameter valuations to linear terms. We write $\delta$, $\nu, \nu \models Z$ if $\nu \models \phi$ for each inequality $\leq$ in the zone $Z$.

Given a parameter valuation $\nu$, we write $\nu(\mathcal{A})$ for the timed automaton (TA) obtained by replacing each parameter $p$ in invariants and guards by $\nu(p)$. The concrete semantics of a PTA $\mathcal{A}$ is derived from that of the TA $\nu(\mathcal{A})$, and is defined as a timed transition system with states $(\ell, w)$, initial state $(\ell_0, 0)$ (we assume that $0 \models I(\ell_0)$), and transitions
Figure 1. A coffee machine (CM) modeled as a PTA.

→ = d; e, where continuous time delay (d) and discrete transitions (e) are defined as follows:

• If d ∈ ℝ ≥ 0 and w + d ⊨ I(ℓ), then (ℓ, w) d (ℓ, w + d).
• If e = (ℓ, g, σ, R, ℓ′) ∈ E and w ⊨ g and w[R] ⊨ I(ℓ′) then (ℓ, w) e (ℓ′, w[R]).

Example 2.1. The PTA in Fig. 1—with 4 locations, 2 clocks (x1 and x2) and 3 parameters (p1, p2, p3)—models a simple coffee machine. Invariants are displayed inside dotted boxes.

The machine can initially be idle for an arbitrarily long time. Then, whenever the user presses the button bStart, the PTA enters location add sugar, resetting both clocks. The machine can remain in this location as long as the invariant (x2 ≤ p2) is satisfied; then, the user can add a dose of sugar by pressing the button bSugar, provided the guard (x1 ≥ p1) is satisfied, which resets x1. Then, p2 time units after the bStart button was last pushed, a cup is delivered (action cup), and the coffee is being prepared; p3 time units after the last bStart button push, the coffee (action coffee) is delivered. After 10 time units, the machine returns to the idle mode—unless a user again requests coffee by pushing bStart.

The parametric zone graph (PZG) provides a symbolic semantics for a PTA. A single PZG treats all parameter valuations symbolically. Although the PZG avoids the uncountably infinite transition system, it may be (countably) infinite. We define the following operations on zones:

Time elapse: Z′ def = (v, w + d) | d ∈ ℝ ≥ 0 ∧ v, w ⊨ Z

Clock reset: Z[R] def = (v, w[R]) | v, w ⊨ Z

The PZG is a transition system where each abstract state consists of a location and a non-empty zone. The PZG of \( \mathcal{A} = (\Sigma, L, \ell_0, X, P, I, R) \) is \((S, \Sigma_0, \Rightarrow)\), with \( S \subseteq L \times C \), initial state \( \Sigma_0 = (\ell_0, (\backslash x \in X x = 0)' \land I(\ell_0)) \). A transition step \((\ell, Z) \Rightarrow (\ell', Z')\) exists for some \((\ell, g, \sigma, R, \ell') \in E\) we have \( Z' = ((Z \land g[R]) \land I(\ell'))' \land I(\ell') \neq \emptyset \). We write \( \Rightarrow^* \) for the reflexive-transitive closure of \( \Rightarrow \).

Example 2.2. Figure 2 presents the beginning of the parametric zone graph of the coffee machine in Example 2.1.

Rewrite Theories. An order-sorted signature \( \Sigma \) is a triple \((S, \leq, F)\) with \( S \) a set of sorts, \( S \leq \) a partial order on \( S \), and \( F \) a set of function symbol declarations \( f : s_1 \times \cdots \times s_n \rightarrow s \), for \( n \geq 0 \). We denote by \( T_{\Sigma,s} \) the set of ground (i.e. not containing variables) \( \Sigma \)-terms of sort \( s \), and by \( T_{\Sigma}(X) \), the set of \( \Sigma \)-terms of sort \( s \) over a set \( X \) of sorted variables. \( T_{\Sigma}(X) \) and \( T_E \) denote all terms and ground terms, respectively.

A substitution \( \theta : X \rightarrow T_{\Sigma}(X) \) maps each variable to a term of the same sort. \( t \theta \) denotes the term obtained by simultaneously replacing each variable \( x \) in \( t \) with \( \theta(x) \).

An order-sorted equational theory is a pair \( \mathcal{E} = (\Sigma, E) \), where \( \Sigma \) is an order-sorted signature and \( E \) is a set of (conditional) equations of the form \( t = t' \) if \( \psi \), where \( t, t' \in T_{\Sigma}(X) \) for some sort \( s \in \Sigma \) and \( \psi \) is a conjunction of equations. We write \( u \equiv_E u' \) if \( \psi \) is a conjunction of equations and rewrites.

A rewrite theory \([37]\) is a tuple \( \mathcal{R} = (\Sigma, E, L, R) \), where \( \Sigma, E \) is an order-sorted signature, \( L \) is a set of labels, and \( R \) is a set of labeled (conditional) rewrite rules of the form \( l : q \rightarrow r \) if \( \psi \), where \( l \in L, q, r \in T_{\Sigma}(X) \) for some sort \( s \in \Sigma \), and \( \psi \) is a conjunction of equations and rewrites.

Rewriting with SMT. A built-in theory \( \mathcal{E}_0 \) of \((\Sigma, E)\) is a first-order theory with a signature \( \Sigma_0 \subseteq \Sigma \), where each sort \( s \in \Sigma_0 \) is minimal in \( \Sigma \) and for each operator \( f : w \rightarrow s \) in \( \Sigma \setminus \Sigma_0, s \notin \Sigma_0 \) and \( \mathsf{f} \) has no other sort-overloaded typing in \( \Sigma_0 \). Satisfiability of a constraint in \( \mathcal{E}_0 \) is assumed to be decidable using the SMT theory \( \mathcal{E}_0 \) which is consistent with \((\Sigma, E)\): for \( t_1, t_2 \in T_{\Sigma_0}, \) if \( t_1 \equiv_E t_2 \), then \( T_{\Sigma_0} \models t_1 = t_2 \).[52]

A constrained term is a pair \( \phi || t \) of a constraint \( \phi \) in \( \mathcal{E}_0 \) and a term \( t \in T_{\Sigma_0}(X_0) \) over variables \( X_0 \subseteq X \) of the built-in sorts in \( \mathcal{E}_0 \).[14, 52]. A constrained term \( \phi || t \) symbolically represents all instances of the pattern \( t \) such that \( \phi \) holds:
\[ (\phi \parallel t) = \{ t' \mid t' = t \parallel \theta \text{ and } \theta _{E_0} = \phi \theta \} \]

A symbolic rewrite on constructed terms symbolically represents a (possibly infinite) set of system transitions. Let \( R \) be a topmost theory such that for each rule \( l : q \rightarrow r \text{ if } \psi \), extra variables not occurring in the left-hand side \( q \) are in \( X_0 \), and \( \psi \) is a constraint in a built-in theory \( E_0 \). Then, a one-step symbolic rewrite \( \phi \parallel t \mapsto \phi' \parallel t' \) holds iff there exist a rule \( l : q \rightarrow r \text{ if } \psi \) and a substitution \( \theta : X \rightarrow \theta _{E_0}(X_0) \) such that

1. \( t = _E \theta \theta \),
2. \( t' = r \theta \),
3. \( \theta _{E_0} = (\phi \land \psi \theta) \Rightarrow \phi' \),
4. and \( \phi' \) is \( \theta _{E_0} \)-satisfiable. We denote by \( \mapsto_R \) the reflexive-transitive closure of \( \mapsto_R \).

If \( \phi_l \parallel t \mapsto * \phi_u \parallel u \) is a symbolic rewrite, then there exists a "concrete" rewrite \( t' \rightarrow u' \) with \( t' \in [\phi_l \parallel t] \) and \( u' \in [\phi_u \parallel u] \). Conversely, for any concrete rewrite \( t' \rightarrow u' \) with \( t' \in [\phi_l \parallel t] \), there exists a symbolic rewrite \( \phi_l \parallel t \mapsto * \phi_u \parallel u \) with \( u' \in [\phi_u \parallel u] \).

**Maude.** Maude [22] is a language and tool supporting the specification and analysis of rewrite theories. We use Maude to specify rewrite theories, and summarize its syntax below:

```
mod M is ... endm    --- Rewrite theory M
pr R             --- Importing a theory R
sorts S ... Sk     --- Declaration of sorts S1,..., Sk
subsort SI < Sk    --- Subsort relation
vars X1 ... Xn : S  --- Logical variables of sort S
op f : SI ... Sn -> S  --- Operator SI x ... x Sn -> S
op c : -> T        --- Constant c of sort T
ceq t = t' if c     --- Conditional equation
cri [1] : q => r if c. --- Conditional rewrite rule
```

Maude provides a number of analysis methods, including computing the normal form ("value") of an expression (command `red`), simulation by rewriting, and explicit-state reachability analysis and LTL model checking. The command `smt-search [n, m] : t => t'` such that \( \Phi \).

symbolically searches for \( n \) states, reachable from \( t \in T_S(X_0) \) within \( m \) steps, that match the pattern \( t' \in T_S(X) \) and satisfy the constraint \( \Phi \in E_0 \). More precisely, it searches for a constrained term \( \phi_u \parallel u \) such that true \( \parallel t \mapsto * \phi_u \parallel u \) and for some \( \theta : X \rightarrow T_S(X), u = t \theta \) and \( \theta _{E_0} = \phi_u \Rightarrow \phi \theta \). The parameters \( m \) and \( n \) are optional.

Maude provides built-in sorts Boolean, Integer, and Real for the SMT theories of Booleans, integers, and reals. Rational constants of sort Real are written \( n/m \) (e.g., \( 0/1 \)).

Maude supports meta-programming, where a Maude module \( M \) (resp. a term \( t \)) can be (meta-)represented as a Maude term \( \overline{M} \) (resp. as a Maude term \( \overline{t} \) of sort Term) in Maude’s META-LEVEL module. Sophisticated analysis commands and model/module transformations can then be easily defined as ordinary Maude functions on such (meta-)terms. For this purpose, Maude provides built-in functions such as metaReduce, metaRewrite, metaMatch, and metaCheck.

Maude-SE [54] extends Maude with additional functionality for rewriting modulo SMT, including witness generation for `smt-search`. It uses two theory transformations to implement symbolic rewriting [52]. In essence, a rewrite rule

\[ l : q \rightarrow r \text{ if } \psi \text{ is transformed into a constrained-term rule} \]

\[ l : \Phi \parallel q \rightarrow (\Phi \land \psi) \parallel r \text{ if } \text{smtCheck}(\Phi \land \psi) \]

where \( \Phi \) is a Boolean variable, and `smtCheck` invokes the underlying SMT solver to check the satisfiability of an SMT condition. This rule is executable if the extra SMT variables in \( (\text{var}(r) \cup \text{var}(\psi)) \setminus \text{var}(q) \) are considered constants.

**Strategy Language.** Maude’s strategy language [21] allows us to define strategies for applying the rewrite rules. The command `srew t using str` rewrites the term \( t \) according to the strategy `str` and returns all its results. Basic strategies include the application of a rule \( l \) once anywhere in the term (strategy `l`), and top(`l`) for rewriting at the top of term using rule `l`; all denotes all rules, `idle` (identity), fail (empty set), and match `P` s.t. `C`, which checks whether the current term matches the pattern `P` subject to the (optional) condition `C`. Compound strategies can then be defined using constructs such as: concatenation (`\alpha ; \beta`), disjunction (`\alpha | \beta`), iteration (`\alpha^*`), and `\alpha \text{ or else } \beta`, which executes `\beta` if `\alpha` fails.

### 3 A Rewriting Logic Semantics for PTA

This section presents a rewriting logic semantics for PTA by defining in Section 3.1 a theory transformation \( [[ \_ ]] \) mapping a PTA \( \mathcal{A} \) into a rewrite theory \( [[ \mathcal{A} ]] \). Section 3.2 provides a bisimulation result relating the concrete semantics of \( \mathcal{A} \) and a rewrite relation induced by \( [[ \mathcal{A} ]] \).

#### 3.1 The PTA to Rewrite Theory Transformation

We fix \( \mathcal{A} \) to be the PTA \( (\Sigma, L, l_0, X, P, l, E) \) with \( n = |X| \) clocks \((x_1, \ldots, x_n)\), \( m = |P| \) parameters \((p_1, \ldots, p_m)\), and \( k = |L| \) locations \( \{l_0, \ldots, l_k\} \). The idea is to represent a concrete state \((\ell, w)\) of the PTA \( \mathcal{A} \) as a Maude term

\[ [\ell : w(x_1); \ldots; w(x_n)] < P_1; \ldots; P_m > \]

where the \( P_i \) are variables. A state \((\ell, w)\) in the TA \( v(\mathcal{A}) \) (i.e., the PTA \( \mathcal{A} \) whose parameters are instantiated with the parameter valuation \( v \)) then has the form

\[ [\ell : w(x_1); \ldots; w(x_n)] < v(p_1); \ldots; v(p_m) > \]

To avoid consecutive steps that advance time, which can be combined into one such step, we use "delayed" states

\[ < \ell : w(x_1); \ldots; w(x_n) > < P_1; \ldots; P_m > \]

where time cannot advance any further.

Each transition in \( \mathcal{A} \) is modeled by a rewrite rule. For example, in the coffee machine in Fig. 1, the transition `bSugar` is modeled by the rewrite rule

```
crl [add_sugar-bSugar] : < add_sugar : X1 ; X2 > < P1 ; P2 ; P3 > =>
[ add_sugar : 0/1 ; X2 ] < P1 ; P2 ; P3 >
if (X1 => P1 and X2 <= P3) = true .
```

Furthermore, for each location \( l \in L \), we add a "tick" rewrite rule that advances the time in all clocks, modeling "idling" in that location. The tick rule for, e.g., location `add_sugar` is
We define two functions $I$ where $I_{\geq}$ ("delayed state"), as follows:

\[
I_{\geq}(\text{State}) = \text{Non-delayed state}
\]

Since time can advance by any amount $T$ where $x_2 + T \leq P_2$, this time increase is modeled by introducing a new variable $T$ in the right-hand side of the rule, thus making this rule not directly executable in Maude ([nonexec], see Section 4).

The rewrite theory $[\mathcal{A}]$ defines the sorts Location and State, with subsorts NState ("Non-delayed state") and DState ("delayed state"), as follows:

\[
pr \ \text{REAL} . \quad \text{--- SMT rational/real numbers}
\]
\[
\text{sorts State NState DState Location} . \quad \text{--- Sorts for states}
\]
\[
\text{subsorts NState DState < State} . \quad \text{--- Constants for locations}
\]
\[
\text{ops \_;_} : \text{Location Real} . \quad \text{--- Time elapse}
\]
\[
\text{vars X1 X2 P1 P2 P3 T} : \text{Real} . \quad \text{--- Clock valuations}
\]
\[
\text{vars X1 X2 P1 P2 P3 T} : \text{Real} . \quad \text{--- Clock valuations}
\]
\[
\text{var T} : \text{Real} . \quad \text{--- Time elapse}
\]

We define two functions $[\_]_b$ and $[\_]_e$ for translating parameter guards and invariants to terms, where

\[
[\text{true}]_b = \text{true} \quad [b_1 \land b_2]_b = ([b_1]_b \land [b_2]_b)
\]

and for each inequality relation in $\{\geq, \leq, =, >, \}<$, we have, e.g.:

\[
[e_1 \land e_2]_b = ([e_1]_b \land [e_2]_b) \quad [e_1 \leq e_2]_b = ([e_1]_b \leq [e_2]_b)
\]

For arithmetic expressions, we define:

\[
[e_1 + e_2]_e = ([e_1]_e + [e_2]_e) \quad \langle x^R \rangle_e = Xi \text{ if } x_i \notin R
\]
\[
[e_1 - e_2]_e = ([e_1]_e - [e_2]_e) \quad \langle x^L \rangle_e = Xi
\]
\[
[e_1 \times e_2]_e = ([e_1]_e \times [e_2]_e) \quad \langle x^D \rangle_e = Xi + T
\]
\[
[e_1/p]_e = p/0 \text{ if } p, q \in \mathbb{N} \quad \langle p/q \rangle_e = Pi
\]
\[
\langle x^I \rangle_e = 0/1 \text{ if } x_i \in R
\]

$[\_]_e$ maps each transition $(t, g, \sigma, R, t') \in E$, to the following conditional rewrite rule $t$-$\sigma$:

\[
\begin{align*}
\text{cr1}[t-\sigma] : & \quad [t : X1 ; \ldots ; Xn ] < P1 ; \ldots ; Pm > \\
& \Rightarrow [t' : X1 ; \ldots ; Xn ] < P1 ; \ldots ; Pm > \\
& \text{if } \langle \langle p \land l(t') \rangle_x^R \rangle = \text{true} .
\end{align*}
\]

where $l(t')\langle x^R \rangle$ denotes substituting $x_i^R$ for $x_i$ in the expression $l(t')$ for each $i$. Furthermore, for each $\ell \in L$, $[\_]_e$ adds a conditional rewrite rule $t$-tick:

\[
\begin{align*}
\text{cr1}[t-tick] : & \quad [t : X1 ; \ldots ; Xn ] < P1 ; \ldots ; Pm > \\
& \Rightarrow [t' : X1 + T ; \ldots ; Xn + T ] < P1 ; \ldots ; Pm > \\
& \text{if } \langle |l(t)\langle x^R \rangle| \rangle = \text{true} .
\end{align*}
\]

Example 3.1. $[\_]_e$ transforms the PTA-COFFEE in Fig. 1 to the following rewrite theory PTA-COFFEE (the complete set of rules can be found in [9]):

![Diagram of PTA-COFFEE](image-url)
4 Symbolic Reachability Analysis

The theory $[[\mathcal{A}]]$ is not directly executable in Maude, since the

the automaton is in state

we prove in Section 4.2 that symbolic executions in $[[\mathcal{A}]]$

The SMT variables $[[\mathcal{A}]]$

and $T'$

$\phi$

expressions as illustrated in the following example.

Maudé-SE allows us to solve symbolic reachability prob-

is a fresh variable, $\#3-T$ is the sum of the delays accumulated in locations $add-sugar$, $preparing-coffee$ and $done$, and therefore $X_2' \geq P_3$.

4.2 Soundness and Completeness

This section shows that the transition system induced by the

symbolic rewrite relation $\leadsto[[\mathcal{A}]]$ is bisimilar to the

problem is that "standard" symbolic reachability analysis

correspond to transitions in the PZG of $\mathcal{A}$. A significant

state in the PZG of $\mathcal{A}$ is not directly executable in Maude, since

in their right-hand

expression in the corresponding constrained term in $[[\mathcal{A}]]$

is satisfiable (and hence, reachable via $\leadsto[[\mathcal{A}]]$).

Lemma 4.2. For any zone $Z$, $Z \neq \emptyset$ iff $[[\mathcal{A}]]_b$ is satisfiable.

Next we define operations on constrained terms corre-

those on zones. We use $\{t : e_1; \ldots; e_n\}$ to denote

expression $0/1$ if $x_i \in R$ and $e_i$ if $x_i \notin R$. Let $U = \phi \llbracket \{t : e_1; \ldots; e_n\} < P_1; \ldots; P_m \rrbracket$. We define the following operations on $\mathcal{T}_Z - \text{terms}$:

- $\text{Reset: } U[R]$ is a boolean expression such that

- $\text{Time elapse: } U^T$ is a fresh variable, not occurring in $\phi$.

- $\text{Conjunction: } U \land G$ is a boolean expression such that $\text{var}(G) \subseteq \text{var}(U)$.

- $\text{Instantiation: } U\{v, w\}$ is a fresh variable, not occurring in $\phi$.

uses a breadth-first search strategy to answer the following reachability question: are there values for the clocks and

parameters such that the location done can be reached from the

location idle? Note that the clocks and the parameters are not given specific values, not even in the initial state. The

symbolic term to the left of the arrow $\Rightarrow$, together with the

constraint in the “such that” section of the query, specify

initial states where the values of the clocks are equal ($X_1 \equiv X_2$) but unknown, and where parameters and clocks

are all non-negative numbers. The first answer to this query includes

the satisfiable constraint (syntax where) accumulated along the path from idle to done:

Solution 1

state: < done : #3-T ; #1-T + #2-T + #3-T > <P1 ; P2 ; P3>

where $X_1 \equiv X_2$ and $X_1 \equiv 0/1$ and $P_1 \equiv 0/1$ and $P_2 \equiv 0/1$ and

... and $#1-T:Real + #2-T:Real \equiv P_3$ and

... and $#3-T:Real \equiv 0/1$ and $#1-T:Real + #2-T:Real \equiv P_3$

$\phi$ and $T' \geq 0/1$ || idle : $X_1 ; X_2$ || $P_1 ; P_2 ; P_3 >$ * ||< idle : $X_1 + T' ; X_2 + T' >$ < $P_1 ; P_2 ; P_3 >$

The SMT variables $X_i$ (resp. $P_i$) represent the values of the

clocks (resp. parameters). The variable $T'$ is a fresh variable,

sort Real, created in the rewrite. This symbolic rewrite captures all the infinitely many delays that can take place

when the automaton is in state idle.

Example 4.1. In the module PTA-COFFEE, the command

smt-search $[[\mathcal{A}]]$ can be symbolically executed using Maude-with-SMT, and

we prove in Section 4.2 that symbolic executions in $[[\mathcal{A}]]$

correspond to transitions in the PZG of $\mathcal{A}$. A significant

problem is that “standard” symbolic reachability analysis

using Maude-SE adds a new SMT variable to the symbolic

state in each tick step, which leads to nontermination if the

desired states are not reachable. To solve this problem, we use “folding” [39] to ignore a new symbolic state when it is

subsumed by a previously encountered one. In Section 4.3 we
define and implement in Maude such symbolic reachability

analysis with folding. We prove that our procedure terminates when the PZG is finite, and hence obtain a decision

procedure for reachability when the number of states in the

PZG of the automaton is finite.

4.1 Symbolic Reachability Analysis

Although the tick rules are not directly executable in Maude,
we can symbolically execute a rewriting-modulo-SMT theory

with the symbolic rewrite relation $\Rightarrow$. For example, we have the following symbolic rewrite in our running example:

The terms $\#i-T$ are fresh SMT variables generated when the

tick rules are applied. The result includes information about

the values of the clocks in location done: the value of the

first clock ($X_1'$) is $#3-T \leq 10/1$, while the second clock ($X_2'$)
is the sum of the delays accumulated in locations $add-sugar$, $preparing-coffee$ and $done$, and therefore $X_2' \geq P_3$.

expression $0/1$ if $x_i \in R$ and $e_i$ if $x_i \notin R$. Let $U = \phi \llbracket \{t : e_1; \ldots; e_n\} < P_1; \ldots; P_m \rrbracket$. We define the following operations on $\mathcal{T}_Z - \text{terms}$:

- $\text{Reset: } U[R]$ is a boolean expression such that

- $\text{Time elapse: } U^T$ is a fresh variable, not occurring in $\phi$.

- $\text{Conjunction: } U \land G$ is a boolean expression such that $\text{var}(G) \subseteq \text{var}(U)$.

- $\text{Instantiation: } U\{v, w\}$ is a fresh variable, not occurring in $\phi$.

with $w$ (similarly for the parameters). Hence, $U\{v, w\}$ agrees with the values assigned by $v$ and $w$. 

$\phi$ and $T' \geq 0/1$ || idle : $X_1 ; X_2$ || $P_1 ; P_2 ; P_3 >$ * ||< idle : $X_1 + T' ; X_2 + T' >$ < $P_1 ; P_2 ; P_3 >$

such that $(X_1 \equiv X_2$ and $X_1 \equiv 0/1$ and $P_1 \equiv 0/1$

and $P_2 \equiv 0/1$ and $P_3 \equiv 0/1)$ = true .
Definition 4.4 (Relation ∼). Define ∼ ⊆ (L × C) × TₗZ as follows: (ℓ, Z) ∼ U = φ || <ℓ : e₁;...;eₙ,⟩ <P₁;...;Pₘ>, whenever for all v and w, we have (v, w ⊳ Z) iff the boolean expression in U{v, w} is satisfiable.

Intuitively, a state (ℓ, Z) in the PZG of A is related to the symbolic state U whenever the locations are the same and the valuations that belong to the zone Z are consistent with the values making the constraint φ in U true.

The following lemmas show that the operations on zones agree with those in Definition 4.3.

Lemma 4.5 (Reset). Let R ⊆ X, Z ≠ ∅, and assume that (ℓ, Z) ∼ U where U = φ || <ℓ : e₁;...;eₙ,⟩ <P₁;...;Pₘ>. Then, (ℓ, Z[R]) ∼ U[R].

Lemma 4.6 (Time elapse). Let Z ≠ ∅, and assume that (ℓ, Z) ∼ U where U = φ || <ℓ : e₁;...;eₙ,⟩ <P₁;...;Pₘ>. Then, (ℓ, Z′) ∼ U′.

Lemma 4.7 (Conjunction). Let G be a guard or an invariant, Z ≠ ∅, U = φ || <ℓ : e₁;...;eₙ,⟩ <P₁;...;Pₘ> and assume that (ℓ, Z) ∼ U. Then, (ℓ, G ∨ Z) ∼ U ∧ ([G]ᵥ[X₁/e₁]).

Recall that the relation ⇒ on the PZG captures, in one step, a discrete transition followed by a delay transition. Hence, a state (ℓ, Z) is ready to perform a discrete transition leading to (ℓ’, Z’) where Z’ = ((Z ∩ g)[R] ∩ I(ℓ’))’ ∩ I(ℓ’) (if Z’ is not empty). Let ̂→ₗσ[A] be the application of a ℓ-σ rule followed by a tick rule, and let ̂→ₗσ[A] be its reflexive and transitive closure. The following Theorem shows that ⇒ on the PZG is bisimilar to ̂→ₗσ[A] on constrained terms.

Theorem 4.8. Let A be a PTA. ∼ is a bisimulation between the transition systems (C, s₀, ⇒) and (TₗZ; φ₀ || t₀, ̂→ₗσ[A]) where t₀ = <ℓ₀ : T₁;...;Tₙ > <P₁;...;Pₘ> and φ₀ = (P₁ ≥ θ₁ and T ≥ θ₁ and ([I(t₀)]ᵥ[X₁/T])).

4.3 Symbolic Reachability Analysis with Folding

Many PTAs A generate finite PZGs (so reachability analysis should terminate for both positive and negative queries), while the number of symbolic states generated by smt-search [l₀ : 0/1] <> =>* <bad : X> does not terminate, since the following symbolic states (omitting some details for readability) appear while exploring the state space:

{s₀} : 0/1 ≤ #0-T || (l₀ : 0-T)
{s₁} : φ₀ and 5/1 ≤ #0-T and 10/1 ≥ #0-T + #1-T ∥ (l₁ : #0-T + #1-T)
{s₂} : φ₁ and 0/1 ≤ #2-T || (l₂ : #2-T)
{s₃} : φ₁ and 5/1 ≤ #2-T and 10/1 ≥ #2-T + #3-T ∥ (l₃ : #2-T + #3-T)
...

where φ₀ is the constraint in the state s₁. Note that [s₀] = [s₂] and [s₁] = [s₃] (i.e. s₃ represents the same set of concrete states as s₂). However, the constrained term s₀ is equivalent to s₂ and the smt-search command keeps exploring the successor states of s₃. Note also that, due to the definition of ∼, the constraints are always accumulated. For instance, φ₁ includes inequalities about #0-T and #1-T that are no longer used in the expression representing the value of the clock x.

We have therefore implemented our own symbolic reachability analysis command, which is based on the subsumption mechanism in [39]. Essentially, we stop searching from a symbolic state if, during the search, we have already encountered another state that subsumes it. More precisely, let U = φ₀ || t₀ and V = φ₁ || t₁. We define U ⊆ V, meaning that U is less general than V, if there is a substitution θ making t₀ and t₁θ equal, and the implication φ₁ = φ₀θ holds. In that case, [[U]] ⊆ [[V]] [39]. Hence, during the search procedure, if a term U is reached and some term V has already been visited s.t. U ⊆ V, the state U will not be further explored. It is known that such reachability analysis with folding is sound and generates no spurious counterexample [10].

The syntax of the implemented command is

red reachability((φ || ℓ), ℓ, bound).

The second and third parameters are optional. This command computes all the reachable symbolic states, using folding, starting from φ || ℓ until either: (1) no new states can be reached; (2) the location ℓ is reached; or (3) the search exceeds the depth bound.

We could quickly implement a prototype of our new symbolic reachability analysis algorithm using Maude’s meta-programming features. For instance, the function metaMatch applied to two terms U and V returns the set of substitutions θ such that U equals Vθ, and the function metaCheck can be used to delegate to the SMT solver the task of checking whether the formula ¬(φₐ ⇒ φₐθ) is unsatisfiable (and hence, the implication valid). Details about the implementation can be found in the companion repository [8].

Example 4.10. For the automaton in Fig. 3b, the command

red reachability((X >> 0/1) || < 10 : X < >).

computes the set of reachable states starting from location ℓ₀, with any non-negative initial clock value. The result is:
Theorem 4.11 (Termination). If the PZG \((C, S_0, \Rightarrow)\) of a PTA \(A\) is a finite transition system, then \((T^P_Z, U_0, \leadsto_{[A]})\) is also a finite transition system.

Our new reachability analysis command therefore terminates whenever the PZG is finite. Furthermore, it terminates when Imitator terminates with default settings since they both use subsumption, so generate the same part of the PZG. However, Imitator also uses heuristics that may synthesize parameters even if the PZG is infinite.

5 Parameter Synthesis and Analysis

Our executable rewriting-modulo-SMT semantics for PTAs gives us the possibility of applying different formal analysis methods for rewrite theories to PTAs. Section 5.1 shows how various parameter synthesis and parametric reachability problems can be solved with our methods, and Section 5.2 exemplifies how we can use Maude’s strategy language to analyze a PTA with a given strategy. In both cases we also provide model checking for PTA properties that go beyond those handled by state-of-the-art tools such as Imitator.

5.1 Reachability and EF-synthesis

This section shows how the smt-search and reachability commands can solve important synthesis and parametric reachability problems for PTAs.

A state predicate is a boolean expression whose atomic propositions are locations (e.g. the formula add_sugar holds if the current location is add_sugar) and inequalities on clocks and parameters (e.g. \(x_1 \neq x_2\)).

Definition 5.1. Let \(A\) be a PTA and \(\phi\) a state predicate. The EF-emptiness problem asks: “is the set of parameter valuations \(\sigma\) such that there exists a reachable state \((t, w)\) in \(\sigma(A)\) satisfying \(\phi\) empty?” EF-synthesis is the problem of computing parameter valuations \(\sigma\) such that a run of \(\sigma(A)\) reaches a state satisfying \(\phi\). The safety synthesis problem \(AG \neg \phi\) is the problem of computing the set of parameter valuations for which states satisfying \(\phi\) are unreachable.

The search commands provide semi-decision procedures (the number of symbolic states can be infinite and the synthesis problem is undecidable) for solving the above synthesis problems. We add \([[\ell]]_K = L \implies \ell\) (for a variable \(L\) of sort Location) to the definition of \([[\ell]]_K\). The command

\[
\text{smt-search } [1] [t_0 : 0/1 ; \ldots ; 0/1 ] < P_1 ; \ldots ; P_m > \implies \not< L : \text{Location} : X_1 ; \ldots ; X_n \not< P_1 ; \ldots ; P_m > \text{ such that } (\phi)_K \text{ and } P_1 \not< 0/1 \text{ and } \ldots \text{ and } P_m \not< 0/1 \\text{ then tries to find a path from } t_0 \text{ to an arbitrary location } L \text{ satisfying } \phi. \text{ The resulting constraint, if any, is an answer to the synthesis problem } EF\phi. \text{ The command reachability can be used similarly.}
\]

EF-emptiness is obtained when the EF-synthesis terminates without finding a path. Finally, the safety synthesis problem \(AG \neg \phi\) can be solved by finding all solutions for \(EF\phi\) and then negating the resulting constraint.

Example 5.2. Let \(\phi\) be the output of the smt-search command in Example 4.1. Since \(\phi\) is satisfiable, there are values for the parameters such that \(\text{done}\) is reachable and the answer to the EF-emptiness problem \(EF(\text{done})\) is false. The obtained constraint also gives us an answer to the corresponding EF-synthesis problem as follows. Since the result of the parameter synthesis only concerns the relations on the parameters, we are interested in the formula \(\phi' = \exists X. \phi\), where \(X\) includes all the variables in \(\phi\), but not the parameters. Using a quantifier elimination procedure, \(\phi'\) can be simplified to \(0 \leq P_2 \land P_2 \leq P_3 \land 0 \leq P_1\). (We are currently using the tactic qe of the Z3 theorem prover to automate this step). This means that \(\text{done}\) is reachable whenever \(P_2 \leq P_1\).

The EF-synthesis problem \(EF(x_1 \neq x_2 \land \text{preparing_coffee})\), asking whether location \text{preparing_coffee} is reachable with different values for the clocks, can be answered by:

\[
\text{smt-search } [1,10] [idle : 0/1 ; 0/1 ] < P_1 ; P_2 ; P_3 > \implies \not< L : X_1 \land X_2 > < P_1 ; P_2 ; P_3 > \text{ such that } (L = \text{preparing_coffee} \land X_1 \not= X_2 \text{ and } P_1 \not= 0/1 \text{ and } P_2 \not= 0/1 \text{ and } \not< P_3 > \text{ true }.
\]

The resulting constraint, after removing the existential quantifiers, determines that \(P_1 \leq P_2 \leq P_3\).

Finally, consider the safety synthesis problem \(AG \neg(x_1 > x_2)\). As explained above, we need to compute the solutions for \(EF(x_1 > x_2)\). This PTA has infinitely many symbolic states, since each extra iteration adding more sugar further constrains the values for \(P_1\) and \(P_2\). The command

\[
\text{smt-search } [1,10] [idle : 0/1 ; 0/1 ] < P_1 ; P_2 ; P_3 > \implies \not< L : X_1 > X_2 > < P_1 ; P_2 ; P_3 >
\]
such that $X_1' > X_2'$ and $P_1 \geq 0/1$ and $P_2 \geq 0/1$ and $P_3 \geq 0/1$.

It is worth remarking that Imitator only supports properties over locations but not over clocks. UPFAAL allows such properties, but does not support parameter synthesis. Our work therefore provides new analysis capabilities for PTAs.

### 5.2 Strategies

We can use Maude’s strategy language to analyze PTAs with different execution strategies. As exemplified below, such strategies can be defined on constrained terms to restrict the reachable symbolic states in $(T_Z, \phi_0 \parallel t_0, \bigwedge \exists \mathbf{k} \parallel \mathbf{a}[\mathbf{k}])$.

**Example 5.3 (Strategies).** The answer $P_2 \leq P_3$ to the synthesis problem $EF(\text{done})$ in Example 5.2 does not constrain $P_1$. This is due to the possibility of moving from idle to done without adding sugar. What if the same PTA needs to be analyzed under the assumption that at least one dose of sugar is required? Instead of manually modifying the PTA—which is error-prone and raises questions about whether the different models are consistent—we can define the following strategy to analyze our model when some sugar is required:

```
--- Strategy declarations
strat with-sugar : Nat @ State . --- Str. with parameter
strat add-sugar : Nat @ State .
--- Strategy definitions
sd with-sugar(N) :=
  match C || (done : X_1 ; X_2 < P_1 ; P_2 ; P_3 >)
  | (add-sugar : X_1 ; X_2 < P_1 ; P_2 ; P_3 > s.t. validity(C implies X_1 == X_2))
    add-sugar(N) ; with-sugar(N)
  | no rule
  all ; with-sugar(N).
--- Adding n doses of sugar
sd add-sugar(N) := idle . --- No more sugar
sd add-sugar(s(N)) := add-sugar ; add-sugar-tick ; add-sugar(N).
```

The strategy `with-sugar(N)`: (i) tests if the current location is `done` and stops if that is the case; (ii) if the location is `add-sugar`, it checks whether the accumulated constraint $C$ implies that the two clocks have the same value (`validity(C)` uses the SMT solver to check whether the formula is unsatisfiable); if so, the strategy `add-sugar(N)` is applied, forcing $N$ iterations in the location `add-sugar`; (iii) otherwise, the other rules of the system (all) are applied.

The command below returns a boolean expression that, after simplification, entails $P_1 \leq P_2 \leq P_3$.

```
srew P_1 \geq 0/1 and P_2 \geq 0/1 and P_3 \geq 0/1
  || [ idle : 0/1 ; 0/1 ] < P_1 ; P_2 ; P_3 >
```

Maude’s strategy language has not been used before in real-time systems specified in Maude or in rewriting with SMT. The example above shows that combining such techniques can lead to a novel mechanism to analyze PTAs. In particular, it is possible to perform reachability analyses on execution traces of the PTA that adhere to a given strategy. Furthermore, the resulting constraint determines the values of the parameters that enable such traces.

### 6 Benchmarks

In this section we compare the performance of Imitator, standard Maude-SE `smt-search`, and our prototype implementation of the command `reachability` on a number of PTAs in the PTA benchmark library [7]. We compare the time it takes for the three methods to solve the synthesis problem $EF(\ell)$ for different locations $\ell$ in the automaton, where all the queries have positive solutions. Figure 4 shows the execution times of Imitator and Maude (with red circles for `smt-search` and blue circles for `reachability`) in log-scale. The following table describes the PTAs considered in Fig. 4 (the complete set of benchmarks can be found in [8]):

<table>
<thead>
<tr>
<th>Model</th>
<th>Clocks</th>
<th>Parameters</th>
<th>Actions</th>
<th>Locations</th>
<th>Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>gear-1000</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>4074</td>
<td>4073</td>
</tr>
<tr>
<td>blowup-200</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>702</td>
<td>800</td>
</tr>
<tr>
<td>Pipeline_KP12_2_3</td>
<td>5</td>
<td>6</td>
<td>11</td>
<td>100</td>
<td>244</td>
</tr>
<tr>
<td>RCP</td>
<td>6</td>
<td>5</td>
<td>16</td>
<td>95</td>
<td>198</td>
</tr>
</tbody>
</table>

### 7 Related Work

#### Formal Analysis of Parametric Timed Automata

Since most analysis and parameter synthesis problems are undecidable for PTAs [4], approaches to address them have focused on heuristics. The state-of-the-art PTA tool, Imitator [5], uses techniques such as subsumption [40] and convex zone merging [6] to provide efficient bounded and unbounded reachability, EF-synthesis, deadlock checking, minimal-time reachability synthesis, and robustness analysis for PTAs.

As shown in Section 6, the PTA-specific Imitator tool generally has better performance than our Maude-with-SMT-based analysis. In addition, although our reachability command terminates whenever the PZG of the PTA is finite, additional heuristics implemented in Imitator allow it sometimes to terminate even when the PZG is infinite (and Maude will loop). Imitator also provides methods for liveness and robustness that we do not yet support for `[[A]]`. On the other hand, in this paper we show how we can analyze PTAs with user-defined strategies, and allow state properties that not only include locations but also conditions on clocks and parameters, which are not supported by Imitator.
Rewriting Semantics for Timed Automata. The paper [46] gives a formal semantics for timed automata using (real-time) rewrite theories. In contrast, our paper targets parametric timed automata, and provides a more elaborate “analysis-friendly” semantics than the one in [46], which was never meant/optimized for execution.

Analysis of Rewriting-based Real-Time Systems. As explained in the introduction, because of its expressiveness and generality, rewriting logic—in particular the Real-Time Maude [48, 49] extension of Maude—has been applied to a wide range of real-time systems [43] and has provided formal semantics and formal analysis to a number of modeling languages [42]. However, Real-Time Maude does not support symbolic analysis methods: when it applies a tick rule, it advances time by a given concrete value. Therefore, most system behaviors are not covered by the formal analysis, which is hence only sound for a restricted class of time-deterministic systems [47], and is not sound for timed automata. In contrast, in this paper we develop sound and complete symbolic analysis methods for a certain class of “time-nondeterministic” systems, namely, PTAs. Furthermore, the techniques seem general and should be applicable to other classes of real-time rewrite theories, which will be investigated in future work.

Rewriting with SMT has also been applied to formally analyze cyber-physical systems such as virtually synchronous systems [31] and soft agents [41]. They focus on hybrid systems with continuous dynamics, and do not consider parametric timed automata.

8 Concluding Remarks
A wide range of sophisticated real-time systems can be formalized in rewriting logic and formally analyzed in (Real-Time) Maude, which is also a suitable semantic framework and formal analysis backend for industrial modeling languages. So far Real-Time Maude has only provided explicit-state analysis methods, which are not sound for many real-time systems, including timed automata. It is clear that symbolic methods are needed for sound and efficient analysis of real-time systems. The recent integration of Maude and SMT solving has made symbolic analysis in Maude possible.

In this paper we take the first steps towards providing sound and efficient symbolic analysis methods for real-time rewrite theories by developing sound and complete analysis methods for parametric timed automata (PTAs), specified as rewrite theories. Since standard Maude-with-SMT reachability analysis does not terminate for real-time systems when the desired states are unreachable, we develop and implement (a prototype of) a general “folding”-based symbolic reachability analysis method and show that it terminates when the reachable symbolic state space of the PTA is finite. We show how our methods can be used to solve important parameter synthesis problems for PTAs. We also provide analysis methods for PTAs that are not supported by the Imitator tool, including symbolic reachability analysis combined with user-defined analysis strategies, and allowing clocks and parameters in state propositions. Furthermore, our executable semantics together with Maude’s meta-programming features provide an environment where new analysis methods for PTAs can be quickly developed and tested before being hard-coded into the Imitator tool.

In future work we should: develop symbolic methods for larger classes of real-time rewrite theories; develop a useful timed strategy language; and extend Maude’s and Real-Time Maude explicit-state LTL and timed CTL model checkers to the symbolic setting. These extensions will then also provide powerful new analysis methods for PTAs.

Acknowledgments. We thank the anonymous FTSCS reviewers for their insightful comments. Arias, Olarte, Olveczky, Petrucci and Rømming acknowledge support from CNRS under the project ESPRITS. Bae was supported in part by the National Research Foundation of Korea (NRF) grants funded by the Korea government (MSIT) (No. 2021R1A5A1021944 and No. 2022R1F1A1074550).

Figure 4. Execution times of Imitator and Maude in log-scale. From left to right, we consider the benchmarks (see Section 6): gear–1000, blowup–200, Pipeline_KP12_2_3, RCP.


Received 2022-09-08; accepted 2022-10-10