Rewriting Logic Semantics and Symbolic Analysis for Parametric Timed Automata
Jaime Arias, Kyungmin Bae, Carlos Olarte, Laure Petrucci, Peter Csaba Ölveczky, Fredrik Rømming

To cite this version:
Jaime Arias, Kyungmin Bae, Carlos Olarte, Laure Petrucci, Peter Csaba Ölveczky, et al.. Rewriting Logic Semantics and Symbolic Analysis for Parametric Timed Automata. [Research Report] LIPN. 2022. hal-03814256

HAL Id: hal-03814256
https://hal.archives-ouvertes.fr/hal-03814256
Submitted on 13 Oct 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Rewriting Logic Semantics and Symbolic Analysis for Parametric Timed Automata*

Jaime Arias\(^1\), Kyungmin Bae\(^2\), Carlos Olarte\(^1\), Peter Csaba Ölveczky\(^3\), Laure Petrucci\(^1\), and Fredrik Rømmin\(^3\)

\(^1\)LIPN, CNRS UMR 7030, Université Sorbonne Paris Nord
\(^2\)Pohang University of Science and Technology
\(^3\)University of Oslo

October 13, 2022

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.

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

Contents

1 Introduction 2
2 Preliminaries 3
3 A Rewriting Logic Semantics for PTA 6
\hspace{0.5em} 3.1 The PTA to Rewrite Theory Transformation 6
\hspace{0.5em} 3.2 Correctness of the \[ \square \] Transformation 9
4 Symbolic Reachability Analysis 11
\hspace{0.5em} 4.1 Symbolic Reachability Analysis 12
\hspace{0.5em} 4.2 Soundness and Completeness 12
\hspace{0.5em} 4.3 Symbolic Reachability Analysis with Folding 14
5 Parameter Synthesis and Analysis 16
\hspace{0.5em} 5.1 Reachability and EF-synthesis 17
\hspace{0.5em} 5.2 Strategies 18

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 [AD94] are a popular formalism for modeling real-time systems, and the timed automaton tool UPPAAL [DLL+15] has been applied to a wide range of safety-critical applications, including automotive [LMT15, LPY01], airborne [CKJ+22], and fire fighting [WWG+15] systems.

Parametric timed automata (PTA) [AHV93] 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 [And21], which has been applied to a number of systems, including protocols [HRSV02, KPT12, JLR15], an asynchronous circuit commercialized by ST-Microelectronics [CEFX09], and a distributed architecture for the flight control system of spacecraft designed at ASTRIUM Space Transportation [FSLM12].

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 [Mes92] and its associated tool Maude [CDE+07] are on the other side of the expressiveness spectrum, and support the above features. The Real-Time Maude tool [OM08, OM07] 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 [OMT06, L09], scheduling protocols with unbounded queues [OC06], state-of-the-art wireless sensor network protocols [OT09], MANET protocols [LO16], turning control algorithms for aircraft [BKMO15], human multitasking [MBO19], large cloud-based transaction systems [BGG18, GO14], and so on (see [olv14] 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 [BOF+12], AADL [OBM10], a language developed at DOCOMO Labs [ADY+09], and others have been given a formal semantics and formal analysis capabilities [olv11].

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 [OM06]. 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 [RMM17], and implemented in the Maude-SE tool [YB20].

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 5 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 6 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 [AMvdP21].

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 hardcoded them in a tool. Third, Maude provides full (explicit-state) LTL model checking, and Real-Time Maude provides timed CTL model checking [L´A¨O15]; 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 UP{PAAL}. 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 companion repository of this paper [ABO+22] 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 [CGP01], parametric timed automata [AHV93], rewriting logic [Mes92], rewriting modulo SMT [RMM17], and Maude [CDE+07] and its strategy language [CDE+22].

Transition Systems and Bisimulations. A transition system \( A \) is a triple \((A, a_0, \rightarrow_A)\), where \( A \) is a set of states, \( a_0 \in A \) is the initial state, and \( \rightarrow_A \subseteq A \times A \) is a transition relation. A function \( h : A \rightarrow B \) is a bisimulation from \( A \) to \( B \) iff: (i) \( h(a_0) = b_0 \); and (ii) for each \( a, a' \in A \), if \( a \rightarrow_A a' \) then \( h(a) \rightarrow_B h(a') \), and if \( h(b) \rightarrow_B b \) then there exists \( a'' \in A \) with \( a \rightarrow_A a'' \) and \( h(a'') = b \).

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 \( \sum_i \alpha_i p_i + \beta \), where \( p_i \in P \) and \( \alpha_i, \beta \in \mathbb{Q} \). A (diagonal) inequality has the form \( x_1 - x_2 \bowtie plt \), with \( x_i \in X \cup \{0\} \) and \( \bowtie \in \{<, \leq, =, \geq, >\} \). 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) \( A \) is a tuple \( (\Sigma, L, \ell_0, X, P, I, E) \), where \( \Sigma \) is a finite set of actions, \( L \) is a finite set of locations, \( \ell_0 \in L \) 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 \( (\ell, g, \sigma, R, \ell') \), with source \( \ell \in L \), target \( \ell' \in L \), guard \( g \in C \), action \( \sigma \in \Sigma \), and clock reset \( R \subseteq X \).

A parameter 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 write \( 0 \) for the clock valuation s.t. \( \forall x \in X : 0(x) = 0 \). We extend parameter valuations to linear terms. We write \( v, w \models (x_i - x_j \bowtie plt) \) if \( w(x_i) - w(x_j) \bowtie v(plt) \), and \( v, w \models Z \) iff \( v, w \models e \) for each inequality \( e \) in the zone \( Z \).

Given a parameter valuation \( v \), we write \( v(A) \) for the timed automaton (TA) obtained by replacing each parameter \( p \) in invariants and guards by \( v(p) \). The concrete semantics of a PTA \( A \) is derived from that of the TA \( v(A) \), 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 \( \rightarrow = (\rightarrow_0; \rightarrow_c) \), where continuous time delay \( (\rightarrow_0) \) and discrete transitions \( (\rightarrow_c) \) are defined as

- If \( d \in \mathbb{R}_{\geq 0} \) and \( w + d \models I(\ell) \), then \( (\ell, w) \rightarrow_0 (\ell, w + d) \).
- If \( e = (\ell, g, \sigma, R, \ell') \in E \) and \( w \models g \) and \( w[R] = I(\ell') \) then \( (\ell, w) \rightarrow_c (\ell', w [R]) \).
Example 1. The PTA in Fig. 1—with 4 locations, 2 clocks ($x_1$ and $x_2$) and 3 parameters ($p_1, p_2, p_3$)—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 ($x_2 \leq p_2$) is satisfied; there, the user can add a dose of sugar by pressing the button $bSugar$, provided the guard ($x_1 \geq p_1$) is satisfied, which resets $x_1$. Then, $p_2$ time units after the $bStart$ button was last pushed, a cup is delivered (action $cup$), and the coffee is being prepared; $p_3$ 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 timed transition system, it may be (countably) infinite. We define the following operations on zones:

- **Time elapse**: $Z' \overset{\text{def}}{=} \{ (v, w + d) \mid d \in \mathbb{R}_{\geq 0} \land v, w \models Z \}$
- **Clock reset**: $Z[R] \overset{\text{def}}{=} \{ (v, w[R]) \mid v, w \models Z \}$

The PZG is a transition system where each abstract state consists of a location and a non-empty zone. The PZG of $A = (\Sigma, L, \ell_0, X, P, I, E)$ is $(S, s_0, \rightarrow)$, with $S \subseteq L \times C$, initial state $s_0 = (\ell_0, (\bigwedge_{x \in X} x = 0)^c \cap I(\ell_0))$. A transition step $(\ell, Z) \rightarrow (\ell', Z')$ exists if for some $(\ell, g, \sigma, R, \ell') \in E$ we have $Z' = ((Z \cap g[R]) \cap I(\ell')) \cap I(\ell) \neq \emptyset$. We write $\Rightarrow$ for the reflexive-transitive closure of $\rightarrow$.

Example 2. Figure 2 presents the beginning of the parametric zone graph of the coffee machine in Example 1.

Rewrite Theories. An order-sorted signature $\Sigma$ is a triple $(S, \leq, F)$ with $S$ a set of sorts, $\leq$ 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 $T_{\Sigma}$ 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 $E = (\Sigma, E)$, 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 $u =_E u'$ iff $(\Sigma, E) \vdash (\forall X) u = u'$ [Mes97].

A rewrite theory $\mathcal{R}$ is a tuple $\mathcal{R} = (\Sigma, E, L, R)$, where $(\Sigma, E)$ is an equational theory, $L$ 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.

$t \rightarrow_\mathcal{R} t'$ is a (one-step) rewrite if there is a rule $l : q \rightarrow r$ if $\psi$, a subterm $u$ of $t$, and a substitution $\theta$ such that $u =_E q\theta$ and $t'$ is obtained from $t$ by replacing the subterm $u$ with $r\theta$, provided $\psi\theta = \psi'$ holds for each equation $v = v'$ in $\psi$. We denote by $\rightarrow^*_{\mathcal{R}}$ the reflexive-transitive closure of $\rightarrow_{\mathcal{R}}$. 

---

Figure 1: A coffee machine ($CM$) modeled as a PTA.
A rewrite theory \( \mathcal{R} \) is called topmost iff there is a sort \( \text{State} \) at the top of one of the connected components of the poset \( (\mathcal{S}, \preceq) \) such that for each rule \( l : q \to r \) if \( \psi \), both \( q \) and \( r \) have the top sort \( \text{State} \), and no operator has sort \( \text{State} \) or any of its subsorts as an argument sort.

### 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 \( f : w \to s \) in \( \Sigma \setminus \Sigma_0 \), \( s \not\in \Sigma_0 \) and \( f \) has no other subsort-overloaded typing in \( \Sigma_0 \). Satisfiability of a constraint in \( \mathcal{E}_0 \) is assumed to be decidable using the SMT theory \( \mathcal{T}_{\mathcal{E}_0} \) which is consistent with \( (\Sigma, E) \): for \( t_1, t_2 \in T_{\Sigma_0} \), if \( t_1 =_E t_2 \), then \( \mathcal{T}_{\mathcal{E}_0} \models t_1=t_2 \) \[\text{RMM17}].

A constrained term is a pair \( \phi \parallel t \) of a constraint \( \phi \) in \( \mathcal{E}_0 \) and a term \( t \in T_{\Sigma}(X_0) \) over variables \( X_0 \subseteq X \) of the built-in sorts in \( \mathcal{E}_0 \) \[\text{RMM17} \text{ BR19}]. A constrained term \( \phi \parallel t \) symbolically represents all instances of the pattern \( t \) such that \( \phi \) holds:

\[ \phi \parallel t = \{ t' \mid t' =_E t \theta \text{ and } \mathcal{T}_{\mathcal{E}_0} \models \phi \theta \text{ for ground } \theta : X_0 \to T_{\Sigma_0} \}. \]

A symbolic rewrite on constrained terms symbolically represents a (possibly infinite) set of system transitions. Let \( \mathcal{R} \) be a topmost theory such that for each rule \( l : q \to r \) 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 \( \mathcal{E}_0 \). Then, a one-step symbolic rewrite \( \phi \parallel t \to_\mathcal{R} \phi' \parallel t' \) holds iff there exist a rule \( l : q \to r \) if \( \psi \) and a substitution \( \theta : X \to T_{\Sigma}(X_0) \) such that (1) \( t =_E q \theta \), (2) \( t' =_E r \theta \), (3) \( \mathcal{T}_{\mathcal{E}_0} \models (\phi \land \psi \theta) \Rightarrow \phi' \), and (4) \( \phi' \) is \( \mathcal{T}_{\mathcal{E}_0} \)-satisfiable. We denote by \( \to_\mathcal{R} \) the reflexive-transitive closure of \( \to_\mathcal{R} \).

If \( \phi_t \parallel t \to_\mathcal{R} \phi_u \parallel u \) is a symbolic rewrite, then there exists a “concrete” rewrite \( t' \to u' \) with \( t' \in [\phi_t \parallel t] \) and \( u' \in [\phi_u \parallel u] \). Conversely, for any concrete rewrite \( t' \to u' \) with \( t' \in [\phi_t \parallel t] \), there exists a symbolic rewrite \( \phi_t \parallel t \to_\mathcal{R} \phi_u \parallel u \) with \( u' \in [\phi_u \parallel u] \).

### Maude

Maude \[\text{CDE}^*07\] 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:

```plaintext
mod M is ... endm  --- Rewrite theory M
pr R.             --- Importing a theory R
sorts S ... Sk.   --- Declaration of sorts S1,..., Sk
subsort S1 < S2.   --- Subsort relation
vars X1 ... Xm : S. --- Logical variables of sort S
op f : S1 ... Sn -> S. --- Operator S1 x ... x Sn -> S
op c : -> T.        --- Constant c of sort T
ceq t = t' if c.    --- Conditional equation
crl [1] : q => r if c. --- Conditional rewrite rule
```

Figure 2: The PZG of the coffee machine example \( CM \).
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

\texttt{smt-search \([n, m]: t =>* t'\) such that \(\Phi\).}

symbolically searches for \(n\) states, reachable from \(t \in T_\Sigma(X_0)\) within \(m\) steps, that match the pattern \(t' \in T_\Sigma(X)\) and satisfy the constraint \(\Phi\) in \(E_0\). More precisely, it searches for a constrained term \(\phi_u \mid u\) such that true \(\mid t \Rightarrow^* \phi_u \mid u\) and for some \(\theta: X \rightarrow T_\Sigma(X)\), \(u = E \circ t'\theta\) and \(T_{E_0} \models \phi_u \Rightarrow \Phi\theta\). The parameters \([n,m]\) are optional.

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

Maude supports \texttt{meta-programming}, where a Maude module \(M\) (resp. a term \(t\)) can be (meta-)represented as a Maude \texttt{term} \(\bar{M}\) of sort \texttt{Module} (resp. as a Maude term \(\bar{t}\) of sort \texttt{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 \texttt{metaReduce}, \texttt{metaRewrite}, \texttt{metaMatch}, and \texttt{metaCheck}.

Maude-SE [YB20] extends Maude with additional functionality for rewriting modulo SMT, including witness generation for \texttt{smt-search}. It uses two theory transformations to implement symbolic rewriting [RMM17]. In essence, a rewrite rule \(l: q \rightarrow r\) if \(\psi\) is transformed into a constrained-term rule

\[
l: \text{PHI} \mid q \rightarrow (\text{PHI and } \psi) \mid r \text{ if } \text{smtCheck} (\text{PHI and } \psi)
\]

where \text{PHI} is a \texttt{Boolean} variable, and \texttt{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 [CIDE+22] allows us to define strategies for applying the rewrite rules. The command \texttt{srew \(t\) using \text{str}\} rewrites the term \(t\) according to the strategy \text{str} and returns all its results. Basic strategies include the application of a rule \(l\) once anywhere in the term (strategy \(l\), and \texttt{top} \((l)\) for rewriting at the top of term using rule \(l\); \texttt{all} denotes all rules), \texttt{idle} (identity), \texttt{fail} (empty set), and \texttt{match} \(P\) \texttt{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 \(A\) into a rewrite theory \([A]\). Section 3.2 provides a bisimulation result relating the concrete semantics of \(A\) and a rewrite relation induced by \([A]\).

#### 3.1 The PTA to Rewrite Theory Transformation

We fix \(A\) to be the PTA \((\Sigma, L, \ell_0, X, P, I, E)\) with \(n = |X|\) clocks \((x_1, \ldots, x_n)\), \(m = |P|\) parameters \((p_1, \ldots, p_m)\), and \(k = |L|\) locations \({\ell_1, \ldots, \ell_k}\). The idea is to represent a concrete state \((\ell, w)\) of the PTA \(A\) 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 \(v(A)\) \(i.e.\) the PTA \(A\) 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 \textit{one} such step, we use ”delayed” states.
< \ell : w(z_i) ; \ldots ; w(x_m) > < 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

\[
\text{crl} \ [\text{add\_sugar-bSugar}] : \\
< \text{add\_sugar} : X_1 ; X_2 > < P_1 ; P_2 ; P_3 > => \\
[ \text{add\_sugar} : 0/1 ; X_2 ] < P_1 ; P_2 ; P_3 > \\
\text{if} (X_1 => P_1 \text{ and } X_2 <= P_2) = \text{true}.
\]

Furthermore, for each location \( \ell \in L \), 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 \( \text{add\_sugar} \) is

\[
\text{crl} \ [\text{add\_sugar-tick}] : \\
< \text{add\_sugar} : X_1 ; X_2 > < P_1 ; P_2 ; P_3 > => \\
< \text{add\_sugar} : X_1 \text{ and } X_2 <= P_2 > 0/1 = \text{true} [\text{nonexec}] .
\]

Since time can advance by \( \text{any} \) 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 ([\text{nonexec}], see Section 4).

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

\[
\begin{align*}
\text{pr REAL} & \quad \text{--- SMT rational/real numbers} \\
\text{sorts State NState DState Location} & \quad \text{--- Sorts for states} \\
\text{subsorts NState DState < State} & \quad \text{--- Sorts for states} \\
\text{ops} & \ell_1 \ldots \ell_k := \rightarrow \text{Location} . \\
\text{--- Constants for locations} \\
\text{ops} & \ell_1 \ldots \ell_k := \rightarrow \text{Location} . \\
\text{--- States of the system} \\
\text{op} [\ldots ; \ell_1 ; \ldots ] < \ldots ; \ell_k > : \text{Location Real} \ldots \text{Real} => \text{DState} . \\
\text{op} [\ldots ; \ell_1 ; \ldots ] < \ldots ; \ell_k > : \text{Location Real} \ldots \text{Real} => \text{NState} .
\end{align*}
\]

\( \mathcal{A} \) defines SMT-variables to represent clock valuations, parameter valuations, and time elapse:

\[
\begin{align*}
\text{vars} & \quad X_1 \ldots X_n : \text{Real} . \quad \text{--- Clock valuations} \\
\text{vars} & \quad P_1 \ldots P_m : \text{Real} . \quad \text{--- Parameter valuations} \\
\text{var} & \quad T : \text{Real} . \quad \text{--- Time elapse}
\end{align*}
\]

We define two functions \( \llbracket \_ \rrbracket_b \) and \( \llbracket \_ \rrbracket_e \) for translating parametric guards and invariants to terms, where

\[
\llbracket \text{true} \rrbracket_b = \text{true} \quad \llbracket b_1 \land b_2 \rrbracket_b = \llbracket b_1 \rrbracket_b \land \llbracket b_2 \rrbracket_b
\]

and for each inequality relation in \( \{ \geq, \leq, =, >, < \} \), we have, e.g.: \( \llbracket e_1 \leq e_2 \rrbracket_b = \llbracket e_1 \rrbracket_e \land \llbracket e_2 \rrbracket_e \) and \( \llbracket e_1 = e_2 \rrbracket_b = \llbracket e_1 \rrbracket_e = \llbracket e_2 \rrbracket_e \). For arithmetic expressions, we define:

\[
\begin{align*}
\llbracket e_1 + e_2 \rrbracket_e & = \llbracket e_1 \rrbracket_e + \llbracket e_2 \rrbracket_e \quad \llbracket x_i^R \rrbracket_e = x_i \text{ if } x_i \notin R \\
\llbracket e_1 - e_2 \rrbracket_e & = \llbracket e_1 \rrbracket_e - \llbracket e_2 \rrbracket_e \quad \llbracket x_i \rrbracket_e = x_i \\
\llbracket e_1 \cdot e_2 \rrbracket_e & = \llbracket e_1 \rrbracket_e \cdot \llbracket e_2 \rrbracket_e \quad \llbracket x_i^T \rrbracket_e = x_i + T \\
\llbracket p/q \rrbracket_e & = p/q \text{ if } p, q \in \text{N} \quad \llbracket p \rrbracket_e = p_i \\
\llbracket x_i^R \rrbracket_e & = 0/1 \text{ if } x_i \in R 
\end{align*}
\]

\( \llbracket \_ \rrbracket \) maps each transition \( (\ell, g, \sigma, R, \ell') \in E \), to the following conditional rewrite rule \( \ell\sigma \):

\[
\text{crl} \ [\ell\sigma] : < \ell : X_1 ; \ldots ; X_n > < P_1 ; \ldots ; P_m > => [ \ell' : \llbracket x_i^R \rrbracket_e ; \ldots ; \llbracket x_i^R \rrbracket_e ] < P_1 ; \ldots ; P_m > \\
\text{if} [g \land I(\ell')\llbracket x_i^R \rrbracket_e]_b = \text{true} .
\]

where \( I(\ell')\llbracket x_i^R \rrbracket_e \) denotes substituting \( x_i^R \) for \( x_i \) in the expression \( I(\ell') \) for each \( i \). Furthermore, for each \( \ell \in L \), \( \llbracket \_ \rrbracket \) adds a conditional rewrite rule \( \ell\text{-tick} \):

\[
\text{crl} \ [\ell\text{-tick}] : [ \ell : X_1 ; \ldots ; X_n ] < P_1 ; \ldots ; P_m > \\
\]
Example 3. \(_\) transforms the PTA CM in Fig. \([1]\) to the rewrite theory \([CM]\) below.

```plaintext
mod PTA-COFFEE is
pr REAL.
sorts State NState DState Location.
sorts NState DState < State.

--- ---------------------------
vars X1 X2 : Real.
vars P1 P2 P3 : Real.
var T : Real.
--- ---------------------------

--- Configurations
ops idle add_sugar preparing_coffee done : -> Location [ctor].
op <_::_> <_::_> : Location Real Real Real Real Real -> DState [ctor].
op <_::_> <_::_> : Location Real Real Real Real Real -> NState [ctor].

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

crl [idle-tick] :
  [ idle : X1 ; X2 ] < P1 ; P2 ; P3 > =>
  < idle : X1 + T ; X2 + T > < P1 ; P2 ; P3 >
  if (true and T >= 0/1) = true [nonexec].

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

crl [add_sugar-cup] :
  < add_sugar : X1 ; X2 > < P1 ; P2 ; P3 > =>
  [ preparing_coffee : X1 ; X2 ] < P1 ; P2 ; P3 >
  if (X2 === P2 and X2 <= P3) = true .

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

crl [preparing_coffee-coffee] :
  < preparing_coffee : X1 ; X2 > < P1 ; P2 ; P3 > =>
  [ done : 0/1 ; X2 ] < P1 ; P2 ; P3 >
  if (X2 === P3 and 0/1 <= 10/1) = true .

crl [preparing_coffee-tick] :
  [ preparing_coffee : X1 ; X2 ] < P1 ; P2 ; P3 > =>
  [ preparing_coffee : X1 + T ; X2 + T ] < P1 ; P2 ; P3 >
  if (X2 + T <= P3 and T >= 0/1) = true [nonexec].

crl [done-sleep] :
  < done : X1 ; X2 > < P1 ; P2 ; P3 > =>
  [ idle : X1 ; X2 ] < P1 ; P2 ; P3 >
  if (X1 === 10/1 and true) = true .

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

crl [done-tick] :
  [ done : X1 ; X2 ] < P1 ; P2 ; P3 > =>
  < done : X1 + T ; X2 + T > < P1 ; P2 ; P3 >
  if (X1 + T <= 10/1 and true) = true [nonexec].
endm
```
3.2 Correctness of the \( \llbracket \_ \rrbracket \) Transformation

In this section we relate \( A \) and \( \llbracket A \rrbracket \) by a bisimulation. Since a transition in \( A \) consists of a delay followed by a discrete transition, we define a corresponding rewrite relation \( \rightarrow_{\llbracket A \rrbracket} \) combining a “tick” rule application with a “transition” rule application. We then show that these respective relations in the concrete semantics of \( v(A) \) and in \( \llbracket A \rrbracket \) are bisimilar w.r.t. initial state \( t_0 : 0/1; \ldots; 0/1 < v(p_1); \ldots; v(p_m) > \).

**Definition 1.** Let \( A = (\Sigma, L, \ell_0, X, P, I, E) \) be a PTA and \( t_1, t_2, t_3 \) be terms of \( \llbracket A \rrbracket \). We write \( t_1 \rightarrow_{\llbracket A \rrbracket} t_3 \) if there exists a \( t_2 \) such that \( t_1 \rightarrow t_2 \) is a one-step rewrite applying an \( \ell \)-tick rule in \( \llbracket A \rrbracket \) for some \( \ell \in L \) and \( t_2 \rightarrow t_3 \) is a one-step rewrite applying an \( \ell \)-rule in \( \llbracket A \rrbracket \) for some \( \ell \in L \) and \( \sigma \in \Sigma \). Furthermore, we write \( t_1\rightarrow_{\llbracket A \rrbracket} t_2 \) to indicate that there exists a sequence of \( t \rightarrow_{\llbracket A \rrbracket} \) rewrites from \( t_1 \) to \( t_2 \).

Let \( v \) be a parameter valuation. \( (T_{\Sigma,NState})_v \) denotes the set of \( E \)-equivalence classes of ground terms of sort \( NState \) with parameter valuations \( v \), and \( S \) denotes the set of concrete states of \( v(A) \). We define a map \( \llbracket \_ \rrbracket_v : S \rightarrow (T_{\Sigma,NState})_v \), relating concrete states in \( A \) to states (of sort \( NState \)) in \( \llbracket A \rrbracket \), where for all concrete states \( (\ell, w) \in S \),

\[
\llbracket (\ell, w) \rrbracket_v = [\ell : \llbracket w(x_1) \rrbracket_v; \ldots; \llbracket w(x_n) \rrbracket_v] < \llbracket v(p_1) \rrbracket_v; \ldots; \llbracket v(p_m) \rrbracket_v >.
\]

Before proving the main theorem of this section, we need some lemmas.

**Lemma 1.** For all sets of clocks \( X \), zones \( Z \), parameter valuations \( v \), and clock valuations \( w \),

\[
v, w' \models Z \iff \llbracket Z[x_i/x'_i] \rrbracket_{\theta} \text{ holds in the following cases:}
\]

1. \( w' = w, x'_i = x_i, \text{ and } \theta = \{ X_i \mapsto \llbracket w(x_i) \rrbracket_v, P_i \mapsto \llbracket v(p_i) \rrbracket_v \}_i \).

2. \( w' = (w + d), d \in \mathbb{R}_{\geq 0}, x'_i = x_i, \text{ and } \theta = \{ X_i \mapsto \llbracket w(x_i) \rrbracket_v, P_i \mapsto \llbracket v(p_i) \rrbracket_v, T \mapsto \llbracket d \rrbracket_v \}_i \).

3. \( w' = w[R], R \subseteq X, x'_i = x_i \in R, \text{ and } \theta = \{ X_i \mapsto \llbracket w(x_i) \rrbracket_v, P_i \mapsto \llbracket v(p_i) \rrbracket_v \}_i \).

**Proof.** By induction on \( Z \).

- **Base case:** E.g., if \( Z \) is the inequality \( x_i - x_j \geq \sum_i (\alpha_i p_i) + \beta \), then for each combination of \( w', x'_i, \theta \):

  1. \( w(x_i) - w(x_j) \geq \sum_i (\alpha_i v(p_i)) + \beta \) holds iff \( \llbracket Z \rrbracket_v \theta = \llbracket w(x_i) \rrbracket_v - \llbracket w(x_j) \rrbracket_v \geq \llbracket \alpha_0 \rrbracket_v + \llbracket v(p_0) \rrbracket_v + \ldots + \llbracket \alpha_m \rrbracket_v + \llbracket \beta \rrbracket_v \).

  2. \( w(x_i) + d - w(x_j) + d \geq \sum_i (\alpha_i v(p_i)) + \beta \) holds iff \( \llbracket Z[x_i/x'_i] \rrbracket \theta = \llbracket w(x_i) \rrbracket_v + \llbracket d \rrbracket_v - \llbracket w(x_j) \rrbracket_v + \llbracket d \rrbracket_v \geq \llbracket \alpha_0 \rrbracket_v + \llbracket v(p_0) \rrbracket_v + \ldots + \llbracket \alpha_m \rrbracket_v + \llbracket \beta \rrbracket_v \).

  3. \( w[R](x_i) - w[R](x_j) \geq \sum_i (\alpha_i v(p_i)) + \beta \) holds iff \( \llbracket Z[x_i/x'_i] \rrbracket \theta = \llbracket w(x_i) \rrbracket_v \text{ if } x_i \in R \text{ else } 0 \rrbracket_v \geq \llbracket \alpha_0 \rrbracket_v + \llbracket v(p_0) \rrbracket_v + \ldots + \llbracket \alpha_m \rrbracket_v + \llbracket \beta \rrbracket_v \).

All the above statements hold since their left hand and right hand denote the “same” expression. Although one in mathematics, and the other in Maude.
• **Induction:** Let $Z = Z_1 \land Z_2$. By definition $v,w' \models Z_1 \land Z_2$ iff $v,w' \models Z_1$ and $v,w' \models Z_2$. We can assume the induction hypotheses that $v,w' \models Z_i$ iff $[[Z_i[x_i/x'_i]]]_{\theta} \models \theta$ holds (i.e., evaluates to true) for $i \in \{1,2\}$. Since $[[Z_i \land Z_j][x_i/x'_i]]_{\theta} \models [Z_i][x_i/x'_i] \land Z_j[x_i/x'_i]_{\theta}$ by definition of $\models$, this is equal to $[[Z_i[x_i/x'_i]]]_{\theta}$ and $[[Z_j[x_i/x'_i]]]_{\theta}$, which again, since a substitution is the homomorphic extension of a variable substitution, equals $[Z_i][x_i/x'_i]_{\theta}$ and $[Z_j][x_i/x'_i]_{\theta}$, which by the induction hypotheses equals true and true, which again is true.

\[ \square \]

**Lemma 2.** Let $A$ be a PTA, $v$ a parameter valuation, and $(\ell,w)$ and $(\ell',w')$ concrete states of $v(A)$. If $(\ell,w) \rightarrow (\ell',w')$, then $[[\ell,w]]_{v} \rightarrow_{A} [[\ell',w']]_{v}$.

**Proof.** By definition, $((\ell,w),(\ell',w')) \in \rightarrow$ if there are $d,w''$ such that $(\ell,w) \xrightarrow{d} (\ell,w'') \xrightarrow{\sigma} (\ell',w')$.

For the first step $\xrightarrow{d}$, we have $(\ell,w) \xrightarrow{d} (\ell,w'')$, which by definition, means that $v,(w+d) \models I(\ell)$. Let $t = [[\ell,w]]_{v} = [\ell : [w(x_1)]_{e} \quad \cdots \quad [w(x_n)]_{e}] < [v(p_1)]_{e} \quad \cdots \quad [v(p_m)]_{e} >$

We can apply the rule $\ell$-tick to $t$ to get $u =$

\[ [\ell' : [w(x_1)]_{e} + [d_1]_{e} \quad \cdots \quad [w(x_n)]_{e} + [d_n]_{e} > [v(p_1)]_{e} \quad \cdots \quad [v(p_m)]_{e} > \]

since there exists a substitution $\theta = \{X_i \mapsto [w''(x_i)]_{e}, P_i \mapsto [v(p_i)]_{e}, T \mapsto [d]_{e}\}$, such that (a) $t = E \theta$, $u = E \, r \, \theta$, and (b) $T_{\theta} = \phi$, where $l,r,\phi$ are the left-hand side, right-hand side, and condition of rule $\ell$-tick respectively. Note that (b) holds by Lemma [1] because $\phi \theta = ([I(\ell)[x_i/x'_i]]_{b}$ and $T \geq 0/1 \theta$ holds iff $v,(w+d) \models I(\ell)$, which is true by assumption.

For the second step $\xrightarrow{\sigma}$, we have $(\ell,w'') \xrightarrow{\sigma} (\ell',w')$, where $w'' = w + d$ and $w' = w'' [R]$, which means there is a discrete transition $(\ell,g,\sigma,R,\ell')$ in $A$ such that $v,w'' \models g$ and $v,w' \models I(\ell')$. We can apply the rule $l$-$\sigma$ to the above $u$ to get $u_d =$

\[ [\ell' : [w(x_1)]_{e} \quad \cdots \quad [w(x_n)]_{e}]_{d} \text{ if } x_i \in R \text{ else } 0/1 ; \quad \cdots \quad [v(p_i)]_{e} \text{ if } x_i \in R \text{ else } 0/1 \]

\[ [v(p_1)]_{e} \quad \cdots \quad [v(p_m)]_{e} > \]

since there exists a substitution $\theta = \{X_i \mapsto [w''(x_i)]_{e}, P_i \mapsto [v(p_i)]_{e}, T \mapsto [d]_{e}\}$, such that (a) $u = E \theta$, $y = E \, r \, \theta$, and (b) $T_{\theta} = \phi$, where $l,r,\phi$ are the left-hand side, right-hand side, and condition of rule $l$-$\sigma$ respectively. Again, note that (b) holds by Lemma [1] because $\phi \theta = [g \land I(\ell')][x_i/x'_i]_{b}$, which is true by assumption.

\[ \square \]

**Lemma 3.** Let $A$ be a PTA, $v$ a parameter valuation, $(\ell,w)$ a concrete state of $v(A)$, and $b \in (T_{\Sigma \text{-base state}})_v$. If $[[\ell,w]]_{v} \rightarrow_{A} b$ then there exists a concrete state $(\ell',w')$ in $v(A)$ such that $(\ell,w) \rightarrow (\ell',w')$ and $b = [[\ell',w']]_{v}$.

**Proof.** By Definition [1] $t_1 \rightarrow_{A} t_3$ if there exists a $t_2$ such that $t_1 \rightarrow t_2$ is a one-step sequential rewrite applying an $\ell$-tick rule in $[A]$ for some $\ell \in L$ and $t_2 \rightarrow t_3$ is a one-step sequential rewrite applying a $l$-$\sigma$ rule of $[A]$ for some $\sigma \in \Sigma$.

For the first step $\xrightarrow{\ell \text{-tick}}$, by assumption there exists a substitution $\theta = \{X_i \mapsto [w''(x_i)]_{e}, P_i \mapsto [v(p_i)]_{e}, T \mapsto [d]_{e}\}$, for some $d \in \mathbb{R}_+$, such that there exists a rewrite from $t = [[\ell,w]]_{v} =$

\[ [\ell : [w(x_1)]_{e} \quad \cdots \quad [w(x_n)]_{e}]_{d} \text{ if } x_i \in R \text{ else } 0/1 ; \quad \cdots \quad [v(p_i)]_{e} \text{ if } x_i \in R \text{ else } 0/1 \]

\[ [v(p_1)]_{e} \quad \cdots \quad [v(p_m)]_{e} > \]

to $u =$

\[ [\ell : [w(x_1)]_{e} \quad \cdots \quad [w(x_n)]_{e}]_{d} \text{ if } x_i \in R \text{ else } 0/1 ; \quad \cdots \quad [v(p_i)]_{e} \text{ if } x_i \in R \text{ else } 0/1 \]

\[ [v(p_1)]_{e} \quad \cdots \quad [v(p_m)]_{e} > \]

for at least one $d \in \mathbb{R}_+$. For this rewrite to be enabled, the condition $([I(\ell)[x_i/x'_i]]_{b}$ and $T \geq 0/1 \theta$ must hold. By Lemma [1] $([I(\ell)[x_i/x'_i]]_{b}$ and $T \geq 0/1 \theta$ holds iff $v,(w+d) \models I(\ell)$. Since $v,(w+d) \models I(\ell)$, there exists a concrete state $(\ell,w+d) = (\ell,w'')$ in $A$ such that $(\ell,w) \rightarrow (\ell,w+d)$.

For the second step $\xrightarrow{\sigma}$, the term $u =$

10
<\ell : [w(x_1)]_e + [d]_e; \ldots; [w(x_n)]_e + [d]_e > < [v(p_1)]_e; \ldots; [v(p_m)]_e >
above is rewritten to \( u_2 = \)
\[ \ell' : ([w(x_1)]_e + [d]_e) \text{ if } x_1 \in R \text{ else } 0/1 ; \ldots; ([w(x_n)]_e + [d]_e) \text{ if } x_n \in R \text{ else } 0/1 \]
\[ < [v(p_1)]_e; \ldots; [v(p_m)]_e > \]
using the rule \( \ell \to u \) in \([A]\). By definition of \([A]\), there is a corresponding discrete transition \((\ell, g, \sigma, R, \ell')\) in \(A\). The matching substitution used in the rewrite is then \( \theta = \{ \text{If } x_i \to [w''(x_i)]_e, \text{Pi } \to [v(p_i)]_e \} \). Since this rewrite took place, \( \theta \), the condition of the rule, \((g \wedge I(\ell')|x_i/x_i'|)|_\theta\), must hold. By Lemma 1, \( \phi_{\theta} = \|[g \wedge I(\ell')|x_i/x_i'|]|_\theta \) holds iff \( v.(w + d) = g \) and \( v.((w + d)[R]) = I(\ell') \). Therefore we can apply the discrete transition \((\ell, g, \sigma, R, \ell')\) in \(A\), to go from \((\ell, w + d)\) to \((\ell', (w + d)[R])\). Altogether, for any PTA-rewrite
\[ \ell : [w(x_1)]_e; \ldots; [w(x_n)]_e ; [v(p_1)]_e; \ldots; [v(p_m)]_e > \]
\[ \ell - \text{tick} \]
\[ \ell' : ([w(x_1)]_e + [d]_e) \text{ if } x_1 \in R \text{ else } 0/1 ; \ldots; ([w(x_n)]_e + [d]_e) \text{ if } x_n \in R \text{ else } 0/1 \]
\[ < [v(p_1)]_e; \ldots; [v(p_m)]_e > \]
there is the corresponding PTA-transition
\[ (\ell, w) \xrightarrow{d} (\ell, w + d) \xrightarrow{\sigma} (\ell', (w + d)[R]) \]
where indeed \([[(\ell, w)]_v = \)
\[ [(\ell : [w(x_1)]_e; \ldots; [w(x_n)]_e ; [v(p_1)]_e; \ldots; [v(p_m)]_e > \]
and \([[(\ell, (w + d)[R])]_v = \)
\[ [(\ell' : [w(x_1)]_e + [d]_e; \ldots; [w(x_n)]_e + [d]_e ; [v(p_1)]_e; \ldots; [v(p_m)]_e > \]
Hence, for all \( b \) such that \([[(\ell, w)]_v \xrightarrow{\sigma} [A] b \), there exists a concrete state \((\ell', w')\) in \(v(A)\) such that \((\ell, w) \xrightarrow{d} (\ell', w')\) and \( b = [((\ell', w')]_v \).

**Theorem 1.** Let \( A = (\Sigma, L, \ell_0, X, P, I, E) \) be a parametric timed automaton, \( v(A) = (S, s_0, \rightarrow) \) be \(A\)'s concrete semantics with respect to a parameter valuation \( v \), and \([A] = (\Sigma, E, L, R)\). Then, \([_\to_v \) is a bisimulation map between the transition systems \((S, s_0, \rightarrow)\) and \((\langle T_{\Sigma.ESize} \rangle_v, [s_0]_{\to_v}[A]\).

**Proof.** (i) By definition \( a_0 = s_0 \) and \( b_0 = [s_0]_v \). (ii) Follows from the lemmas above.

### 4 Symbolic Reachability Analysis

The theory \([A]\) is not directly executable in Maude, since the tick rules introduce a new variable \( T \) in their right-hand sides. Section 4.1 describes how the rewrite theory \([A]\) can be symbolically executed using Maude-with-SMT, and we prove in Section 4.2 that symbolic executions in \([A]\) correspond to transitions in the PZG of \( A \). 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” \[Mes20\] 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 \( \rightsquigarrow \). For example, we have the following symbolic rewrite in our running example:

\[
\phi \parallel [\text{idle} : X_1 ; X_2 ] < P_1 ; P_2 ; P_3 > \rightsquigarrow [ [A] ] \\
\phi \text{ and } T' \geq 0/1 \parallel [\text{idle} : X_1 + T' ; X_2 + T'] < P_1 ; P_2 ; P_3 >
\]

The SMT variables \( X_i \) (resp. \( P_i \)) represent the values of the clocks (resp. parameters). The variable \( T' \) is a fresh variable, of sort \text{Real}, created in the rewrite. This symbolic rewrite captures all the infinitely many delays that can take place when the automaton is in state \text{idle}.

Maude-SE allows us to solve symbolic reachability problems as illustrated in the following example.

Example 4. In the module \( \text{PTA-COFFEE} \), the command

\[
\text{smt-search [\text{idle} : X_1 ; X_2 ] < P_1 ; P_2 ; P_3 > \Rightarrow*} \\< \text{done} : X_1' ; X_2' > < P_1 ; P_2 ; P_3 > \\
\text{such that } (X_1 = = X_2 \text{ and } X_1 \geq 0/1 \text{ and } P_1 \geq 0/1 \text{ and } P_2 \geq 0/1 \text{ and } P_3 \geq 0/1) = \text{true}.
\]

uses a breadth-first search strategy to answer the following reachability question: are there values for the clocks and parameters such that the location \text{done} can be reached from the location \text{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 \( X_1 = = X_2 \) but unknown, and where parameters and clocks are all non-negative numbers. The first answer to this query includes the satisfiable constraint (syntax \text{where}) accumulated along the path from \text{idle} to \text{done}:

Solution 1
state: \( \langle \text{done} : #3-T ; #1-T + #2-T + #3-T \rangle \langle P_1 ; P_2 ; P_3 \rangle \)

where \( X_1 = = X_2 \) and \( X_1 \geq 0/1 \) and \( P_1 \geq 0/1 \) and \( P_2 \geq 0/1 \) and
\( \ldots \text{ and } #1-T: \text{Real} + #2-T: \text{Real} = = P_3 \text{ and } \)
\( \ldots \text{ and } #3-T: \text{Real} \leq 10/1 \) and \( #1-T: \text{Real} + #2-T: \text{Real} = = P_3 \)

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 \text{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 \text{add-sugar}, \text{preparing-coffee} and \text{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 \( \rightsquigarrow[A] \) is bisimilar to the PZG of \( A \). We start with a lemma establishing the correspondence between a zone \( Z \) and the (SMT) boolean expression \([Z]_b\). This is useful to later show that \((\ell, Z)\) is a valid reachable state in the PZG of \( A \) (\( Z \) cannot be empty) iff the boolean expression in the corresponding constrained term in \([A]_b\) is satisfiable (and hence, reachable via \( \rightsquigarrow[A] \)).

Lemma 4. For any zone \( Z \), \( Z \neq \emptyset \) iff \([Z]_b\) is satisfiable.

\textbf{Proof.} For the \((\Rightarrow)\) side, we know that there exists \( v \) and \( w \) s.t. \( v, w \models Z \) (making true all the inequalities in \( Z \)). By the definition of \([Z]_b\), we can show that the valuation \((v, w)\) is a witness for satisfiability of \([Z]_b\).

For the \((\Leftarrow)\) side, a witness for \([Z]_b\) must necessarily give values to all the clock variables and parameters. This witness is the needed valuations \( v, w \) showing that \( Z \) is not empty. \hfill \Box

Next we define operations on constrained terms corresponding to those on zones. We use \( \{\ell : e_1 ; \ldots ; e_n\} \) to denote either \([\ell : e_1 ; \ldots ; e_n]\) or \( < \ell : e_1 ; \ldots ; e_n > \). \( \mathcal{T}_Z \) is the set of terms of the form

\[
\phi \parallel \{\ell : e_1 ; \ldots ; e_n\} < P_1 ; \ldots ; P_m >
\]
where \( P_\ell \) are variables, \( e_i \) expressions (possibly containing SMT variables) and \( \phi \) must contain at least one inequality for each variable occurring after the symbol \( \parallel \) (e.g. \#1-T \( \geq 0/1 \)). We use \( U \) and \( V \) to range over elements in \( T_Z \).

**Definition 2.** Given \( R \subseteq X \), we use \( e_i^R \) to denote the expression \( 0/1 \) if \( x_i \in R \) and \( e_i \) if \( x_i \notin R \). Let \( U = \phi \parallel \{ \ell : e_1; \ldots ; e_n \} < P_1; \ldots ; P_m > \). We define the following operations on \( T_Z - \) terms:

- **Reset:** \( U[\ell] \overset{\text{def}}{=} \phi \parallel \{ \ell_1 : e_1^R; \ldots ; e_n^R \} < P_1; \ldots ; P_m > \)
- **Time Elapse:** \( U^\ell \overset{\text{def}}{=} (\phi \parallel T \geq 0/1) \parallel \{ \ell : e_1 + T; \ldots ; e_n + T \} < P_1; \ldots ; P_m > \) where \( T \) is a fresh variable (not occurring in \( \phi \)).
- **Conjunction:** Let \( G \) be a boolean expression such that \( \text{var}(G) \subseteq \text{var}(U) \). Then,
  \[
  U \land G \overset{\text{def}}{=} (\phi \parallel \psi) \parallel \{ \ell : e_1; \ldots ; e_n \} < P_1; \ldots ; P_m > .
  \]
- **Instantiation:** Given a clock valuation \( w \) and a parameter valuation \( v \),
  \[
  U\{v,w\} \overset{\text{def}}{=} (\phi \parallel \psi) \parallel \{ \ell : e_1; \ldots ; e_n \} < P_1; \ldots ; P_m > .
  \]
  where \( \psi \) is the boolean expression
  \[
  e_1 \overset{\text{def}}{=} [w(x_1)]_e \quad \text{and} \quad \ldots \quad e_n \overset{\text{def}}{=} [w(x_n)]_e \quad \text{and} \quad P_1 \overset{\text{def}}{=} [v(p_1)]_e \quad \text{and} \quad \ldots \quad P_m \overset{\text{def}}{=} [v(p_m)]_e .
  \]

\( U\{v,w\} \) equates \( \text{def} \) the expressions of the clocks with the values given by \( w \) (similarly for the parameters). Hence, \( U\{v,w\} \) agrees with the values assigned by \( v \) and \( w \).

**Definition 3** (Relation \( \sim \)). Define \( \sim \subseteq (L \times C) \times T_Z \) as follows:

\[
(\ell, Z) \sim U = \phi \parallel \ell : e_1; \ldots ; e_n > < P_1; \ldots ; P_m >
\]

whenever for all \( v \) and \( w \), we have \((v,w) \models Z\) iff the boolean expression in \( U\{v,w\} \) is satisfiable.

Intuitively, a state \((\ell, Z)\) 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 \( \phi \) in \( U \) true.

The following lemmas show that the operations on zones agree with those in Definition 2.

**Lemma 5** (Reset). Let \( R \subseteq X \), \( Z \neq \emptyset \), and assume that \((\ell, Z) \sim U\) where

\[
U = \phi \parallel \ell : e_1; \ldots ; e_n > < P_1; \ldots ; P_m >
\]

Then, \((\ell, Z[R]) \sim U[R]\).

**Proof.** Since \((\ell, Z) \sim U\), for all \( v, w \), the constraint in \( U\{v,w\} \) is satisfiable iff \( v, w \models Z \). If \( x_i \in R \), then \( e_i^R = 0/1 \). Consider a clock valuation \( w' \) s.t. \( v, w' \models Z[R] \). It must be the case that \( w' = w[R] \) and \( w'(x_i) = 0 \) for all \( x_i \in R \). Since the constraint \( \phi \land \psi \) (see Definition 2) in \( U\{v,w\} \) is satisfiable, so it is \( \phi \) \land \psi \) and \( 0/1 \overset{\text{def}}{=} 0/1 \). On the other side, if \( x_i \in R \), \( 0/1 = [w'(x_i)]_e \) is satisfiable iff \( w'(x_i) = 0 \).

**Lemma 6** (Time Elapse). Let \( Z \neq \emptyset \), and assume that \((\ell, Z) \sim U\) where

\[
U = \phi \parallel \ell : e_1; \ldots ; e_n > < P_1; \ldots ; P_m >
\]

Then, \((\ell, Z') \sim U'\).
Proof. By definition, \( v, w + T \models Z \) whenever \( v, w \models Z \) (and \( T \geq 0 \)). The result follows by noticing that the expression \( e_i \) in \( U \) is replaced by \( e_i + T \) in \( U' \) (and the inequality \( T \geq 0 \) is added to the constraint). \( \square \)

Lemma 7 (Conjunction). Let \( G \) be a guard or an invariant, \( Z \neq \emptyset \), \( U = \phi \parallel < \ell : e_1; \ldots; e_n > < P_1; \ldots; P_m > \) and assume that \((\ell, Z) \sim U\). Then, \((\ell, Z \cap G) \sim U \land ([G]_b[X_i/e_i])\).

Recall that the relation \( \Rightarrow \) on the PZGs captures, in one step, a discrete transition followed by a delay transition. Hence, a state \((\ell, Z)\) is ready to perform a discrete transition leading to \((\ell', Z')\) where \( Z' = (\{Z \cap g\}[R] \cap I(\ell')) \cap I(\ell) \) (if \( Z' \) is not empty). Let \( \sim_{[A]} \) be the application of a \( \ell \)-\( \sigma \) rule followed by a tick rule, and let \( \sim_{[A]} \) be its reflexive and transitive closure. The following Theorem shows that \( \Rightarrow \) on the PZG is bisimilar to \( \sim_{[A]} \) on constrained terms.

Theorem 2. Let \( A \) be a PTA. \( \sim \) is a bisimulation between the transition systems \((C, s_0, \Rightarrow)\) and \((\mathcal{T}_Z, \phi_0 \parallel t_0, \sim_{[A]} \) where \( t_0 = < \ell_0 : T; \ldots; T < P_1; \ldots; P_m > \) and \( \phi_0 = (P_1 \geq 0/1 \text{ and } T \geq 0/1 \text{ and } \lceil I(t_0) \rceil_b[X_i/T])\).

Proof. Let \( U = \phi \parallel t \) be a term in \( \mathcal{T}_Z \). If \( t \) is of sort \( \text{NState} \), \( U^\dagger \) denotes the corresponding term \( \phi \parallel t' \) where \( t' \) is of sort \( \text{DState} \) (changing \( \{\ell : e_1; \ldots; e_n\} \) to \( < \ell : e_1; \ldots; e_n > \)). Similarly, if \( t \) is of sort \( \text{DState} \), \( U^\dagger \) denotes the corresponding term \( \phi \parallel t' \) where \( t' \) is of sort \( \text{NState} \).

Let \( U = \phi \parallel \{\ell : e_1; \ldots; e_n\}/(P_1; \ldots; P_m) \) and \((\ell, Z) \sim U \). The result follows from lemmas 5, 6, and 7 and noticing that \( U \sim_{[A]} \left(( U \land [\bar{g}]_b[R] \land I(\ell')) \right) \sim_{[A]} \left(( U \land [\bar{g}]_b[R] \land I(\ell')) \right) \land I(\ell') \) where, \( \bar{g} = [\bar{g}]_b[e_i/X_i] \), and similarly for the invariants. Note also that Lemma 4 guarantees that a given transition is enabled in \((\ell, Z)\) iff it is also enabled at \( U \) whenever \((\ell, Z) \sim U \). \( \square \)

## 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 \texttt{smt-search} of the corresponding \([A]\) is infinite. The problem is that the search command stops exploring from a symbolic state only if it has already visited the same state. In many cases, due to the fresh variables created, symbolic states representing the same set of concrete states are not the same, even though they are logically equivalent, as exemplified below.

Example 5. Consider the automaton in Fig. 9a. After each iteration \( i \) between \( \ell_0 \) and \( \ell_1 \), we have \( y - x \geq 5i \) and hence, infinitely many different symbolic states. After 10 iterations, the constraint \( y \geq 50 \) is satisfiable and the state \texttt{bad} can be reached. In this case, the command

\texttt{smt-search [10 : 0;1/0;1]< => * \texttt{bad} : X';Y'>< >}

finds a solution to this reachability problem.

Now consider the automaton in Fig. 9b, where the location \texttt{bad} cannot be reached. The execution of the command \texttt{smt-search [10 : 0/1]< => * \texttt{bad} : X'>< >} does not terminate, since the following symbolic states (omitting some details for readability) appear while exploring the state space:

\[
\begin{align*}
s_0 : & \ 0/1 \leq \#0-T \parallel (\ell_0 : \#0-T) \\
\ell_1 : & \ \phi_0 \text{ and } 5/1 \leq \#0-T \text{ and } 10/1 \geq \#0-T + \#1-T \parallel (\ell_1 : \#0-T + \#1-T) \\
\ell_2 : & \ \phi_1 \text{ and } 0/1 \leq \#2-T \parallel (\ell_0 : \#2-T) \\
\ell_3 : & \ \phi_2 \text{ and } 5/1 \leq \#2-T \text{ and } 10/1 \geq \#2-T + \#3-T \parallel (\ell_1 : \#2-T + \#3-T) \\
\end{align*}
\]

where \( \phi_i \) is the constraint in the state \( s_i \). Note that \([s_0] = [s_2] = [s_3]\) (i.e. \( s_0 \) represents the same set of concrete states as \( s_2 \)). However, the constrained term \( s_0 \) is not equivalent to \( s_2 \) and the \texttt{smt-search} command keeps exploring the successor states of \( s_2 \). Note also that, due to the definition of \( \sim \), the constraints are always accumulated. For instance, \( \phi_2 \) 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 [Mes20]. Essentially, we stop searching from a symbolic state if, during the search, we have already encountered another state that subsumes it. More precisely, let \( U = \phi_u \parallel t_u \) and \( V = \phi_v \parallel t_v \). We define \( U \sqsubseteq V \), meaning that \( U \) is less general than \( V \), if there is a substitution \( \theta \) making \( t_u \) and \( t_v \theta \) equal and the implication \( \phi_u \Rightarrow \phi_v \theta \) holds. In that case, \([U] \subseteq [V] \) [Mes20]. Hence, during the search procedure, if a term \( U \) is reached and some term \( V \) has already been visited s.t. \( U \sqsubseteq V \), the state \( U \) will not be further explored. It is known that such reachability analysis with folding is sound and generates no spurious counterexample [BEM13].

The syntax of the implemented command is

\[
\text{red reachability}(\phi \parallel t, \ell, \text{bound}) .
\]

The second and third parameters are optional. This command computes all the reachable symbolic states, using folding, starting from \( \phi \parallel t \) until either: (1) no new states can be reached; (2) the location \( \ell \) is reached; or (3) the search exceeds the depth \( \text{bound} \).

We could quickly implement a prototype of our new symbolic reachability analysis algorithm using Maude’s meta-programming features. For instance, the function \text{metaMatch} \ applied to two terms \( U \) and \( V \) returns the set of substitutions \( \theta \) such that \( U \) equals \( V \theta \), and the function \text{metaCheck} \ can be used to delegate to the SMT solver the task of checking whether the formula \( \neg(\phi_u \Rightarrow \phi_v \theta) \) is unsatisfiable (and hence, the implication valid). Details about the implementation can be found in the companion repository [ABO+22].

**Example 6.** For the automaton in Fig. 3b, the command

\[
\text{red reachability}(X > 0/1 || < 10 : X > < >) .
\]

computes the set of reachable states starting from location \( \ell_0 \), with any non-negative initial clock value. The result is:

- \( X > 0/1 || < 10 : X > < > ; \quad \text{--- State 1} \)
- \( X > 0/1 \text{ and } 5/1 \leq X \text{ and } 10/1 \Rightarrow X \text{ and } #1-T \Rightarrow 0/1 \text{ and } 10/1 \Rightarrow X + #1-T \text{ \text{and} } < 11 : X + #1-T > < > \quad \text{--- State 2} \)

In contrast to \text{smt-search}, the above command terminates and shows that there are only two (distinct) reachable symbolic states, and that the location \text{bad} is not reachable.

Formally, the command \text{reachable} \ computes, iteratively, the folding reachable transition system \( (T_{\mathbb{Z}}^f, U_0, 2f_{[A]}^f) \) [EM07] \text{BM14} \ where \( T_{\mathbb{Z}}^f \) is defined as \( \bigcup_{i\in\mathbb{N}} S_i \), \( S_0 = \{U_0\} \) and \( S_{n+1} = \{U \mid \exists V \in U_n \text{ s.t. } V \overset{2f_{[A]}^f}}{\rightarrow} \}

\( U \) and \( U \not\sqsubseteq V' \) for any \( V' \in S_{k\leq n} \); and \( 2f_{[A]}^f = \bigcup_{n\in\mathbb{N}} (-\rightarrow_n) \) where \( -\rightarrow_0 = \emptyset \) and \( -\rightarrow_{n+1} = \{(U,V) \in S_n \times \bigcup_{0\leq i\leq n+1} S_i \mid \exists V'.U \overset{2f_{[A]}^f}}{\rightarrow} V' \text{ and } V' \sqsubseteq V\} \).

**Theorem 3** (Termination). If the PZG \((C, s_0, \Rightarrow)\) of a PTA \( A \) is a finite transition system, then \( (T_{\mathbb{Z}}^f, U_0, 2f_{[A]}^f) \) is also a finite transition system.

**Proof.** Suppose, to obtain a contradiction, that the transition system \( (T_{\mathbb{Z}}^f, U_0, 2f_{[A]}^f) \) has infinitely many
Our executable rewriting-modulo-SMT semantics for PTAs gives us the possibility of applying different
5 Parameter Synthesis and Analysis
PZG is infinite.
the same part of the PZG. However, Imitator also uses heuristics that may synthesize parameters even if the
it terminates when Imitator terminates with default settings since they both use subsumption, so generate
states. Since the number of locations is finite, there must be a cycle\footnote{ℓ₀ is not intended to be the initial location. The index 0 has been used to simplify the notation.} \((ℓ₀, Z₀) \Rightarrow^* (ℓ_{k-1}, Z_{k-1}) \Rightarrow (ℓ₀, Z₀)\) in
the PZG that is simulated (\(\sim\)) in the transition system of \([A]\) as
\[ V₀ \overset{2^*\sim\sim[A]}{\rightsquigarrow} Vₖ \overset{2^*\sim\sim[A]}{\rightsquigarrow} V₂ₖ \overset{2^*\sim\sim[A]}{\rightsquigarrow} \cdots \]
where: for all \(i \geq 0\), \((ℓ_{i \mod k}, Z_{i \mod k}) \sim Vᵢ\); and \(Vⱼ \not\sim Vᵢ\) for \(0 \leq i < j\). In each cycle, from \(Vᵢk\) to \(Vⱼk\), \(k\) new fresh variables are created. Assume that the clock expressions in \(V₀\) are \((x₀, x₁, \cdots, xₙ)\). Let \(\Deltaᵢ = Tᵢ₁ + \cdots + Tᵢₖ₋₁ + Tᵢₖ+₁\) be the sequence of \(k\) fresh variables created when moving from \(Vᵢₖ⁺₁\) to \(Vⱼk\). Moreover, for any of such sequences, let \(\Delta[p]\) be the suffix of \(\Delta\) with \(p\) elements and \(\Delta[−p]\) the prefix with \(k − p\) elements. As explained below, the position \(p\) will be the last position in the sequence where a given clock was reset. Since the relation \(\sim\) is deterministic, in the sense that rules always add the same constraint (using fresh variables when needed), the expression for a clock \(xᵢ\) when it is reset or not is, respectively:

<table>
<thead>
<tr>
<th>Iter/State</th>
<th>Reset</th>
<th>Non-reset</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ((V₀))</td>
<td>(xᵢ + T₀ⁱ)</td>
<td>(xᵢ + T₀ⁱ)</td>
<td>(G_{k-1}(k-1) \land I₀(k))</td>
</tr>
<tr>
<td>1 ((V₁))</td>
<td>(Δ₁²)</td>
<td>(xᵢ + T₀ⁱ + Δⁱ)</td>
<td>(G_{k-1}(k-1) \land I₀(k))</td>
</tr>
<tr>
<td>2 ((V₂))</td>
<td>(Δ²₀)</td>
<td>(xᵢ + T₀ⁱ + Δⁱ + Δ²)</td>
<td>(G_{k-1}(2k-1) \land I₀(2k))</td>
</tr>
<tr>
<td>(\cdots)</td>
<td>(\cdots)</td>
<td>(\cdots)</td>
<td>(\cdots)</td>
</tr>
</tbody>
</table>

where \(G_{k-1}((k-1))\) denotes the guard of the transition from location \(ℓ_{k-1}\) to \(ℓ₀\) evaluated using the clock expressions in the state \(Vₖ₋₁\). Similarly, \(I₀(k)\) is the invariant at location \(ℓ₀\) evaluated with the clock expressions in the state \(Vₖ\). Recall that the last element in \(\Deltaⁱ\) if \(T₀ᵢ\) and it corresponds to the variable created due to the delay in state \(Vₖ\).

Now we build a unifier \(θ\) between \(Vₖ\) and \(V₂ₖ\) as follows. If every clock is reset, at least once at some point in the sequence, we choose the clock expression with the smallest \(p\) and \(θ\) maps each variable in \(\Deltaⁱ[p]\) to the corresponding one in \(Δ²ⁱ[p]\). Note that such a \(θ\) unifies also the clock expressions with smaller suffixes (and bigger \(p\)). If none of the clocks is reset, then \(θ\) matches the last element of \(\Deltaⁱ\) with \(T₀ᵢ + \sum Δ²ᵢ\) (and the remaining variables as expected). This intuitively corresponds to “extending” the delay \(T₀ᵢ\) (at state \(Vₖ\)) with the sum of all the delays of the cycle, including the one in \(V₂ₖ\). Finally, in the case of reset and non-reset clocks, the variables in the reset clocks are unified as in the first case (choosing the smallest \(p\)), and for a non-reset clock \(xᵢ\), the subsequence \(xᵢ + T₀ⁱ + Δⁱ[−p]\) is unified with \(xᵢ + T₀ⁱ + Δⁱ + Δ²[−p]\) as in the second case, matching the last element \(Tᵢ\) in \(Δⁱ[−p]\) with \(Tᵢ + Δ²[−p]\).

Let \(ψ\) be a valuation for the variables in the constraint \(C₂ₖ\) (state \(V₂ₖ\)). From \(ψ\), we can build the corresponding parameter \(v\) and clock \(w\) valuation. Since both \(Vₖ\) and \(V₂ₖ\) are related to \((ℓ₀, Z₀)\) via \(\sim\), \(C₂ₖ\{v, w\}\) and \(Cₖ\{v, w\}\) both eval to true. Since \(G_{k-1}(2k-1)ψ\) evals to true, so does \(G_{k-1}(k-1)θψ\). Similarly for the invariants. Hence, \(ψ\) is also a valuation making true \(C₂ₖ\) and the implication \(C_k ⇒ C₂ₖθ\) holds. We have \(V₂ₖ \subseteq Vₖ\), thus a contradiction.

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 \texttt{smt-search} and \texttt{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 \texttt{add\_sugar} holds if the current location is \texttt{add\_sugar}) and inequalities on clocks and parameters (e.g. \( x_1 \neq x_2 \)).

**Definition 4.** Let \( A \) be a PTA and \( \phi \) a state predicate. The EF-emptiness problem asks: “is the set of parameter valuations \( v \) such that there exists a reachable state \((\ell, w)\) in \( v(A) \) satisfying \( \phi \) empty?” EF-synthesis is the problem of computing parameter valuations \( v \) such that a run of \( v(A) \) 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 \([\ell]_b = L == \ell\) (for a variable L of sort Location) to the definition of \([\ell]_b\). The command

\[\texttt{smt-search} \left[ 1 \right] \left[ \ell_0 : 0/1 ; \ldots ; 0/1 \right] < P_1 ; \ldots ; P_m > \Rightarrow \]
\[< L : \text{Location} > X' ; \ldots ; X_n > < P_1 ; \ldots ; P_m >\]
\[\text{such that} \ [\phi]_b \text{ and } P_1 >= 0/1 \text{ and } \ldots \text{ and } P_m >= 0/1 .\]

then tries to find a path from \( \ell_0 \) to an arbitrary location \( L \) satisfying \( \phi \). The resulting constraint, if any, is an answer to the synthesis problem \( \text{EF}\phi \). The command \texttt{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 \( \text{EF}\phi \) and then negating the resulting constraint.

**Example 7.** Let \( \phi \) be the output of the \texttt{smt-search} command in Example 4. Since \( \phi \) is satisfiable, there are values for the parameters such that \texttt{done} is reachable and the answer to the \( \text{EF}\)-emptiness problem \( \text{EF}(\text{done}) \) is false. The obtained constraint also gives us an answer to the corresponding \( \text{EF}\)-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 \texttt{ge} of the Z3 theorem prover to automate this step). This means that \texttt{done} is reachable whenever \( p_2 \leq p_3 \).

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

\[\texttt{smt-search} \left[ \text{idle} : 0/1 ; 0/1 \right] < P_1 ; P_2 ; P_3 > \Rightarrow\]
\[< L : X' \land X_2' > < P_1 ; P_2 ; P_3 >\]
\[\text{such that} \ (L == \text{preparing\_coffee} \land X' \neq X_2' \land \]
\[P_1 >= 0/1 \text{ and } P_2 >= 0/1 \text{ and } P_3 >= 0/1) \text{ is true} .\]

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 \( \text{EF}(x_1 > x_2) \). 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

\[\texttt{smt-search} \left[ 1,10 \right] \left[ \text{idle} : 0/1 ; 0/1 \right] < P_1 ; P_2 ; P_3 > \Rightarrow\]
\[< L : X' \land X_2' > < P_1 ; P_2 ; P_3 >\]
\[\text{such that} \ X'_1 \neq X_2' \text{ and } P_1 >= 0/1 \text{ and } P_2 >= 0/1 \text{ and } P_3 >= 0/1 .\]

searches for states where the first clock is strictly greater than the second one. Maude does not find any solution to this query in 10 steps. If we add the condition \( 2/1 \times p_1 > p_2 \), the number of symbolic states is finite (at most one dose of sugar is possible). \texttt{smt-search}, without depth bound, does not terminate. However, the new command \texttt{reachability} halts, finding 9 (different) symbolic states, with none of them satisfying \( X'_1 > X_2' \). Therefore, when \( 2 \times p_1 > p_2 \), there is no valuation reaching a state where \( x_1 > x_2 \). \( \square \)
It is worth remarking that Imitator only supports properties over locations but not over clocks. UPPAAL 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 \((T_\mathcal{Z}, \phi_0 \parallel t_0, s^t, l_\mathcal{A})\).

**Example 8 (Strategies).** The answer \(p_2 \leq p_3\) to the synthesis problem \(EF(\text{done})\) in Example 7 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 : X1; X2> < P1; P2; P3> --- Stop at done
  or-else --- in location add_sugar, add sugar if needed
  match C || < add_sugar : X1; X2> < P1; P2; P3> s.t.
  validity(C implies X1 == X2); add-sugar(N); with-sugar(N)
  or-else --- otherwise, apply any rule
  all ; with-sugar(N) .
--- Adding m doses of sugar
sd add-sugar(0) := 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(F)` uses the SMT solver to check whether the formula \(\neg F\) 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 P1 ::= 0/1 and P2 ::= 0/1 and P3 ::= 0/1 || [ idle : 0/1 ; 0/1 ] < P1 ; P2 ; P3 > using with-sugar(1) .
```

**Solution 1 result State:** [Details省略]

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 [AMvdP21]. 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. Table 1 describes the PTAs considered in Fig. 4 (the complete set of benchmarks can be found in [ABO*22]).
### Table 1: Benchmarks considered.

<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>

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.

### 7 Related Work

**Formal Analysis of Parametric Timed Automata.** Since most analysis and parameter synthesis problems are undecidable for PTAs [And19], approaches to address them have focused on heuristics. The state-of-the-art PTA tool, Imitator [And21], uses techniques such as subsumption [NPvdP18] and convex zone merging [AFS13] 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 [4]. 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.

There are very few parameter synthesis tools for PTAs. The algorithms described in [BBBC16] perform time-bounded model-checking. Roméo [LRST09] provides parameter synthesis for parametric timed Petri nets instead of PTAs.

**Rewriting Semantics for Timed Automata.** The paper [OM02] 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 [OM02], 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 [OM08, OM07] extension of Maude—has been applied to a wide range of real-time systems [Olv14] and has provided formal semantics and formal analysis to a number of modeling languages [Olv11]. 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 [OM06], 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 [LKB21] and soft agents [NT22]. 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, Őlveczky, Petrucci and Rømning 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).

References


