Rewriting Logic Semantics and Symbolic Analysis for Parametric Timed Automata

Jaime Arias, Kyungmin Bae, Carlos Olarte, Laure Petrucci, Peter Csaba Ölveczky, Fredrik Rømming

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 Rømming[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

*To appear in the proceedings of the 8th International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS'22).

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, KP12, 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$^+$07] are on the other side of the expressiveness spectrum, and support the above features. The Real-Time Maude tool [ÖM08, ÖM07] extends Maude to real-time systems and has been used to analyze a wide range of systems where the above features are needed. Such applications include state-of-the-art 50-page multicast and IETF protocols [ÖMT06, LÖ09], scheduling protocols with unbounded queues [ÖC06], state-of-the-art wireless sensor network protocols [ÖT09], MANET protocols [LÖM16], turning control algorithms for aircraft [BKMÖ15], human multitasking [BMÖ19], large cloud-based transaction systems [BGG$^+$18, GÖ14], and so on (see [Ölv14] for a dated overview). In particular, thanks to its expressiveness, Real-Time Maude has been applied as a semantic framework and formal analysis backend in which a number of modeling languages, such as (subsets of) Ptolemy II [BÖF$^+$12], AADL [ÖBM10], a language developed at DOCOMO Labs [ADY$^+$09], and others have been given a formal semantics and formal analysis capabilities [Ölv11].

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 [ÖM06]. 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, straight-forward 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 [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 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 [LÁÖ15]; 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 Uppaal. 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* $\mathcal{A}$ is a triple $(A, a_0, \to_{\mathcal{A}})$, where A is a set of *states*, $a_0 \in A$ is the *initial state*, and $\to_{\mathcal{A}} \subseteq A \times A$ is a *transition relation.* A function $h : A \to B$ is a *bisimulation* from $\mathcal{A}$ to $\mathcal{B}$ iff: (i) $h(a_0) = b_0$; and (ii) for each $a \in A$, if $a \to_{\mathcal{A}} a'$ then $h(a) \to_{\mathcal{B}} h(a')$, and if $h(a) \to_{\mathcal{B}} b$ then there exists $a'' \in A$ with $a \to_{\mathcal{A}} a''$ 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 $x_1 - x_2 \bowtie \mathit{plt}$, 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 $\mathcal{C}$ for the set of zones.

A *parametric timed automaton* (PTA) $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, L, \ell_0, X, P, I, E)$, where Σ 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. $I : L \to \mathcal{C}$ 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 $g \in \mathcal{C}$, action $\sigma \in \Sigma$, and clock reset $R \subseteq X$.

A parameter valuation is a function $v : P \to \mathbb{Q}_{\geq 0}$ and a clock valuation is a function $w : X \to \mathbb{R}_{\geq 0}$. 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 $\vec{0}$ for the clock valuation s.t. $\forall x \in X : \vec{0}(x) = 0$. We extend parameter valuations to linear terms. We write $v, w \models (x_i - x_j \bowtie \mathit{plt})$ iff $w(x_i) - w(x_j) \bowtie v(\mathit{plt})$, 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 $v(\mathcal{A})$ for the timed automaton (TA) obtained by replacing each parameter p in invariants and guards by $v(p)$. The *concrete semantics* of a PTA $\mathcal{A}$ is derived from that of the TA $v(\mathcal{A})$, and is defined as a timed transition system with states (ℓ, w), initial state $(\ell_0, \vec{0})$ (we assume that $\vec{0} \models I(\ell_0)$), and transitions $\to \; = \; \xrightarrow{d} ; \xrightarrow{e}$, where continuous time delay ($\xrightarrow{d}$) and discrete transitions ($\xrightarrow{e}$) are defined as

- If $d \in \mathbb{R}_{\geq 0}$ and $w + d \models I(\ell)$, then $(\ell, w) \xrightarrow{d} (\ell, w + d)$.
- If $e = (\ell, g, \sigma, R, \ell') \in E$ and $w \models g$ and $w[R] \models I(\ell')$ then $(\ell, w) \xrightarrow{e} (\ell', w[R])$.

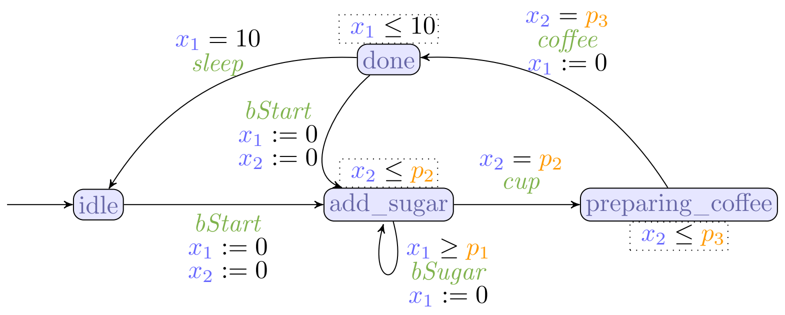


Figure 1: A coffee machine (*CM*) modeled as a PTA.

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: $Z^{\nearrow} \overset{\text{def}}{=} \{(v, w + d) \mid d \in \mathbb{R}_{\geq 0} \wedge v, w \models Z\}$

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 $\mathcal{A} = (\Sigma, L, \ell_0, X, P, I, E)$ is $(S, s_0, \Rightarrow)$, with $S \subseteq L \times \mathcal{C}$, initial state $s_0 = (\ell_0, (\bigwedge_{x \in X} x = 0)^{\nearrow} \cap I(\ell_0))$. A transition step $(\ell, Z) \Rightarrow (\ell', Z')$ exists if for some $(\ell, g, \sigma, R, \ell') \in E$ we have $Z' = ((Z \cap g)[R] \cap I(\ell'))^{\nearrow} \cap I(\ell') \neq \emptyset$. 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* Σ 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 $f : s_1 \times \cdots \times s_n \to s$, for $n \geq 0$. We denote by $T_{\Sigma,s}$ the set of ground (*i.e.* not containing variables) Σ-terms of sort s, and by $T_\Sigma(X)_s$ the set of Σ-terms of sort s over a set X of sorted variables. $T_\Sigma(X)$ and T_Σ denote all terms and ground terms, respectively.

A substitution $\theta : X \to T_\Sigma(X)$ 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 Σ is an order-sorted signature and E is a set of (conditional) equations of the form $t = t'$ **if** ψ, where $t, t' \in T_\Sigma(X)_s$ for some sort $s \in \Sigma$ and ψ is a conjunction of equations. We write $u =_E u'$ iff $(\Sigma, E) \vdash (\forall X)\ u = u'$ [Mes97].

A *rewrite theory* [Mes92] is a tuple $\mathcal{R} = (\Sigma, E, L, R)$, where (Σ, E) is an equational theory, L is a set of *labels*, and R is a set of labeled (conditional) rewrite rules of the form $l : q \longrightarrow r$ **if** ψ, where $l \in L$, $q, r \in T_\Sigma(X)_s$ for some sort $s \in \Sigma$, and ψ is a conjunction of equations and rewrites.

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

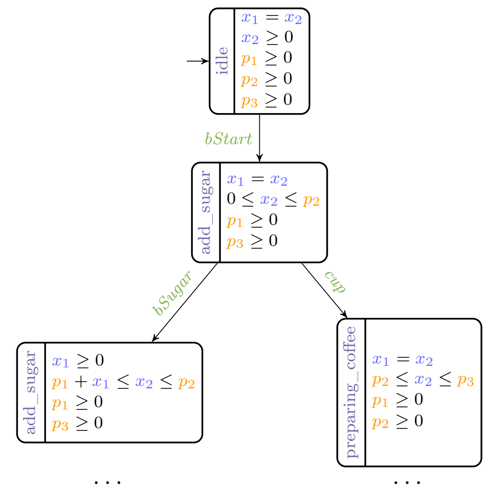


Figure 2: The PZG of the coffee machine example *CM*.

A rewrite theory $\mathcal{R}$ is called *topmost* iff there is a sort *State* at the top of one of the connected components of the poset $(S, \leq)$ such that for each rule $l : q \longrightarrow r$ **if** ψ, both q and r have the top sort *State*, and no operator has sort *State* or any of its subsorts as an argument sort.

Rewriting with SMT A *built-in theory* $\mathcal{E}_0$ of (Σ, E) is a first-order theory with a signature $\Sigma_0 \subseteq \Sigma$, where each sort s in Σ_0 is minimal in Σ and for each operator $f : w \longrightarrow s$ in $\Sigma \setminus \Sigma_0$, $s \notin \Sigma_0$ and f has no other subsort-overloaded typing in Σ_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 (Σ, E): for $t_1, t_2 \in T_{\Sigma_0}$, if $t_1 =_E t_2$, then $\mathcal{T}_{\mathcal{E}_0} \models t_1 = t_2$ [RMM17].

A *constrained term* is a pair $\phi \parallel t$ of a constraint ϕ 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 $\mathcal{E}_0$ [RMM17, BR19]. A constrained term $\phi \parallel t$ *symbolically* represents all instances of the pattern t such that ϕ 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 $l : q \longrightarrow r$ **if** ψ, extra variables not occurring in the left-hand side q are in X_0, and ψ is a constraint in a built-in theory $\mathcal{E}_0$. Then, a *one-step symbolic rewrite* $\phi \parallel t \rightsquigarrow_{\mathcal{R}} \phi' \parallel t'$ holds iff there exist a rule $l : q \longrightarrow r$ **if** ψ and a substitution $\theta : X \longrightarrow T_\Sigma(X_0)$ such that (1) $t =_E q\theta$, (2) $t' =_E r\theta$, (3) $\mathcal{T}_{\mathcal{E}_0} \models (\phi \wedge \psi\theta) \Leftrightarrow \phi'$, and (4) ϕ' is $\mathcal{T}_{\mathcal{E}_0}$-satisfiable. We denote by $\rightsquigarrow^*_{\mathcal{R}}$ the reflexive-transitive closure of $\rightsquigarrow_{\mathcal{R}}$.

If $\phi_t \parallel t \rightsquigarrow^* \phi_u \parallel u$ is a symbolic rewrite, then there exists a "concrete" rewrite $t' \longrightarrow^* u'$ with $t' \in [\![\phi_t \parallel t]\!]$ and $u' \in [\![\phi_u \parallel u]\!]$. Conversely, for any concrete rewrite $t' \longrightarrow^* u'$ with $t' \in [\![\phi_t \parallel t]\!]$, there exists a symbolic rewrite $\phi_t \parallel t \rightsquigarrow^* \phi_u \parallel u$ with $u' \in [\![\phi_u \parallel u]\!]$.

Maude. Maude [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:

```
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 [l] : 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 Φ .
```

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 Φ in $\mathcal{E}_0$. More precisely, it searches for a constrained term $\phi_u \parallel u$ such that $\mathit{true} \parallel t \rightsquigarrow^* \phi_u \parallel u$ and for some $\theta : X \longrightarrow T_\Sigma(X)$, $u =_E t'\theta$ and $\mathcal{T}_{\mathcal{E}_0} \models \phi_u \Rightarrow \Phi\theta$. The parameters `[n,m]` 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* $\overline{M}$ of sort `Module` (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 [YB20] extends Maude with additional functionality for rewriting modulo SMT, including witness generation for `smt-search`. It uses two theory transformations to implement symbolic rewriting [RMM17]. In essence, a rewrite rule $l : q \longrightarrow r$ **if** ψ is transformed into a constrained-term rule

$$l : \texttt{PHI} \parallel q \longrightarrow (\texttt{PHI}\ \mathit{and}\ \psi) \parallel r \ \textbf{if}\ \texttt{smtCheck}(\texttt{PHI}\ \mathit{and}\ \psi)$$

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 $(\mathit{var}(r) \cup \mathit{var}(\psi)) \setminus \mathit{var}(q)$ are considered constants.

Strategy Language. Maude's *strategy language* [CDE+22] 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 $(\alpha \mid \beta)$, iteration (α^*), and α `or-else` β, which executes β if α 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 $(\Sigma, L, \ell_0, X, P, I, E)$ with $n = |X|$ clocks $(x_1, \ldots, x_n)$, $m = |P|$ parameters $(p_1, \ldots, p_m)$, and $k = |L|$ locations $\{\ell_1, \cdots, \ell_k\}$. The idea is to represent a concrete state (ℓ, w) of the PTA $\mathcal{A}$ as a Maude term

```
[ ℓ : w(xi) ; ... ; w(xn) ] < P1 ; ... ; Pm >
```

where the `P`i are variables. A state (ℓ, 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

```
[ ℓ : w(xi) ; ... ; w(xn) ] < v(p1) ; ... ; v(pm) >.
```

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

```
< ℓ : w(xi) ; ... ; w(xn) > <  P1 ; ... ; Pm  >
```
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 <= P2) = 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 add_sugar is

```
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] .
```

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 of `State`, `NState` ("Non-delayed state") and `DState` ("delayed state"), as follows:

```
pr REAL .                    --- SMT rational/real numbers
sorts State NState DState Location . --- Sorts for states
subsorts NState DState < State .
--- Constants for locations
ops ℓ1···ℓk : -> Location .
--- States of the system
op <_:_;...;_> <_;...;_> : Location Real ... Real -> DState  .
```
$\underbrace{\texttt{Real}\ \ldots\ \texttt{Real}}_{n+m}$
```
op [_:_;...;_] <_;...;_> : Location Real ... Real -> NState  .
```
$\underbrace{\texttt{Real}\ \ldots\ \texttt{Real}}_{n+m}$

$[\![\mathcal{A}]\!]$ defines SMT-variables to represent clock valuations, parameter valuations, and time elapse:

```
vars X1 ... Xn    : Real .    --- Clock valuations
vars P1 ... Pm    : Real .    --- Parameter valuations
var  T            : Real .    --- Time elapse
```

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

$$[\![true]\!]_b = \texttt{true} \qquad [\![b_1 \wedge b_2]\!]_b = [\![b_1]\!]_b \ \texttt{and}\ [\![b_2]\!]_b$$

and for each inequality relation in $\{\geq, \leq, =, >, <\}$, we have, *e.g.*: $[\![e_1 \leq e_2]\!]_b = [\![e_1]\!]_e$ `<=` $[\![e_2]\!]_e$ and $[\![e_1 = e_2]\!]_b = [\![e_1]\!]_e$ `===` $[\![e_2]\!]_e$. For arithmetic expressions, we define:

$$\begin{array}{llll}
[\![e_1 + e_2]\!]_e = [\![e_1]\!]_e \ \texttt{+}\ [\![e_2]\!]_e & \qquad [\![x_i^R]\!]_e = \texttt{X}i \ \textit{if}\ x_i \notin R \\
[\![e_1 - e_2]\!]_e = [\![e_1]\!]_e \ \texttt{-}\ [\![e_2]\!]_e & \qquad [\![x_i]\!]_e \ = \texttt{X}i \\
[\![e_1 e_2]\!]_e \quad = [\![e_1]\!]_e \ \texttt{*}\ [\![e_2]\!]_e & \qquad [\![x_i^d]\!]_e \ = \texttt{X}i\ \texttt{+ T} \\
[\![p/q]\!]_e \quad\ = \texttt{p/q}\ \textit{if}\ p, q \in \mathbb{N} & \qquad [\![p_i]\!]_e \ = \texttt{P}i \\
[\![x_i^R]\!]_e \quad\ = \texttt{0/1}\ \textit{if}\ x_i \in R &
\end{array}$$

$[\![\_]\!]$ maps each transition $(\ell, g, \sigma, R, \ell') \in E$, to the following conditional rewrite rule ℓ-σ:

```
crl [ℓ-σ] : < ℓ  : X1      ; ... ; Xn      > < P1 ; ... ; Pm >
       =>   [ ℓ' : [[x1^R]]e ; ... ; [[xn^R]]e ] < P1 ; ... ; Pm >
            if [[g ∧ I(ℓ')[xi/xi^R]]]b = true .
```

where $I(\ell')[x_i/x_i^R]$ denotes substituting x_i^R for x_i in the expression $I(\ell')$ for each i. Furthermore, for each $\ell \in L$, $[\![\_]\!]$ adds a conditional rewrite rule ℓ`-tick`:

```
crl [ℓ-tick] : [ ℓ : X1     ; ... ; Xn     ] < P1 ; ... ; Pm >
```

```
       =>   < ℓ : X1 + T ; ... ; Xn + T > < P1 ; ... ; Pm >
    if ($[\![I(\ell)[x_i/x_i^d]]\!]_b$ and T >= 0/1) = true [nonexec] .
```

Example 3. *$[\![\_]\!]$ transforms the PTA CM in Fig. 1 to the rewrite theory $[\![CM]\!]$ below.*

```
mod PTA-COFFEE is
    pr REAL .
    sorts State NState DState Location .
    subsorts 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 ; 0/1 ] < P1 ; 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 T >= 0/1) = true [nonexec] .
endm
```

3.2 Correctness of the $[\![\_]\!]$ Transformation

In this section we relate $\mathcal{A}$ and $[\![\mathcal{A}]\!]$ by a bisimulation. Since a transition in $\mathcal{A}$ consists of a delay followed by a discrete transition, we define a corresponding rewrite relation $\mapsto_{[\![\mathcal{A}]\!]}$ combining a "tick" rule application with a "transition" rule application. We then show that these respective relations in the concrete semantics of $v(\mathcal{A})$ and in $[\![\mathcal{A}]\!]$ are bisimilar w.r.t. initial state `[`ℓ_0 `: 0/1 ; ... ; 0/1] <` $v(p_1)$ `; ... ;` $v(p_m)$ `>`.

Definition 1. *Let $\mathcal{A} = (\Sigma, L, \ell_0, X, P, I, E)$ be a PTA and t_1, t_2, t_3 be terms of $[\![\mathcal{A}]\!]$. We write $t_1 \mapsto_{[\![\mathcal{A}]\!]} t_3$ if there exists a t_2 such that $t_1 \longrightarrow t_2$ is a one-step rewrite applying an ℓ-`tick` rule in $[\![\mathcal{A}]\!]$ for some $\ell \in L$ and $t_2 \longrightarrow t_3$ is a one-step rewrite applying an ℓ -σ rule of $[\![\mathcal{A}]\!]$ for some $\ell \in L$ and $\sigma \in \Sigma$. Furthermore, we write $t_1 \mapsto^*_{[\![\mathcal{A}]\!]} t_2$ to indicate that there exists a sequence of $\mapsto_{[\![\mathcal{A}]\!]}$ rewrites from t_1 to t_2.*

Let v be a parameter valuation. $(T_{\Sigma,\mathtt{NState}})_v$ 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 $v(\mathcal{A})$. We define a map $[\![\_]\!]_v : S \to (T_{\Sigma,\mathtt{NState}})_v$, relating concrete states in $\mathcal{A}$ to states (of sort `NState`) in $[\![\mathcal{A}]\!]$, where for all concrete states $(\ell, w) \in S$,

$$[\![(\ell, w)]\!]_v = \texttt{[}\, \ell : [\![w(x_1)]\!]_e \texttt{ ; } \ldots \texttt{ ; } [\![w(x_n)]\!]_e \,\texttt{]}\, \texttt{<} [\![v(p_1)]\!]_e \texttt{ ; } \ldots \texttt{ ; } [\![v(p_m)]\!]_e \,\texttt{>}.$$

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,*

$$v, w' \models Z \quad \textit{iff} \quad [\![Z[x_i/x'_i]]\!]_b\theta \textit{ holds}$$

in the following cases:

1. *$w' = w$, $x'_i = x_i$, and*
   $\theta = \{\mathtt{Xi} \mapsto [\![w(x_i)]\!]_e, \mathtt{Pi} \mapsto [\![v(p_i)]\!]_e\}$.
2. *$w' = (w + d)$, $d \in \mathbb{R}_{\geq 0}$, $x'_i = x^d_i$, and*
   $\theta = \{\mathtt{Xi} \mapsto [\![w(x_i)]\!]_e, \mathtt{Pi} \mapsto [\![v(p_i)]\!]_e, \mathtt{T} \mapsto [\![d]\!]_e\}$.
3. *$w' = w[R]$, $R \subseteq X$, $x'_i = x^R_i$, and*
   $\theta = \{\mathtt{Xi} \mapsto [\![w(x_i)]\!]_e, \mathtt{Pi} \mapsto [\![v(p_i)]\!]_e\}$.

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, θ:

  1. $w(x_i) - w(x_j) \geq \sum_i(\alpha_i v(p_i)) + \beta$ holds iff $[\![Z]\!]_e\theta = [\![w(x_i)]\!]_e$ `-` $[\![w(x_j)]\!]_e$ `>=` $[\![\alpha_0]\!]_e$ `*` $[\![v(p_0)]\!]_e$ `+ ... +` $[\![\alpha_m]\!]_e$ `*` $[\![v(p_m)]\!]_e$ `+` $[\![\beta]\!]_e$ holds.

  2. $w(x_i) + d - w(x_j) + d \geq \sum_i(\alpha_i v(p_i)) + \beta$ holds iff $[\![Z[x_i/x^d_i]]\!]_e\theta =$
  $[\![w(x_i)]\!]_e$ `+` $[\![d]\!]_e$ `-` $[\![w(x_j)]\!]_e$ `+` $[\![d]\!]_e$
  `>=` $[\![\alpha_0]\!]_e$ `*` $[\![v(p_0)]\!]_e$ `+ ... +` $[\![\alpha_m]\!]_e$ `*`
  $[\![v(p_m)]\!]_e$ `+` $[\![\beta]\!]_e$

  3. $w[R](x_i) - w[R](x_j) \geq \sum_i(\alpha_i v(p_i)) + \beta$ holds iff $[\![Z[x_i/x^R_i]]\!]_e\theta = [\![w(x_i) \text{ if } x_i \in R \text{ else } 0]\!]_e$ `-` $[\![w(x_j) \text{ if } x_j \in R \text{ else } 0]\!]_e$ `>=` $[\![\alpha_0]\!]_e$ `*` $[\![v(p_0)]\!]_e$ `+ ... +` $[\![\alpha_m]\!]_e$ `*` $[\![v(p_m)]\!]_e$ `+` $[\![\beta]\!]_e$

  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 $Z = Z_1 \wedge Z_2$. By definition $v, w' \models Z_1 \wedge Z_2$ 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 $[\![Z_i[x_i/x'_i]]\!]_b\theta$ holds (i.e., evaluates to `true`) for $i \in \{1, 2\}$. Since $[\![(Z_1 \wedge Z_2)[x_i/x'_i]]\!]_b\theta$ equals $[\![Z_1[x_i/x'_i] \wedge Z_2[x_i/x'_i]]\!]_b\theta$, by definition of $[\![\_]\!]_b$, this is equal to $([\![Z_1[x_i/x'_i]]\!]_b$ `and` $[\![Z_2[x_i/x'_i]]\!]_b)\theta$, which again, since a substitution is a homomorphic extension of a variable substitution, equals $[\![Z_1[x_i/x'_i]]\!]_b\theta$ `and` $[\![Z_2[x_i/x'_i]]\!]_b\theta$, which by the induction hypotheses equals `true and true`, which again is `true`.

□

Lemma 2. *Let $\mathcal{A}$ be a PTA, v a parameter valuation, and (ℓ, w) and (ℓ', w') concrete states of $v(\mathcal{A})$. If $(\ell, w) \longrightarrow (\ell', w')$, then $[\![(\ell, w)]\!]_v \mapsto_{[\![\mathcal{A}]\!]} [\![(\ell', w')]\!]_v$.*

Proof. By definition, $((\ell, w), \sigma, (\ell', w')) \in \longrightarrow$ 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 $t = [\![(\ell, w)]\!]_v =$

`[` ℓ `:` $[\![w(x_1)]\!]_e$ `; ... ;` $[\![w(x_n)]\!]_e$ `] <` $[\![v(p_1)]\!]_e$ `; ... ;` $[\![v(p_m)]\!]_e$ `>.`

We can apply the rule ℓ`-tick` to t to get $u =$

`<` ℓ `:` $[\![w(x_1)]\!]_e$ `+` $[\![d]\!]_e$ `; ... ;` $[\![w(x_n)]\!]_e$ `+` $[\![d]\!]_e$ `> <` $[\![v(p_1)]\!]_e$ `; ... ;` $[\![v(p_m)]\!]_e$ `>`

since there exists a substitution $\theta = \{\mathtt{X}i \mapsto [\![w(x_i)]\!]_e, \mathtt{P}i \mapsto [\![v(p_i)]\!]_e, \mathtt{T} \mapsto [\![d]\!]_e\}$, such that (a) $t =_E l\theta$, $u =_E r\theta$, and (b) $\mathcal{T}_{\mathcal{E}_0} \models \phi\theta$, where l, r, ϕ are the left-hand side, right-hand side, and condition of rule ℓ-tick respectively. Note that (b) holds by Lemma 1 because $\phi\theta = ([\![I(\ell)[x_i/x_i^d]]\!]_b$ `and T >= 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 $\mathcal{A}$ such that $v, w'' \models g$ and $v, w' \models I(\ell')$. We can apply the rule l-σ to the above u to get $u_2 =$

`[` ℓ' `: (`$[\![w(x_1)]\!]_e$ `+` $[\![d]\!]_e$`) if` $x_1 \in R$ `else 0/1 ; ... ; (`$[\![w(x_n)]\!]_e$ `+` $[\![d]\!]_e$`) if` $x_n \in R$ `else 0/1 ]`
`<` $[\![v(p_1)]\!]_e$ `; ... ;` $[\![v(p_m)]\!]_e$ `>`

since there exists a substitution $\theta = \{\mathtt{X}i \mapsto [\![w''(x_i)]\!]_e, \mathtt{P}i \mapsto [\![v(p_i)]\!]_e\}$, such that (a) $u =_E l\theta$, $y =_E r\theta$, and (b) $\mathcal{T}_{\mathcal{E}_0} \models \phi\theta$, where l, r, ϕ are the left-hand side, right-hand side, and condition of rule ℓ-σ respectively. Again, note that (b) holds by Lemma 1 because $\phi\theta = [\![g \wedge I(\ell')[x_i/x_i^R]]\!]_b\theta$ holds iff $v, w'' \models g$ and $v, w' \models I(\ell')$, which is true by assumption. □

Lemma 3. *Let $\mathcal{A}$ be a PTA, v a parameter valuation, (ℓ, w) a concrete state of $v(\mathcal{A})$, and $b \in (T_{\Sigma,\mathtt{NState}})_v$. If $[\![(\ell, w)]\!]_v \mapsto_{[\![\mathcal{A}]\!]} b$ then there exists a concrete state (ℓ', w') in $v(\mathcal{A})$ such that $(\ell, w) \longrightarrow (\ell', w')$ and $b = [\![(\ell', w')]\!]_v$.*

Proof. By Definition 1, $t_1 \mapsto_{[\![\mathcal{A}]\!]} t_3$ if there exists a t_2 such that $t_1 \longrightarrow t_2$ is a one-step sequential rewrite applying an ℓ`-tick` rule in $[\![\mathcal{A}]\!]$ for some $\ell \in L$ and $t_2 \longrightarrow t_3$ is a one-step sequential rewrite applying a ℓ-σ rule of $[\![\mathcal{A}]\!]$ for some $\sigma \in \Sigma$.
For the first step $\left(\xrightarrow{\ell\text{-}\mathtt{tick}}\right)$, by assumption there exists a substitution $\theta = \{\mathtt{X}i \mapsto [\![w(x_i)]\!]_e, \mathtt{P}i \mapsto [\![v(p_i)]\!]_e, \mathtt{T} \mapsto [\![d]\!]_e\}$, for some $d \in \mathbb{R}_+$, such that there exists a rewrite from $t = [\![(\ell, w)]\!]_v =$

`[` ℓ `:` $[\![w(x_1)]\!]_e$ `; ... ;` $[\![w(x_n)]\!]_e$ `] <` $[\![v(p_1)]\!]_e$ `; ... ;` $[\![v(p_m)]\!]_e$ `>`

to $u =$

`<` ℓ `:` $[\![w(x_1)]\!]_e$ `+` $[\![d]\!]_e$ `; ... ;` $[\![w(x_n)]\!]_e$ `+` $[\![d]\!]_e$ `> <` $[\![v(p_1)]\!]_e$ `; ... ;` $[\![v(p_m)]\!]_e$ `>`

for at least one $d \in \mathbb{R}_+$. For this rewrite to be enabled, the condition $([\![I(\ell)[x_i/x_i^d]]\!]_b$ `and T >= 0/1`$)\theta$ must hold. By Lemma 1, $([\![I(\ell)[x_i/x_i^d]]\!]_b$ `and T >= 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 $\mathcal{A}$ such that $(\ell, w) \xrightarrow{d} (\ell, w + d)$.
For the second step $\left(\xrightarrow{\sigma}\right)$, the term $u =$

`< ` ℓ ` : ` $[\![w(x_1)]\!]_e$ ` + ` $[\![d]\!]_e$ ` ; ... ; ` $[\![w(x_n)]\!]_e$ ` + ` $[\![d]\!]_e$ ` > < ` $[\![v(p_1)]\!]_e$ ` ; ... ; ` $[\![v(p_m)]\!]_e$ ` >`

above is rewritten to $u_2 =$

`[ ` ℓ' ` : (` $[\![w(x_1)]\!]_e$ ` + ` $[\![d]\!]_e$`) if ` $x_1 \in R$ ` else 0/1 ; ... ; (` $[\![w(x_n)]\!]_e$ ` + ` $[\![d]\!]_e$`) if ` $x_n \in R$ ` else 0/1 ]`
`< ` $[\![v(p_1)]\!]_e$ ` ; ... ; ` $[\![v(p_m)]\!]_e$ ` >`

using the rule ℓ-σ in $[\![\mathcal{A}]\!]$. By definition of $[\![\mathcal{A}]\!]$, there is a corresponding discrete transition $(\ell, g, \sigma, R, \ell')$ in $\mathcal{A}$. The matching substitution used in the rewrite is then $\theta = \{\texttt{X}i \mapsto [\![w''(x_i)]\!]_e, \texttt{P}i \mapsto [\![v(p_i)]\!]_e\}$. Since this rewrite took place, θ, the condition of the rule, $[\![g \wedge I(\ell')[x_i/x_i^R]]\!]_b\theta$, must hold. By Lemma 1, $\phi\theta = [\![(g \wedge I(\ell')[x_i/x_i^R])]\!]_b\theta$ holds iff $v,(w+d) \models g$ and $v,(w+d)[R] \models I(\ell')$. Therefore we can apply the discrete transition $(\ell, g, \sigma, R, \ell')$ in $\mathcal{A}$, to go from $(\ell, w+d)$ to $(\ell', (w+d)[R])$. Altogether, for any PTA-rewrite

`[ ` ℓ ` : ` $[\![w(x_1)]\!]_e$ ` ; ... ; ` $[\![w(x_n)]\!]_e$ ` ] < ` $[\![v(p_1)]\!]_e$ ` ; ... ; ` $[\![v(p_m)]\!]_e$ ` >`

$\xrightarrow{\ell\text{-}\texttt{tick}}$

`< ` ℓ ` : ` $[\![w(x_1)]\!]_e$ ` + ` $[\![d]\!]_e$ ` ; ... ; ` $[\![w(x_n)]\!]_e$ ` + ` $[\![d]\!]_e$ ` > < ` $[\![v(p_1)]\!]_e$ ` ; ... ; ` $[\![v(p_m)]\!]_e$ ` >`

$\xrightarrow{\ell\text{-}\sigma}$

`[ ` ℓ' ` : (` $[\![w(x_1)]\!]_e$ ` + ` $[\![d]\!]_e$`) if ` $x_1 \in R$ ` else 0/1 ; ... ; (` $[\![w(x_n)]\!]_e$ ` + ` $[\![d]\!]_e$`) if ` $x_n \in R$ ` else 0/1 ]`
`< ` $[\![v(p_1)]\!]_e$ ` ; ... ; ` $[\![v(p_m)]\!]_e$ ` >`

there is the corresponding PTA-transition

$$(\ell, w) \xrightarrow{d} (\ell, w+d) \xrightarrow{\sigma} (\ell', (w+d)[R])$$

where indeed $[\![(\ell, w)]\!]_v =$

`[ ` ℓ ` : ` $[\![w(x_1)]\!]_e$ ` ; ... ; ` $[\![w(x_n)]\!]_e$ ` ] < ` $[\![v(p_1)]\!]_e$ ` ; ... ; ` $[\![v(p_m)]\!]_e$ ` >`

and $[\![(\ell, (w+d)[R])]\!]_v =$

`[ ` ℓ' ` : ` $[\![w(x_1^R)]\!]_e$ ` + ` $[\![d]\!]_e$ ` ; ... ; ` $[\![w(x_n^R)]\!]_e$ ` + ` $[\![d]\!]_e$ ` ] < ` $[\![v(p_1)]\!]_e$ ` ; ... ; ` $[\![v(p_m)]\!]_e$ ` >`

Hence, for all b such that $[\![(\ell, w)]\!]_v \mapsto_{[\![\mathcal{A}]\!]} b$, there exists a concrete state (ℓ', w') in $v(\mathcal{A})$ such that $(\ell, w) \longrightarrow (\ell', w')$ and $b = [\![(\ell', w')]\!]_v$. □

Theorem 1. *Let $\mathcal{A} = (\Sigma, L, \ell_0, X, P, I, E)$ be a parametric timed automaton, $v(\mathcal{A}) = (S, s_0, \longrightarrow)$ be $\mathcal{A}$'s concrete semantics with respect to a parameter valuation v, and $[\![\mathcal{A}]\!] = (\Sigma, E, L, R)$. Then, $[\![\_]\!]_v$ is a bisimulation map between the transition systems $(S, s_0, \longrightarrow)$ and $\left((T_{\Sigma,\mathit{NState}})_v, [\![s_0]\!]_v, \mapsto_{[\![\mathcal{A}]\!]}\right)$.*

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 $[\![\mathcal{A}]\!]$ 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 $[\![\mathcal{A}]\!]$ 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" [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:

```
φ                 ‖ [ idle : X1 ; X2 ] < P1 ; P2 ; P3 >   ⇝[[A]]
φ and T' ≥ 0/1 ‖ < idle : X1 + T' ; X2 + T' > < P1 ; P2 ; P3 >
```

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 `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`.

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

Example 4. *In the module `PTA-COFFEE`, the command*

```
smt-search  [ idle : X1  ; X2  ] < P1 ; P2 ; P3 >  =>*
            < done : X1' ; X2' > < P1 ; P2 ; P3 >
    such that (X1 === X2 and X1 >= 0/1 and P1 >= 0/1
               and P2 >= 0/1 and P3 >= 0/1) = 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 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 `=>*`, together with the constraint in the "`such that`" section of the query, specify initial states where the values of the clocks are equal (`X1 === X2`) 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 X1 === X2 and X1 >= 0/1 and P1 >= 0/1 and P2 >= 0/1 and
  ... and #1-T:Real + #2-T:Real === P3 and
  ... and #3-T:Real <= 10/1 and #1-T:Real + #2-T:Real === P3
```

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 (`X1'`) is `#3-T` $\leq$ `10/1`, while the second clock (`X2'`) is the sum of the delays accumulated in locations `add-sugar`, `preparing-coffee` and `done`, and therefore `X2' >= P3`.

4.2 Soundness and Completeness

This section shows that the transition system induced by the symbolic rewrite relation $\rightsquigarrow_{[\![\mathcal{A}]\!]}$ is bisimilar to the PZG of $\mathcal{A}$. 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 (ℓ, Z) is a valid reachable state in the PZG of $\mathcal{A}$ (Z cannot be empty) iff the boolean expression in the corresponding constrained term in $[\![\mathcal{A}]\!]$ is satisfiable (and hence, reachable via $\rightsquigarrow_{[\![\mathcal{A}]\!]}$).

Lemma 4. *For any zone Z, $Z \neq \emptyset$ iff $[\![Z]\!]_b$ is satisfiable.*

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 $[\![\_]\!]_b$, 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. □

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

$$\phi \parallel \{\ell : e_1; \ldots; e_n\} \texttt{ <P1;} \ldots \texttt{;P}m\texttt{>}$$

where Pi are variables, e_i expressions (possibly containing SMT variables) and ϕ must contain at least one inequality for each variable occurring after the symbol $\|$ (*e.g.* `#1-T` $\geq$ `0/1`). We use U and V to range over elements in $\mathcal{T}_\mathcal{Z}$.

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 $U = \phi \parallel \{\ell : e_1; \cdots ; e_n\}$<P1$; \cdots ;$Pm>. We define the following operations on $\mathcal{T}_\mathcal{Z}$ − terms:*

- ***Reset:*** $U[R] \overset{def}{=} \phi \parallel \{\ell : e_1^R; \dots ; e_n^R\}$ <P1$; \dots ;$Pm>
- ***Time elapse:*** $U^{\nearrow} \overset{def}{=} (\phi$ *and* T $\geq$ 0/1$) \parallel \{\ell : e_1 + T; \dots ; e_n + T\}$ <P1$; \dots ;$Pm> *where* T *is a fresh variable (not occurring in ϕ).*
- ***Conjunction:*** *Let G be a boolean expression such that $var(G) \subseteq var(U)$. Then,*

$$U \wedge G \overset{def}{=} (\phi \;\; \textit{and} \;\; G) \parallel \{\ell : e_1; \dots ; e_n\} \texttt{<}P1; \dots ; Pm\texttt{>}.$$

- ***Instantiation:*** *Given a clock valuation w and a parameter valuation v,*

$$U\{v, w\} \overset{def}{=} (\phi \;\; \textit{and} \;\; \psi) \parallel \{\ell : e_1; \dots ; e_n\} \texttt{<}P1; \dots ; Pm\texttt{>}$$

where ψ is the boolean expression

$$e_1 === [\![w(x_1)]\!]_e \;\; \textit{and} \;\; \cdots \;\; \textit{and} \;\; e_n === [\![w(x_n)]\!]_e \;\; \textit{and}$$
$$P1 === [\![v(p_1)]\!]_e \;\; \textit{and} \;\; \cdots \;\; \textit{and} \;\; Pm === [\![v(p_m)]\!]_e.$$

$U\{v, w\}$ equates (`===`) 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 $\sim \;\; \subseteq (L \times \mathcal{C}) \times \mathcal{T}_\mathcal{Z}$ as follows:*

$$(\ell, Z) \sim U = \phi \parallel \texttt{<}\ell : e_1; \dots ; e_n\texttt{>} \; \texttt{<}P1; \dots ; Pm\texttt{>}$$

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 (ℓ, Z) in the PZG of $\mathcal{A}$ 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 2.

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

$$U = \phi \parallel \texttt{<}\ell : e_1; \dots ; e_n\texttt{>}\texttt{<}P1; \dots ; Pm\texttt{>}$$

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 ϕ `and` ψ (see Definition 2) in $U\{v, w\}$ is satisfiable, so it is ϕ `and` ψ `and` `0/1 === 0/1`. On the other side, if $x_i \in R$, `0/1` $= [\![w'(x_i)]\!]_e$ is satisfiable iff $w'(x_i) = 0$. $\square$

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

$$U = \phi \parallel \texttt{<}\ell : e_1; \dots ; e_n\texttt{>}\texttt{<}P1; \cdots ; Pm\texttt{>}$$

Then, $(\ell, Z^{\nearrow}) \sim U^{\nearrow}$.

Proof. By definition, $v, w + T \models Z^{\nearrow}$ 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 $U^{\nearrow}$ (and the inequality $T \geq 0$ is added to the constraint) . $\square$

Lemma 7 (Conjunction)**.** *Let G be a guard or an invariant, $Z \neq \emptyset$, $U = \phi \parallel$ <$\ell : e_1; \ldots ; e_n$ ><P1; ... ; Pm > and assume that $(\ell, Z) \sim U$. Then, $(\ell, Z \cap G) \sim U \wedge ([\![G]\!]_b[\mathtt{Xi}/e_i])$.*

Recall that the relation $\Rightarrow$ 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 $Z' = ((Z \cap g)[R] \cap I(\ell'))^{\nearrow} \cap I(\ell')$ (if Z' is not empty). Let $\overset{2}{\rightsquigarrow}_{[\![\mathcal{A}]\!]}$ be the application of a ℓ-σ rule followed by a tick rule, and let $\overset{2}{\rightsquigarrow}{}^{*}_{[\![\mathcal{A}]\!]}$ be its reflexive and transitive closure. The following Theorem shows that $\Rightarrow$ on the PZG is bisimilar to $\overset{2}{\rightsquigarrow}_{[\![\mathcal{A}]\!]}$ on constrained terms.

Theorem 2. *Let $\mathcal{A}$ be a PTA. $\sim$ is a bisimulation between the transition systems $(\mathcal{C}, s_0, \Rightarrow)$ and $(\mathcal{T}_{\mathcal{Z}}, \phi_0 \parallel t_0, \overset{2}{\rightsquigarrow}_{[\![\mathcal{A}]\!]})$ where $t_0 =$ <ℓ_0 : T; ... ; T ><P1; ... ; Pm > and $\phi_0 =$ (Pi $\geq$ 0/1 and T $\geq$ 0/1 and $[\![I(\ell_0)]\!]_b[\mathtt{Xi}/\mathtt{T}]$).*

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

Let $U = \phi \parallel \langle \ell : e_1; \cdots ; e_n \rangle \langle \mathtt{P1}; \cdots ; \mathtt{Pm} \rangle$ and (ℓ, Z) s.t. $(\ell, Z) \sim U$. The result follows from lemmas 5, 6 and 7 and noticing that $U \rightsquigarrow_{[\![\mathcal{A}]\!]} \left((U \wedge \overline{g})[R] \wedge \overline{I(\ell')}\right)^{\dagger} \rightsquigarrow_{[\![\mathcal{A}]\!]} \left((U \wedge \overline{g})[R] \wedge \overline{I(\ell')}\right)^{\nearrow} \wedge \overline{I(\ell')}$, where, $\overline{g} = [\![g]\!]_b[e_i/\mathtt{Xi}]$, and similarly for the invariants. Note also that Lemma 4 guarantees that a given transition is enabled in (ℓ, Z) iff it is also enabled at U whenever $(\ell, Z) \sim U$. $\square$

4.3 Symbolic Reachability Analysis with Folding

Many PTAs $\mathcal{A}$ generate finite PZGs (so reachability analysis should terminate for both positive and negative queries), while the number of symbolic states generated by `smt-search` of the corresponding $[\![\mathcal{A}]\!]$ 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. 3a. After each iteration i between ℓ_0 and ℓ_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* `bad` *can be reached. In this case, the command*

```
smt-search [l0 : 0/1;0/1]< > =>* <bad : X';Y'>< >
```

finds a solution to this reachability problem.

Now consider the automaton in Fig. 3b, where the location `bad` *cannot be reached. The execution of the command* `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:*

s_0 : 0/1 $\leq$ #0-T $\parallel$ $\langle \ell_0$: #0-T$\rangle$

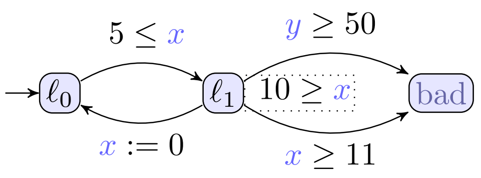
s_1 : ϕ_0 and 5/1 $\leq$ #0-T and 10/1 $\geq$ #0-T + #1-T $\parallel$ $\langle \ell_1$: #0-T + #1-T$\rangle$

s_2 : ϕ_1 and 0/1 $\leq$ #2-T $\parallel$ $\langle \ell_0$: #2-T$\rangle$

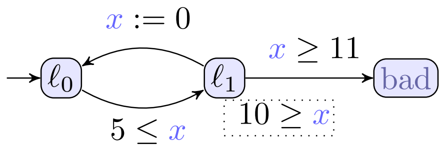
s_3 : ϕ_2 and 5/1 $\leq$ #2-T and 10/1 $\geq$ #2-T + #3-T $\parallel$ $\langle \ell_1$: #2-T + #3-T$\rangle$

...

where ϕ_i is the constraint in the state s_i. Note that $[\![s_0]\!] = [\![s_2]\!]$ and $[\![s_1]\!] = [\![s_3]\!]$ (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 `smt-search` *command keeps exploring the successor states of s_2. Note also that, due to the definition of $\rightsquigarrow$, the constraints are always accumulated. For instance, ϕ_2 includes inequalities about* `#0-T` *and* `#1-T` *that are no longer used in the expression representing the value of the clock x.*



(a) Automaton with two clocks.



(b) Automaton with one clock.

Figure 3: Automata in Example 5.

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 θ 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

```
red reachability((φ ∥ t), ℓ, 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 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 $V\theta$, and the function `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*

```
red reachability((X >= 0/1) || < l0 : X > < >) .
```

computes the set of reachable states starting from location ℓ_0, with any non-negative initial clock value. The result is:

```
X >= 0/1 || < l0: X > < > ;;        --- State 1
X >= 0/1 and 5/1 <= X and 10/1 >= X and #1-T >= 0/1
  and 10/1 >= X + #1-T   || < l1: X + #1-T > < >  --- State 2
```

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

Formally, the command `reachable` computes, iteratively, the folding reachable transition system $(\mathcal{T}^f_{\mathcal{Z}}, U_0, \overset{2f}{\rightsquigarrow}_{[\![\mathcal{A}]\!]})$ [EM07, BM14] where $\mathcal{T}^f_{\mathcal{Z}}$ is defined as $\bigcup_{i\in\mathbb{N}} S_i$, $S_0 = \{U_0\}$ and $S_{n+1} = \{U \mid \exists V \in U_n \text{ s.t. } V \overset{2}{\rightsquigarrow}_{[\![\mathcal{A}]\!]} U \text{ and } U \not\sqsubseteq V' \text{ for any } V' \in S_{k\leq n}\}$; and $\overset{2f}{\rightsquigarrow}_{[\![\mathcal{A}]\!]} = \bigcup_{n\in\mathbb{N}}(\rightarrow_n)$ where $\rightarrow_0 = \emptyset$ and $\rightarrow_{n+1} = \{(U,V) \in S_n \times \bigcup_{0\leq i\leq n+1} S_i \mid \exists V'.U \overset{2}{\rightsquigarrow}_{[\![\mathcal{A}]\!]} V' \text{ and } V' \sqsubseteq V\}$.

Theorem 3 (Termination)**.** *If the PZG $(\mathcal{C}, s_0, \Rightarrow)$ of a PTA $\mathcal{A}$ is a finite transition system, then $(\mathcal{T}^f_{\mathcal{Z}}, U_0, \overset{2f}{\rightsquigarrow}_{[\![\mathcal{A}]\!]})$ is also a finite transition system.*

Proof. Suppose, to obtain a contradiction, that the transition system $(\mathcal{T}^f_{\mathcal{Z}}, U_0, \overset{2f}{\rightsquigarrow}_{[\![\mathcal{A}]\!]})$ has infinitely many states. Since the number of locations is finite, there must be a cycle[1] $(\ell_0, Z_0) \Rightarrow^* (\ell_{k-1}, Z_{k-1}) \Rightarrow (\ell_0, Z_0)$ in the PZG that is simulated ($\sim$) in the transition system of $[\![\mathcal{A}]\!]$ as

$$V_0 \overset{2}{\rightsquigarrow}{}^*_{[\![\mathcal{A}]\!]} V_{k-1} \overset{2}{\rightsquigarrow}_{[\![\mathcal{A}]\!]} V_k \overset{2}{\rightsquigarrow}{}^*_{[\![\mathcal{A}]\!]} V_{2k-1} \overset{2}{\rightsquigarrow}_{[\![\mathcal{A}]\!]} V_{2k} \overset{2}{\rightsquigarrow}{}^*_{[\![\mathcal{A}]\!]} \cdots$$

where: for all $i \geq 0$, $(\ell_{i \ \mathbf{mod}\ k}, Z_{i \ \mathbf{mod}\ k}) \sim V_i$; and $V_{jk} \not\sqsubseteq V_{ik}$ for $0 \leq i < j$. In each cycle, from V_{ik} to $V_{k(i+1)}$, k new fresh variables are created. Assume that the clock expressions in V_0 are $(x_0, x_1, \cdots, x_n)$. Let $\Delta^i = T_1^i + \cdots + T_{k-1}^i + T_0^{i+1}$ be the sequence of k fresh variables created when moving from V_{ik+1} to $V_{k(i+1)}$. Moreover, for any of such sequences, let $\Delta[p]$ be the suffix of Δ 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 $\rightsquigarrow$ is deterministic, in the sense that rules always add the same constraint (using fresh variables when needed), the expression for a clock x_i when it is reset or not is, respectively:

Iter/State.	Reset	Non-reset	Constraint
0 (V_0)	$x_i + T_0^0$	$x_i + T_0^0$	
1 (V_k)	Δ_p^1	$x_i + T_0^0 + \Delta^1$	$G_{k-1}(k-1) \wedge I_0(k)$
2 (V_{2k})	Δ_p^2	$x_i + T_0^0 + \Delta^1 + \Delta^2$	$G_{k-1}(2k-1) \wedge I_0(2k)$
$\cdots$			

where $G_{k-1}((k-1))$ denotes the guard of the transition from location ℓ_{k-1} to ℓ_0 evaluated using the clock expressions in the state V_{k-1}. Similarly, $I_0(k)$ is the invariant at location ℓ_0 evaluated with the clock expressions in the state V_k. Recall that the last element in Δ^1 if T_0^2 and it corresponds to the variable created due to the delay in state V_k.

Now we build a unifier θ between V_k and V_{2k} 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 $\Delta^1[p]$ to the corresponding one in $\Delta^2[p]$. 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 Δ^1 with $T_0^2 + \sum \Delta^2$ (and the remaining variables as expected). This intuitively corresponds to "extending" the delay T_0^2 (at state V_k) with the sum of all the delays of the cycle, including the one in V_{2k}. 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 x_i, the subsequence $x_i + T_0^0 + \Delta^1[-p]$ is unified with $x_i + T_0^0 + \Delta^1 + \Delta^2[-p]$ as in the second case, matching the last element T' in $\Delta^1[-p]$ with $T' + \Delta^2[-p]$.

Let ψ be a valuation for the variables in the constraint C_{2k} (state V_{2k}). From ψ, we can build the corresponding parameter (v) and clock (w) valuation. Since both V_k and V_{2k} are related to (ℓ_0, Z_0) via $\sim$, $C_{2k}\{v, w\}$ and $C_k\{v, w\}$ both eval to true. Since $G_{k-1}(2k-1)\psi$ evals to true, so does $G_{k-1}(k-1)\theta\psi$. Similarly for the invariants. Hence, ψ is also a valuation making true C_{2k} and the implication $C_k \Rightarrow C_{2k}\theta$ holds. We have $V_{2k} \sqsubseteq V_k$, thus a contradiction. $\square$

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

[1] ℓ_0 is not intended to be the initial location. The index 0 has been used to simplify the notation.

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 4. *Let $\mathcal{A}$ be a PTA and ϕ a state predicate. The* EF-emptiness problem *asks: "is the set of parameter valuations v such that there exists a reachable state (ℓ, w) in $v(\mathcal{A})$ satisfying ϕ empty?".* EF-synthesis *is the problem of computing parameter valuations v such that a run of $v(\mathcal{A})$ reaches a state satisfying ϕ. The* safety synthesis problem *$AG\neg\phi$ is the problem of computing the set of parameter valuations for which states satisfying ϕ 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 $[\![\ell]\!]_b = \mathtt{L} == \ell$ (for a variable `L` of sort `Location`) to the definition of $[\![\_]\!]_b$. The command

```
smt-search [1] [ℓ0 : 0/1 ; ... ; 0/1 ] < P1 ; ... ; Pm > =>*
    < L:Location : X1' ; ... ; Xn' > < P1 ; ... ; Pm >
    such that [[φ]]b  and P1 >= 0/1 and ... and Pm >= 0/1 .
```

then tries to find a path from ℓ_0 to an arbitrary location `L` satisfying ϕ. The resulting constraint, if any, is an answer to the synthesis problem $\mathsf{EF}\phi$. 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 $\mathsf{AG}\neg\phi$ can be solved by finding all solutions for $\mathsf{EF}\phi$ and then negating the resulting constraint.

Example 7. *Let ϕ be the output of the `smt-search` command in Example 4. Since ϕ is satisfiable, there are values for the parameters such that `done` is reachable and the answer to the EF-emptiness problem EF(`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 ϕ, but not the parameters. Using a quantifier elimination procedure, ϕ' can be simplified to $0 \leq p_2 \wedge p_2 \leq p_3 \wedge 0 \leq p_1$. (We are currently using the tactic* `qe` *of the Z3 theorem prover to automate this step). This means that* done *is reachable whenever $p_2 \leq p_3$.*

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

```
smt-search  [ idle : 0/1 ; 0/1 ]    < P1 ; P2 ; P3 >   =>*
            < L    : X1' ; X2' >    < P1 ; P2 ; P3 >
  such that (L == preparing_coffee and X1' =/== X2' and
             P1 >= 0/1 and P2 >= 0/1 and P3 >= 0/1) = 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

```
smt-search [1,10] [ idle : 0/1 ; 0/1 ] < P1 ; P2 ; P3 >  =>*
                  < L    : X1' ; X2' > < P1 ; P2 ; P3 >
  such that X1' > X2' and P1 >= 0/1 and P2 >= 0/1 and P3 >= 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 * P1 > P2`, the number of symbolic states is finite (at most one dose of sugar is possible). `smt-search`, without depth bound, does not terminate. However, the new command `reachability` halts, finding 9 (different) symbolic states, with none of them satisfying `X1' > X2'`. Therefore, when $2 \times p_1 > p_2$, there is no valuation reaching a state where $x_1 > x_2$.* □

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 $(\mathcal{T}_{\mathcal{Z}}, \phi_0 \parallel t_0, \overset{2}{\rightsquigarrow}_{[\![\mathcal{A}]\!]})$.

Example 8 (Strategies)**.** *The answer $p_2 \leq p_3$ to the synthesis problem* EF(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 n 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* **not 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 : ... #1-T >= P1 and #1-T <= P2 ... || <done : 0/1 ; #5-T + #6-T + #7-T> < P1 ; P2 ; P3 >
```

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(ℓ) for different locations ℓ 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]).

Model	Clocks	Parameters	Actions	Locations	Transitions
gear-1000	2	3	6	4074	4073
blowup-200	3	5	4	702	800
Pipeline_KP12_2_3	5	6	11	100	244
RCP	6	5	16	95	198

Table 1: Benchmarks considered.

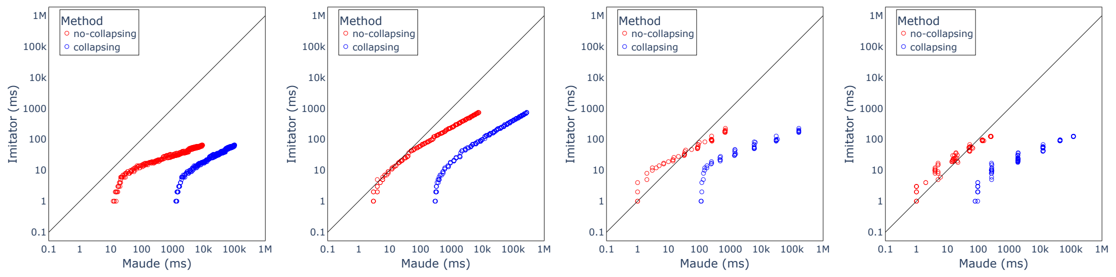


Figure 4: Execution times of Imitator and Maude in log-scale. From left to right, we consider the benchmarks (see Section 6): `gear-1000`, `blowup-200`, `Pipeline_KP12_2_3`, `RCP`.

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 $[\![\mathcal{A}]\!]$. 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.

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 [ÖM02] 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 [ÖM02], 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 [ÖM08, ÖM07] extension of Maude—has been applied to a wide range of real-time systems [Ölv14] and has provided formal semantics and formal analysis to a number of modeling languages [Ölv11]. 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 [ÖM06], 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 [LKBÖ21] 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, Ölveczky, 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).

References

[ABO+22] Jaime Arias, Kyungmin Bae, Carlos Olarte, Peter Csaba Ölveczky, Laure Petrucci, and Fredrik Rømming. pta2maude, 2022.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[ADY+09] Musab AlTurki, Dinakar Dhurjati, Dachuan Yu, Ajay Chander, and Hiroshi Inamura. Formal specification and analysis of timing properties in software systems. In Marsha Chechik and Martin Wirsing, editors, *Fundamental Approaches to Software Engineering, 12th International Conference, FASE 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5503 of *Lecture Notes in Computer Science*, pages 262–277. Springer, 2009.

[AFS13] Étienne André, Laurent Fribourg, and Romain Soulat. Merge and conquer: State merging in parametric timed automata. In Dang Van Hung and Mizuhito Ogawa, editors, *Automated Technology for Verification and Analysis - 11th International Symposium, ATVA 2013, Hanoi, Vietnam, October 15-18, 2013. Proceedings*, volume 8172 of *Lecture Notes in Computer Science*, pages 381–396. Springer, 2013.

[AHV93] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 592–601. ACM, 1993.

[AMvdP21] Étienne André, Dylan Marinho, and Jaco van de Pol. A benchmarks library for extended parametric timed automata. In Frédéric Loulergue and Franz Wotawa, editors, *Tests and Proofs - 15th International Conference, TAP 2021, Held as Part of STAF 2021, Virtual Event, June 21-22, 2021, Proceedings*, volume 12740 of *Lecture Notes in Computer Science*, pages 39–50. Springer, 2021.

[And19] Étienne André. What's decidable about parametric timed automata? *Int. J. Softw. Tools Technol. Transf.*, 21(2):203–219, 2019.

[And21] Étienne André. IMITATOR 3: Synthesis of timing parameters beyond decidability. In Alexandra Silva and K. Rustan M. Leino, editors, *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part I*, volume 12759 of *Lecture Notes in Computer Science*, pages 552–565. Springer, 2021.

[BBBC16] Peter Bezdek, Nikola Benes, Jiri Barnat, and Ivana Cerná. LTL parameter synthesis of parametric timed automata. In Rocco De Nicola and eva Kühn, editors, *Software Engineering and Formal Methods - 14th International Conference, SEFM 2016, Held as Part of STAF 2016, Vienna, Austria, July 4-8, 2016, Proceedings*, volume 9763 of *Lecture Notes in Computer Science*, pages 172–187. Springer, 2016.

[BEM13] Kyungmin Bae, Santiago Escobar, and José Meseguer. Abstract logical model checking of infinite-state systems using narrowing. In Femke van Raamsdonk, editor, *24th International Conference on Rewriting Techniques and Applications, RTA 2013, June 24-26, 2013, Eindhoven, The Netherlands*, volume 21 of *LIPIcs*, pages 81–96. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013.

[BGG$^+$18] Rakesh Bobba, Jon Grov, Indranil Gupta, Si Liu, José Meseguer, Peter Csaba Ölveczky, and Stephen Skeirik. *Survivability: Design, Formal Modeling, and Validation of Cloud Storage Systems Using Maude*, chapter 2, pages 10–48. John Wiley & Sons, 2018.

[BKMÖ15] Kyungmin Bae, Joshua Krisiloff, José Meseguer, and Peter Csaba Ölveczky. Designing and verifying distributed cyber-physical systems using Multirate PALS: an airplane turning control system case study. *Sci. Comput. Program.*, 103:13–50, 2015.

[BM14] Kyungmin Bae and José Meseguer. Infinite-state model checking of LTLR formulas using narrowing. In Santiago Escobar, editor, *Rewriting Logic and Its Applications - 10th International Workshop, WRLA 2014, Held as a Satellite Event of ETAPS, Grenoble, France, April 5-6, 2014, Revised Selected Papers*, volume 8663 of *Lecture Notes in Computer Science*, pages 113–129. Springer, 2014.

[BMÖ19] Giovanna Broccia, Paolo Milazzo, and Peter Csaba Ölveczky. Formal modeling and analysis of safety-critical human multitasking. *Innov. Syst. Softw. Eng.*, 15(3-4):169–190, 2019.

[BÖF$^+$12] Kyungmin Bae, Peter Csaba Ölveczky, Thomas Huining Feng, Edward A. Lee, and Stavros Tripakis. Verifying hierarchical ptolemy II discrete-event models using Real-Time Maude. *Sci. Comput. Program.*, 77(12):1235–1271, 2012.

[BR19] Kyungmin Bae and Camilo Rocha. Symbolic state space reduction with guarded terms for rewriting modulo SMT. *Sci. Comput. Program.*, 178:20–42, 2019.

[CDE$^+$07] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott, editors. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.

[CDE$^+$22] Manuel Clavel, Francisco Durán, Steven Eker, Santiago Escobar, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, Rubén Rubio, and Carolyn Talcott. *Maude Manual (Version 3.2.1)*. SRI International, 2022. Available at `http://maude.cs.illinois.edu`.

[CEFX09] Remy Chevallier, Emmanuelle Encrenaz-Tiphène, Laurent Fribourg, and Weiwen Xu. Timed verification of the generic architecture of a memory circuit using parametric timed automata. *Formal Methods Syst. Des.*, 34(1):59–81, 2009.

[CGP01] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking, 1st Edition*. MIT Press, 2001.

[CKJ$^+$22] Eu-teum Choi, Tae-hyung Kim, Yong-Kee Jun, Seongjin Lee, and Mingyun Han. On-the-fly repairing of atomicity violations in arinc 653 software. *Applied Sciences*, 12(4), 2022.

[DLL$^+$15] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, and Danny Bøgsted Poulsen. Uppaal SMC tutorial. *Int. J. Softw. Tools Technol. Transf.*, 17(4):397–415, 2015.

[EM07] Santiago Escobar and José Meseguer. Symbolic model checking of infinite-state systems using narrowing. In Franz Baader, editor, *Term Rewriting and Applications, 18th International Conference, RTA 2007, Paris, France, June 26-28, 2007, Proceedings*, volume 4533 of *Lecture Notes in Computer Science*, pages 153–168. Springer, 2007.

[FSLM12] Laurent Fribourg, Romain Soulat, David Lesens, and Pierre Moro. Robustness analysis for scheduling problems using the inverse method. In Ben C. Moszkowski, Mark Reynolds, and Paolo Terenziani, editors, *19th International Symposium on Temporal Representation and Reasoning, TIME 2012, Leicester, United Kingdom, September 12-14, 2012*, pages 73–80. IEEE Computer Society, 2012.

[GÖ14] Jon Grov and Peter Csaba Ölveczky. Formal modeling and analysis of google's megastore in Real-Time Maude. In Shusaku Iida, José Meseguer, and Kazuhiro Ogata, editors, *Specification, Algebra, and Software - Essays Dedicated to Kokichi Futatsugi*, volume 8373 of *Lecture Notes in Computer Science*, pages 494–519. Springer, 2014.

[HRSV02] Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits W. Vaandrager. Linear parametric model checking of timed automata. *J. Log. Algebraic Methods Program.*, 52-53:183–220, 2002.

[JLR15] Aleksandra Jovanovic, Didier Lime, and Olivier H. Roux. Integer parameter synthesis for real-time systems. *IEEE Trans. Software Eng.*, 41(5):445–461, 2015.

[KP12] Michal Knapik and Wojciech Penczek. Bounded model checking for parametric timed automata. *Trans. Petri Nets Other Model. Concurr.*, 5:141–159, 2012.

[LÁÖ15] Daniela Lepri, Erika Ábrahám, and Peter Csaba Ölveczky. Sound and complete timed CTL model checking of timed kripke structures and real-time rewrite theories. *Sci. Comput. Program.*, 99:128–192, 2015.

[LKBÖ21] Jaehun Lee, Sharon Kim, Kyungmin Bae, and Peter Csaba Ölveczky. Hybridsynchaadl: Modeling and formal analysis of virtually synchronous cpss in AADL. In Alexandra Silva and K. Rustan M. Leino, editors, *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part I*, volume 12759 of *Lecture Notes in Computer Science*, pages 491–504. Springer, 2021.

[LMT15] Kim Guldstrand Larsen, Marius Mikucionis, and Jakob Haahr Taankvist. Safe and optimal adaptive cruise control. In Roland Meyer, André Platzer, and Heike Wehrheim, editors, *Correct System Design - Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday, Oldenburg, Germany, September 8-9, 2015. Proceedings*, volume 9360 of *Lecture Notes in Computer Science*, pages 260–277. Springer, 2015.

[LÖ09] Elisabeth Lien and Peter Csaba Ölveczky. Formal modeling and analysis of an IETF multicast protocol. In Dang Van Hung and Padmanabhan Krishnan, editors, *Seventh IEEE International Conference on Software Engineering and Formal Methods, SEFM 2009, Hanoi, Vietnam, 23-27 November 2009*, pages 273–282. IEEE Computer Society, 2009.

[LÖM16] Si Liu, Peter Csaba Ölveczky, and José Meseguer. Modeling and analyzing mobile ad hoc networks in Real-Time Maude. *J. Log. Algebraic Methods Program.*, 85(1):34–66, 2016.

[LPY01] Magnus Lindahl, Paul Pettersson, and Wang Yi. Formal design and analysis of a gear controller. *Int. J. Softw. Tools Technol. Transf.*, 3(3):353–368, 2001.

[LRST09] Didier Lime, Olivier H. Roux, Charlotte Seidner, and Louis-Marie Traonouez. Romeo: A parametric model-checker for petri nets with stopwatches. In Stefan Kowalewski and Anna Philippou, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 15th International Conference, TACAS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5505 of *Lecture Notes in Computer Science*, pages 54–57. Springer, 2009.

[Mes92] José Meseguer. Conditioned rewriting logic as a united model of concurrency. *Theor. Comput. Sci.*, 96(1):73–155, 1992.

[Mes97] José Meseguer. Membership algebra as a logical framework for equational specification. In Francesco Parisi-Presicce, editor, *Recent Trends in Algebraic Development Techniques, 12th International Workshop, WADT'97, Tarquinia, Italy, June 1997, Selected Papers*, volume 1376 of *Lecture Notes in Computer Science*, pages 18–61. Springer, 1997.

[Mes20] José Meseguer. Generalized rewrite theories, coherence completion, and symbolic methods. *J. Log. Algebraic Methods Program.*, 110, 2020.

[NPvdP18] Hoang Gia Nguyen, Laure Petrucci, and Jaco van de Pol. Layered and collecting NDFS with subsumption for parametric timed automata. In *23rd International Conference on Engineering of Complex Computer Systems, ICECCS 2018, Melbourne, Australia, December 12-14, 2018*, pages 1–9. IEEE Computer Society, 2018.

[NT22] Vivek Nigam and Carolyn L. Talcott. Automating safety proofs about cyber-physical systems using rewriting modulo SMT. In Kyungmin Bae, editor, *Rewriting Logic and Its Applications - 14th International Workshop, WRLA@ETAPS 2022, Munich, Germany, April 2-3, 2022, Revised Selected Papers*, volume 13252 of *Lecture Notes in Computer Science*, pages 212–229. Springer, 2022.

[ÖBM10] Peter Csaba Ölveczky, Artur Boronat, and José Meseguer. Formal semantics and analysis of behavioral AADL models in Real-Time Maude. In John Hatcliff and Elena Zucca, editors, *Formal Techniques for Distributed Systems, Joint 12th IFIP WG 6.1 International Conference, FMOODS 2010 and 30th IFIP WG 6.1 International Conference, FORTE 2010, Amsterdam, The Netherlands, June 7-9, 2010. Proceedings*, volume 6117 of *Lecture Notes in Computer Science*, pages 47–62. Springer, 2010.

[ÖC06] Peter Csaba Ölveczky and Marco Caccamo. Formal simulation and analysis of the CASH scheduling algorithm in Real-Time Maude. In Luciano Baresi and Reiko Heckel, editors, *Fundamental Approaches to Software Engineering, 9th International Conference, FASE 2006, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 27-28, 2006, Proceedings*, volume 3922 of *Lecture Notes in Computer Science*, pages 357–372. Springer, 2006.

[Ölv11] Peter Csaba Ölveczky. Semantics, simulation, and formal analysis of modeling languages for embedded systems in Real-Time Maude. In Gul Agha, Olivier Danvy, and José Meseguer, editors, *Formal Modeling: Actors, Open Systems, Biological Systems - Essays Dedicated to Carolyn Talcott on the Occasion of Her 70th Birthday*, volume 7000 of *Lecture Notes in Computer Science*, pages 368–402. Springer, 2011.

[Ölv14] Peter Csaba Ölveczky. Real-Time Maude and its applications. In Santiago Escobar, editor, *Rewriting Logic and Its Applications - 10th International Workshop, WRLA 2014, Held as a Satellite Event of ETAPS, Grenoble, France, April 5-6, 2014, Revised Selected Papers*, volume 8663 of *Lecture Notes in Computer Science*, pages 42–79. Springer, 2014.

[ÖM02] Peter Csaba Ölveczky and José Meseguer. Specification of real-time and hybrid systems in rewriting logic. *Theor. Comput. Sci.*, 285(2):359–405, 2002.

[ÖM06] Peter Csaba Ölveczky and José Meseguer. Abstraction and completeness for Real-Time Maude. In Grit Denker and Carolyn L. Talcott, editors, *Proceedings of the 6th International Workshop on Rewriting Logic and its Applications, WRLA 2006, Vienna, Austria, April 1-2, 2006*, volume 174 of *Electronic Notes in Theoretical Computer Science*, pages 5–27. Elsevier, 2006.

[ÖM07] Peter Csaba Ölveczky and José Meseguer. Semantics and pragmatics of real-time maude. *High. Order Symb. Comput.*, 20(1-2):161–196, 2007.

[ÖM08] Peter Csaba Ölveczky and José Meseguer. The real-time maude tool. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 332–336. Springer, 2008.

[ÖMT06] Peter Csaba Ölveczky, José Meseguer, and Carolyn L. Talcott. Specification and analysis of the AER/NCA active network protocol suite in real-time maude. *Formal Methods Syst. Des.*, 29(3):253–293, 2006.

[ÖT09] Peter Csaba Ölveczky and Stian Thorvaldsen. Formal modeling, performance estimation, and model checking of wireless sensor network algorithms in Real-Time Maude. *Theor. Comput. Sci.*, 410(2-3):254–280, 2009.

[RMM17] Camilo Rocha, José Meseguer, and César A. Muñoz. Rewriting modulo SMT and open system analysis. *J. Log. Algebraic Methods Program.*, 86(1):269–297, 2017.

[WWG$^+$15] Ya Wang, Rui Wang, Yong Guan, Xiaojuan Li, Hongxing Wei, and Jie Zhang. Formal modeling and verification of the safety critical fire-fighting control system. In Sheikh Iqbal Ahamed, Carl K. Chang, William C. Chu, Ivica Crnkovic, Pao-Ann Hsiung, Gang Huang, and Jingwei Yang, editors, *39th Annual Computer Software and Applications Conf., COMPSAC Workshops 2015, Taichung, Taiwan, July 1-5, 2015*, pages 536–541. IEEE Computer Society, 2015.

[YB20] Geunyeol Yu and Kyungmin Bae. Maude-SE: a tight integration of maude and SMT solvers. In *Preliminary proceedings of WRLA@ETAPS*, pages 220–232, 2020.