Symbolic Analysis and Parameter Synthesis for Time Petri Nets Using Maude and SMT Solving

Jaime Arias[1], Kyungmin Bae[2], Carlos Olarte[1(✉)], Peter Csaba Ölveczky[3], Laure Petrucci[1], and Fredrik Rømming[4]

[1] LIPN, CNRS UMR 7030, Université Sorbonne Paris Nord, Villetaneuse, France
olarte@lipn.fr
[2] Pohang University of Science and Technology, Pohang, South Korea
[3] University of Oslo, Oslo, Norway
[4] University of Cambridge, Cambridge, UK

Abstract. In this paper we present a concrete and a symbolic rewriting logic semantics for parametric time Petri nets with inhibitor arcs (PITPNs). We show how this allows us to use Maude combined with SMT solving to provide sound and complete formal analyses for PITPNs. We develop a new general folding approach for symbolic reachability that terminates whenever the parametric state-class graph of the PITPN is finite. We explain how almost all formal analysis and parameter synthesis supported by the state-of-the-art PITPN tool `Roméo` can be done in Maude with SMT. In addition, we also support analysis and parameter synthesis from *parametric* initial markings, as well as full LTL model checking and analysis with user-defined execution strategies. Experiments on three benchmarks show that our methods outperform `Roméo` in some cases.

Keywords: parametric timed Petri nets · semantics · rewriting logic · Maude · SMT · parameter synthesis · symbolic reachability analysis

1 Introduction

System designers often do not know in advance the concrete values of key system parameters, and want to find those values that make the system behave as desired. *Parametric time Petri nets with inhibitor arcs* (PITPNs) [2,20,28,49] extend the popular time(d) Petri nets [22,30,51] to the setting where bounds on when transitions can fire are unknown or only partially known.

The formal analysis of PITPNs—including synthesizing the values of the parameters which make the system satisfy desired properties—is supported by the state-of-the-art tool `Roméo` [29], which has been applied to a number of applications, e.g., [3,19,46]. `Roméo` supports the analysis and parameter synthesis for reachability (is a certain marking reachable?), liveness (will a certain marking be reached in all behaviors?), time-bounded "until," and bounded response (will each P-marking be followed by a Q-marking within time Δ?), all from *concrete* initial markings. `Roméo` does not support a number of desired features, including:

L. Gomes and R. Lorenz (Eds.): PETRI NETS 2023, LNCS 13929, pp. 369–392, 2023.
https://doi.org/10.1007/978-3-031-33620-1_20

- Broader set of system properties, e.g., full (i.e., nested) temporal logic.
- Starting with *parametric* initial markings and synthesizing also the initial markings that make the system satisfy desired properties.
- Analysis with user-defined execution strategies. For example, what happens if I always choose to fire transition t instead of t' when they are both enabled?
- Providing a "testbed" for PITPNs in which different analysis methods can quickly be developed and evaluated. This is not supported by `Roméo`, which is a high-performance tool with dedicated algorithms implemented in `C++`.

PITPNs do not support many features needed for large distributed systems, such as user-defined data types and functions. Rewriting logic [31,32]—supported by the Maude language and tool [18], and by Real-Time Maude [37,43] for real-time systems—is an expressive logic for distributed and real-time systems. In rewriting logic, any computable data type can be specified as an (algebraic) equational specification, and the dynamic behaviors of a system are specified by rewriting rules over terms (representing states). Because of its expressiveness, Real-Time Maude has been successfully applied to a number of large and sophisticated real-time systems—including 50-page active networks and IETF protocols [27,44], industrial cloud systems [13,21], scheduling algorithms with unbounded queues [39], airplane turning algorithms [8], and so on—beyond the scope of most popular formalisms for real-time systems. Its expressiveness has also made Real-Time Maude a useful semantic framework and formal analysis backend for industrial modeling languages [1,9,36,38].

This expressiveness comes at a price: most analysis problems are undecidable in general. Real-Time Maude uses explicit-state analysis where only *some* points in time are visited. All possible system behaviors are therefore *not* analyzed (for dense time domains), and hence the analysis is unsound in many cases [41].

This paper exploits the recent integration of SMT solving into Maude to address the first problem above (more features for PITPNs) and to take the second step towards addressing the second problem (developing sound and complete analysis methods for rewriting-logic-based real-time systems).

Maude combined with SMT solving, e.g., as implemented in the Maude-SE tool [53], allows us to perform *symbolic rewriting* of "states" $\phi \parallel t$, where the term t is a state pattern that contains variables, and ϕ is an SMT constraint restricting the possible values of those variables.

Section 3 defines a (non-executable) "concrete" rewriting logic semantics for (instantiated) PITPNs in "Real-Time Maude style" [42], and proves that this semantics is bisimilar to the one for PITPNs in [49]. Section 4 transforms this semantics into a Maude-with-SMT semantics for *parametric* PITPNs, and shows how to perform sound symbolic analysis of such nets using Maude-with-SMT. However, existing symbolic reachability analysis methods may fail to terminate even when the state class graph of the PITPN is finite (and hence `Roméo` analysis terminates). We therefore develop a new method for "folding" symbolic states, and show that reachability analysis with such folding terminates whenever the state class graph of the PITPN is finite.

In Sect. 5 we show how all analysis methods supported by `Roméo`—with one small exception: the time bounds in some temporal formulas cannot be

parameters—also can be performed using Maude-with-SMT. In addition, we support analysis and parameter synthesis for *parametric* initial markings, model checking full temporal logic formulas, and analysis w.r.t. user-defined execution strategies. Our methods are implemented in Maude, using its meta-programming features. This makes it very easy to develop new analysis methods for PITPNs.

This work also constitutes the second step in our quest to develop sound and complete analysis methods for dense-time real-time systems in Real-Time Maude. We present both a Real-Time Maude-style semantics in Sect. 3 *and* the symbolic semantics in Sect. 4 to explore how we can transform Real-Time Maude models into Maude-with-SMT models for symbolic analysis. In our first step in this quest, we studied symbolic rewrite methods for the much simpler parametric timed automata [4]; see Sect. 7 for a comparison with that work.

In Sect. 6 we benchmark both `Roméo` and our Maude-with-SMT methods, and find that in some cases our high-level prototype outperforms `Roméo`.

The longer report [6] has proofs of all results in this paper and much more detail. All executable Maude files with analysis commands, tools for translating `Roméo` files into Maude, and data from the benchmarking are available at [5].

2 Preliminaries

Transition Systems. A *transition system* $\mathcal{A}$ is a triple $(A, a_0, \rightarrow_{\mathcal{A}})$, where A is a set of *states*, $a_0 \in A$ is the *initial state*, and $\rightarrow_{\mathcal{A}} \subseteq A \times A$ is a *transition relation*. We call $\mathcal{A}$ *finite* if the set of states reachable by $\rightarrow_{\mathcal{A}}$ from a_0 is finite. A relation $\sim \subseteq A \times B$ is a *bisimulation* [16] between $\mathcal{A}$ and $\mathcal{B} = (B, b_0, \rightarrow_{\mathcal{B}})$ iff: (i) $a_0 \sim b_0$; and (ii) for all a, b s.t. $a \sim b$: if $a \rightarrow_{\mathcal{A}} a'$ then there is a b' s.t. $b \rightarrow_{\mathcal{B}} b'$ and $a' \sim b'$, and, vice versa, if $b \rightarrow_{\mathcal{B}} b''$, then there is a a'' s.t. $a \rightarrow_{\mathcal{A}} a''$ and $a'' \sim b''$.

Parametric Time Petri Nets with Inhibitor Arcs (PITPN). $\mathbb{N}$, $\mathbb{Q}_+$, and $\mathbb{R}_+$ denote, resp., the natural numbers, the non-negative rational numbers, and the non-negative real numbers. We assume a finite set $\Lambda = \{\lambda_1, \ldots, \lambda_l\}$ of *time parameters*. A *parameter valuation* π is a function $\pi : \Lambda \rightarrow \mathbb{R}_+$. A (linear) *inequality* over Λ is an expression $\sum_{1 \leq i \leq l} a_i \lambda_i \prec b$, where $\prec \in \{<, \leq, =, \geq, >\}$ and $a_i, b \in \mathbb{R}$. A *constraint* is a conjunction of such inequalities. $\mathcal{L}(\Lambda)$ denotes the set of all constraints over Λ. A parameter valuation π *satisfies* a constraint $K \in \mathcal{L}(\Lambda)$, written $\pi \models K$, if the expression obtained by replacing each λ in K with $\pi(\lambda)$ evaluates to true. An interval I of $\mathbb{R}_+$ is a $\mathbb{Q}_+$-interval if its left endpoint ${}^{\uparrow}I$ belongs to $\mathbb{Q}_+$ and its right endpoint $I^{\uparrow}$ belongs to $\mathbb{Q}_+ \cup \{\infty\}$. We denote by $\mathcal{I}(\mathbb{Q}_+)$ the set of $\mathbb{Q}_+$-intervals. A parametric time interval is a function $I : {\mathbb{Q}_+}^{\Lambda} \rightarrow \mathcal{I}(\mathbb{Q}_+)$ that associates with each parameter valuation a $\mathbb{Q}_+$-interval. The set of parametric time intervals over Λ is denoted $\mathcal{I}(\Lambda)$.

Definition 1. *A* parametric time Petri net with inhibitor arcs *(PITPN) [49] is a tuple* $\mathcal{N} = \langle P, T, \Lambda, {}^{\bullet}(.), (.)^{\bullet}, {}^{\circ}(.), M_0, J, K_0 \rangle$ *where*

- $P = \{p_1, \ldots, p_m\}$ *is a non-empty finite set (of* places*),*
- $T = \{t_1, \ldots, t_n\}$ *is a non-empty finite set (of* transitions*), with* $P \cap T = \emptyset$*,*

- $\Lambda = \{\lambda_1, \ldots, \lambda_l\}$ *is a finite set of* parameters,
- ${}^\bullet(.) \in [T \to \mathbb{N}^P]$ *is the* backward incidence function,
- $(.)^\bullet \in [T \to \mathbb{N}^P]$ *is the* forward incidence function,
- ${}^\circ(.) \in [T \to \mathbb{N}^P]$ *is the* inhibition function,
- $M_0 \in \mathbb{N}^P$ *is the* initial marking,
- $J \in [T \to \mathcal{I}(\Lambda)]$ *assigns a* parametric time interval *to each transition, and*
- $K_0 \in \mathcal{L}(\Lambda)$ *is the* initial constraint *over* Λ.

If $\Lambda = \emptyset$ *then* $\mathcal{N}$ *is a (non-parametric)* time Petri net with inhibitor arcs *(ITPN).*

A *marking* of $\mathcal{N}$ is an element $M \in \mathbb{N}^P$, where $M(p)$ is the number of tokens in place p. $\pi(\mathcal{N})$ denotes the ITPN where each occurrence of λ_i in the PITPN $\mathcal{N}$ has been replaced by $\pi(\lambda_i)$ for a parameter valuation π.

The *concrete semantics* of a PITPN $\mathcal{N}$ is defined in terms of concrete ITPNs $\pi(\mathcal{N})$ where $\pi \models K_0$. A transition t is *enabled* in M if $M \geq {}^\bullet t$ (the number of tokens in M in each input place of t is greater than or equal to the value on the arc between this place and t). A transition t is *inhibited* if the place connected to one of its inhibitor arcs is marked with at least as many tokens as the weight of the inhibitor arc. A transition t is *active* if it is enabled and not inhibited. The sets of enabled and inhibited transitions in marking M are denoted *Enabled*(M) and *Inhibited*(M), respectively. Transition t is *firable* if it has been (continuously) enabled for at least time ${}^\uparrow J(t)$, without counting the time it has been inhibited. Transition t is *newly enabled* by the firing of transition t_f in M if it is enabled in the resulting marking $M' = M - {}^\bullet t_f + t_f^\bullet$ but was not enabled in $M - {}^\bullet t_f$:

$$\mathit{NewlyEnabled}(t, M, t_f) = ({}^\bullet t \leq M - {}^\bullet t_f + t_f^\bullet) \wedge ((t = t_f) \vee \neg({}^\bullet t \leq M - {}^\bullet t_f)).$$

NewlyEnabled(M, t_f) denotes the transitions newly enabled by firing t_f in M.

The semantics of an ITPN is defined as a transition system with states (M, I), where M is a marking and I is a function mapping each transition enabled in M to a time interval, and two kinds of transitions: *time* transitions where time elapses, and discrete transitions when a transition in the net is fired.

Definition 2 (ITPN Semantics [49]). *The transition system for an ITPN* $\pi(\mathcal{N})$ *is* $\mathcal{S}_{\pi(\mathcal{N})} = (\mathcal{A}, a_0, \to)$*, where:* $\mathcal{A} = \mathbb{N}^P \times [T \to \mathcal{I}(\mathbb{Q})]$*,* $a_0 = (M_0, J)$ *and* $(M, I) \to (M', I')$ *if there exist* $\delta \in \mathbb{R}_+$*,* $t \in T$*, and state* (M'', I'') *such that* $(M, I) \xrightarrow{\delta} (M'', I'')$ *and* $(M'', I'') \xrightarrow{t} (M', I')$*, for the following relations:*

- *the* time transition relation*, defined* $\forall \delta \in \mathbb{R}_+$ *by:* $(M, I) \xrightarrow{\delta} (M, I')$ *iff* $\forall t \in T$*:*
$$\begin{cases} I'(t) = \begin{cases} I(t) \text{ if } t \in \mathit{Enabled}(M) \text{ and } t \in \mathit{Inhibited}(M) \\ {}^\uparrow I'(t) = \max(0, {}^\uparrow I(t) - \delta) \text{ and } I'(t)^\uparrow = I(t)^\uparrow - \delta \text{ otherwise} \end{cases} \\ M \geq {}^\bullet(t) \implies I'(t)^\uparrow \geq 0 \end{cases}$$
- *the* discrete transition relation*, defined* $\forall t_f \in T$ *by:* $(M, I) \xrightarrow{t_f} (M', I')$ *iff*
$$\begin{cases} t_f \in \mathit{Enabled}(M) \wedge t_f \notin \mathit{Inhibited}(M) \wedge M' = M - {}^\bullet t_f + t_f^\bullet \wedge {}^\uparrow I(t_f) = 0 \\ \forall t \in T, I'(t) = \begin{cases} J(t) \text{ if } \mathit{NewlyEnabled}(t, M, t_f) \\ I(t) \text{ otherwise} \end{cases} \end{cases}$$

The *symbolic* semantics of PITPNs is given in [2] as a transition system $(\mathbb{N}^P \times \mathcal{L}(\Lambda), (M_0, K_0), \Rightarrow)$ on *state classes*, i.e., pairs $c = (M, D)$ consisting of a marking M and a constraint D over Λ. The firing of a transition leads to a new marking as in the concrete semantics, and also captures the new constraints induced by the time that has passed for the transition to fire. See [2] for details.

Rewrite Theories. A *rewrite theory* [31] is a tuple $\mathcal{R} = (\Sigma, E, L, R)$ where

- the signature Σ declares sorts, a subsort partial order, and function symbols;
- E is a set of equations of the form $t = t'$ **if** ψ, where t and t' are Σ-terms of the same sort, and ψ is a conjunction of equations;
- L is a set of *labels*; and
- R is a set of rewrite rules of the form $l : q \longrightarrow r$ **if** ψ, where $l \in L$ is a label, q and r are Σ-terms of the same sort, and ψ is a conjunction of equations.

$T_{\Sigma,s}$ denotes the set of ground (i.e., variable-free) terms of sort s, and $T_\Sigma(X)_s$ the set of terms of sort s over a set of variables X. $T_\Sigma(X)$ and T_Σ denote all terms and ground terms, respectively. A substitution $\sigma : X \to T_\Sigma(X)$ maps each variable to a term of the same sort, and $t\sigma$ denotes the term obtained by simultaneously replacing each variable x in a term t with $\sigma(x)$.

A *one-step rewrite* $t \longrightarrow_\mathcal{R} t'$ holds if there is a rule $l : q \longrightarrow r$ **if** ψ, a subterm u of t, and a substitution σ such that $u = q\sigma$ (modulo equations), t' is the term obtained from t by replacing u with $r\sigma$, and $v\sigma = v'\sigma$ holds for each $v = v'$ in ψ. We denote by $\longrightarrow^*_\mathcal{R}$ the reflexive-transitive closure of $\longrightarrow_\mathcal{R}$. A rewrite theory $\mathcal{R}$ is *topmost* iff there is a sort *State* at the top of one of the connected components of the subsort partial order such that for each rule, both sides have the top sort *State*, and no operator has sort *State* or any of its subsorts as an argument sort.

Rewriting with SMT [47]. For a signature Σ and equations E, a *built-in theory* $\mathcal{E}_0$ is a first-order theory with a signature $\Sigma_0 \subseteq \Sigma$, where (1) each sort s in Σ_0 is minimal in Σ; (2) $s \notin \Sigma_0$ for each operator $f : s_1 \times \cdots \times s_n \to s$ in $\Sigma \setminus \Sigma_0$; and (3) f has no other subsort-overloaded typing in Σ_0. The satisfiability of a constraint in $\mathcal{E}_0$ is assumed decidable using the SMT theory $\mathcal{T}_{\mathcal{E}_0}$.

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 $\mathcal{E}_0$ [11,47]. A constrained term $\phi \parallel t$ *symbolically* represents all instances of the pattern t such that ϕ holds: $[\![\phi \parallel t]\!] = \{t' \mid t' = t\sigma \text{ (modulo } E) \text{ and } \mathcal{T}_{\mathcal{E}_0} \models \phi\sigma \text{ for ground } \sigma : X_0 \to T_{\Sigma_0}\}$.

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 $\mathcal{E}_0$. 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 $\sigma : X \to T_\Sigma(X_0)$ such that (1) $t = q\sigma$ and $t' = r\sigma$ (modulo equations), (2) $\mathcal{T}_{\mathcal{E}_0} \models (\phi \wedge \psi\sigma) \Leftrightarrow \phi'$, and (3) ϕ' is $\mathcal{T}_{\mathcal{E}_0}$-satisfiable. We denote by $\rightsquigarrow^*_\mathcal{R}$ the reflexive-transitive closure of $\rightsquigarrow_\mathcal{R}$.

A *symbolic rewrite* on constrained terms symbolically represents a (possibly infinite) set of system transitions. 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 [18] is a language and tool supporting the specification and analysis of rewrite theories. We summarize its syntax below:

```
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
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 of a term t (`red` t), simulation by rewriting (`rew` t), and rewriting following a given strategy (`srew` t `using` str). Basic strategies include $r[\sigma]$ (apply rule r once with the optional ground substitution σ) and `all` (apply any of the rules once). Compound strategies include concatenation ($\alpha\,;\,\beta$), α `or-else` β (execute β if α fails), normalization $\alpha!$ (execute α until it cannot be further applied), etc.

Maude also offers explicit-state reachability analysis from a ground term t (`search [`n`,`m`]` t `=>*` t' `such that` Φ) and model checking an LTL formula F (`red modelCheck(`t`,` F`)`). For symbolic reachability analysis, the command

```
smt-search [n, m]: t =>* t' such that Φ  --- n and m are optional
```

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 $\mathcal{E}_0$.

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-SE [53] extends Maude with additional functionality for rewriting modulo SMT and bindings with different SMT solvers.

3 A Rewriting Logic Semantics for ITPNs

This section presents a rewriting logic semantics for (non-parametric) ITPNs, using a (non-executable) rewrite theory $\mathcal{R}_\mathbf{0}$. We provide a bisimulation relating the concrete semantics of a net $\mathcal{N}$ and a rewrite relation in $\mathcal{R}_\mathbf{0}$, and discuss variants of $\mathcal{R}_\mathbf{0}$ to avoid consecutive tick steps and to enable time-bounded analysis.

3.1 Formalizing ITPNs in Maude: The Theory $\mathcal{R}_\mathbf{0}$

We fix $\mathcal{N}$ to be the ITPN $\langle P, T, \emptyset, {}^\bullet(.), (.)^\bullet, {}^\circ(.), M_0, J, true\rangle$, and show how ITPNs and markings of such nets can be represented as Maude terms.

The usual approach is to represent a transition t_i and a place p_j as a constant of sort `Label` and `Place`, respectively (e.g., `ops` p_1 p_2 `...` p_m `: -> Place [ctor]`). To use a single rewrite theory $\mathcal{R}_0$ to define the semantics of all ITPNs, we instead assume that places and transition (labels) can be represented as strings; i.e., there is an injective naming function $\eta : P \cup T \to$ `String` which we usually do not mention explicitly.[1]

[1] We do not show variable declarations, but follow the convention that variables are written in (all) capital letters.

```
protecting STRING .   protecting RAT .
sorts Label Place .   --- identifiers for transitions and places
subsorts String < Label Place .   --- we use strings for simplicity
sorts Time TimeInf .  --- time values
subsort Zero PosRat < Time  < TimeInf .
op inf : -> TimeInf [ctor] .
eq T <= inf = true .
```

The sort `TimeInf` adds an "infinity" value `inf` to the sort `Time` of time values, which are the non-negative rational numbers (`PosRat`).

The "standard" way of formalizing Petri nets in rewriting logic [31,48] represents, e.g., a marking with two tokens in place p and three tokens in place q as the Maude term $p\ p\ q\ q\ q$. This is crucial to support *concurrent* firings of transitions in a net. Since the semantics of PITPNs is an *interleaving* semantics, to enable rewriting-with-SMT-based analysis from *parametric* initial markings, we instead represent markings as maps from places to the number of tokens in that place, so that the above marking is represented by the Maude term $\eta(p)$ `|->2 ;` $\eta(q)$ `|->3` of sort `Marking`. The Maude term $\eta(t)$ `:` *pre* `->` *post* `inhibit` *inhibit* `in` *interval* represents a transition $t \in T$, where *pre*, *post*, and *inhibit* are markings representing, respectively, ${}^\bullet(t), (t)^\bullet, {}^\circ(t)$; and *interval* represents the interval $J(t)$. A `Net` is represented as a `;`-separated set of such transitions:

```
sort Marking . --- Markings
op empty : -> Marking [ctor] .
op _|->_ : Place Nat -> Marking [ctor] .
op _;_ : Marking Marking -> Marking [ctor assoc comm id: empty] .
sort Interval .  --- Time intervals (upper bound can be infinite)
op `[_:_`] : Time TimeInf -> Interval [ctor] .
sorts Net Transition .   subsort Transition < Net .
op _`:_-->_inhibit_in_ :
  Label Marking Marking Marking Interval -> Transition [ctor] .
op emptyNet : -> Net [ctor] .
op _;_ : Net Net -> Net [ctor assoc comm id: emptyNet] .
```

Example 1. Assuming the obvious naming function η mapping A to `"A"`, and so on, the Maude term `net3(`a`)` represents the net in Fig. 1:

```
op net3 : Time -> Net .
eq net3(T) =
   "t1" : "p5" |-> 1 --> "p1" |-> 1 inhibit empty in [2 : 6] ;
   "t2" : "p1" |-> 1 --> "p2" |-> 1 ; "p5" |-> 1 inhibit empty in [2 : 4] ;
   "t3" : "p2" |-> 1 ; "p4" |-> 1 --> "p3" |-> 1 inhibit empty in [T : T] ;
   "t4" : "p3" |-> 1 --> "p4" |-> 1 inhibit empty in [0 : 0] .
```

It is very easy to define operations `+`, `-`, and `<=` on markings (see [6]); we can then check whether a transition is active in a marking:

```
op active : Marking Transition -> Bool .  --- Active transition
eq active(M, L : PRE --> POST inhibit INHIBIT in INTERVAL) =
     (PRE <= M) and not inhibited(M, INHIBIT) .
op inhibited : Marking Marking -> Bool .  --- Inhibited transition
eq inhibited(M, empty) = false .
eq inhibited((P |-> N2) ; M, (P |-> N) ; INHIBIT) =
     ((N > 0) and (N2 >= N)) or inhibited(M, INHIBIT) .
```

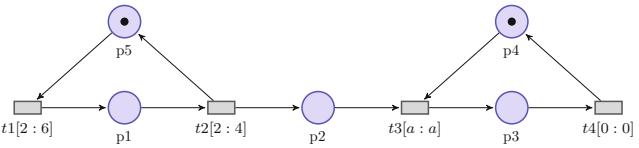

Fig. 1. A simple production-consumption system taken from [52]

Dynamics. We define the dynamics of ITPNs as a Maude "interpreter" for such nets. The definition of the semantics in [49] adjusts the "time intervals" of non-inhibited transitions when time elapses, but seems slightly "inconsistent": Time interval end-points should be non-negative, and only enabled transitions have intervals in the states; however, the definition of time and discrete transitions in [49] mentions $\forall t \in T, I'(t) = \ldots$ and $M \geq {}^\bullet(t) \implies I'(t)^\uparrow \geq 0$. Taking the definition of time and transition steps in [49] leads us to time intervals where the right end-points of disabled transitions could have arbitrarily large *negative* values. To have a simple and well-defined semantics, we use "clocks" instead of "decreasing intervals"; a clock denotes how long the corresponding transition has been enabled (but not inhibited). Our semantics is equivalent to the (natural interpretation of the) one in [49] in a way made precise in Theorem 1.

The sort `ClockValues` (see [6]) denotes sets of `;`-separated terms $\eta(t)$ `->`τ, where t is the (label of the) transition and τ represents the current value of t's "clock." The states in $\mathcal{R}_\mathbf{0}$ are terms m : *clocks* : *net* of sort `State`, where m represents the current marking, *clocks* the current values of the transition clocks, and *net* the representation of the Petri net:

```
sort State .    op _:_:_ : Marking ClockValues Net -> State [ctor] .
```

The following rewrite rule models the application of a transition `L`. Since `_;_` is associative and commutative, *any* transition `L` in the net can be applied:

```
crl [applyTransition] :
     M  :  (L -> T) ; CLOCKS  :
     (L : PRE --> POST inhibit INHIBIT in INTERVAL) ; NET
 =>  (M - PRE) + POST : L -> 0 ; updateClocks(CLOCKS, M - PRE, NET) :
     (L : PRE --> POST inhibit INHIBIT in INTERVAL) ; NET'
 if active(M, L : PRE --> POST inhibit INHIBIT in INTERVAL)
    and (T in INTERVAL) .
```

The transition `L` is active in the marking `M` and its clock value `T` is in the `INTERVAL`. After performing the transition, the marking is `(M - PRE) + POST`, the clock of `L` is reset and the other clocks are updated using the following function:

```
eq updateClocks((L' -> T') ; CLOCKS, INTERM-M,
                (L' : PRE --> POST inhibit INHIBIT in INTERVAL) ; NET)
 = if PRE <= INTERM-M then (L' -> T') else (L' -> 0) fi ;
   updateClocks(CLOCKS, INTERM-M, NET) .
eq updateClocks(empty,  INTERM-M, NET) = empty .
```

The second rewrite rule in $\mathcal{R}_\mathbf{0}$ specifies how time advances. Time can advance by *any* value `T`, as long as time does not advance beyond the time when an active

transition must be taken. The clocks are updated according to the elapsed time T, except for those transitions that are disabled or inhibited:

```
crl [tick] : M : CLOCKS : NET => M : increaseClocks(M, CLOCKS, NET, T) : NET
    if  T <= mte(M, CLOCKS, NET) [nonexec] .
```

This rule is not executable ([nonexec]), since the variable T, which denotes how much time advances, only occurs in the right-hand side of the rule. T is therefore *not* assigned any value by the substitution matching the rule with the state being rewritten. This time advance T must be less or equal to the minimum of the upper bounds of the enabled transitions in the marking M:[2]

```
op mte : Marking ClockValues Net -> TimeInf .
eq mte(M, (L -> T) ; CLOCKS, (L : PRE --> POST ... in [T1 : inf]) ; NET)
 = mte(M, CLOCKS, NET) .
eq mte(M, (L -> T) ; CLOCKS, (L : PRE --> ... in [T1 : T2]) ; NET)
 = if active(M, L : PRE --> ...) then min(T2 - T, mte(M, CLOCKS, NET))
   else mte(M, CLOCKS, NET) fi .
eq mte(M, empty, NET) = inf .
```

The function increaseClocks increases the transitions clocks according to the elapsed time, except for those transitions that are disabled or inhibited:

```
op increaseClocks : Marking ClockValues Net Time -> ClockValues .
eq increaseClocks(M, (L -> T1) ; CLOCKS, (L : PRE --> ...) ; NET, T)
 = if active(M, L : PRE --> ...)
   then (L -> T1 + T) else (L -> T1) fi ; increaseClocks(M,CLOCKS,NET,T) .
eq increaseClocks(M, empty, NET, T) = empty .
```

The function $[\![\_]\!]_{\mathcal{R}_0}$ (see [6] for its formal definition) formalizes how markings and nets are represented as terms in rewriting logic.[3]

To show that $\mathcal{R}_0$ correctly simulates any ITPN $\mathcal{N}$, we provide a bisimulation relating behaviors from $a_0 = (M_0, J)$ in $\mathcal{N}$ with behaviors in $\mathcal{R}_0$ starting from initial state $[\![M_0]\!]_{\mathcal{R}_0}$: initClocks($[\![\mathcal{N}]\!]_{\mathcal{R}_0}$) : $[\![\mathcal{N}]\!]_{\mathcal{R}_0}$, where initClocks(*net*) is a clock valuation which assigns the value 0 to each transition (label) $\eta(t)$ in *net*.

Since a transition in $\mathcal{N}$ consists of a delay followed by a discrete transition, we define a corresponding rewrite relation $\mapsto$ combining the tick and applyTransition rules, and prove (in [6]) the bisimulation for this relation. The following relation relates our clock-based states with the interval-based states:

Definition 3. *Let $\mathcal{N}$ be an ITPN and $\mathcal{S}_{\mathcal{N}} = (\mathcal{A}, a_0, \rightarrow)$ its concrete semantics. We define a relation $\approx \subseteq \mathcal{A} \times T_{\Sigma,State}$, relating states in the concrete semantics of $\mathcal{N}$ to states in $\mathcal{R}_0$, where for all states $(M, I) \in \mathcal{A}$, $(M, I) \approx m$: clocks : net if and only if $m = [\![M]\!]_{\mathcal{R}_0}$ and net $= [\![\mathcal{N}]\!]_{\mathcal{R}_0}$ and for each transition $t \in T$,*

- *the value of $\eta(t)$ in clocks is 0 if t is not enabled in M;*
- *otherwise:*
  - *if $J(t)^{\uparrow} \neq \infty$ then the value of clock $\eta(t)$ in clocks is $J(t)^{\uparrow} - I(t)^{\uparrow}$;*
  - *otherwise, if ${}^{\uparrow}I(t) > 0$ then $\eta(t)$ has the value ${}^{\uparrow}J(t) - {}^{\uparrow}I(t)$ in clocks; otherwise, the value of $\eta(t)$ in clocks could be any value $\tau \geq {}^{\uparrow}J(t)$.*

[2] Parts of Maude specification will be replaced by '...' throughout the paper.

[3] $[\![\_]\!]_{\mathcal{R}_0}$ is parametrized by the naming function η, not shown explicitly here.

Theorem 1. $\approx$ *is a bisimulation between transition systems* $\mathcal{S}_{\mathcal{N}} = (\mathcal{A}, a_0, \rightarrow)$ *and* $(T_{\Sigma,State}, (\llbracket M_0 \rrbracket_{\mathcal{R}_0} : \texttt{initClocks}(\llbracket \mathcal{N} \rrbracket_{\mathcal{R}_0}) : \llbracket \mathcal{N} \rrbracket_{\mathcal{R}_0}), \mapsto)$.

3.2 Some Variations of $\mathcal{R}_0$

The theory $\mathcal{R}_1$ avoids consecutive applications of the `tick` rule by adding a new component—with value `tickOk` or `tickNotOk`—to the global state. The tick rule can only be applied when this component is `tickOk`. We add a new constructor `_:_:_:_` for these global states, a new sort `TickState` with values `tickOk` and `tickNotOk`, and add two rewrite rules:

```
sort TickState .      ops tickOk tickNotOk : -> TickState [ctor] .
op _:_:_:_ : TickState Marking ClockValues Net -> State [ctor] .
crl [applyTransition] :
  TS : M : ((L -> T) ; CLOCKS) : (L : PRE --> ...) ; NET) =>
  tickOk : ((M - PRE) + POST) :  ... if active(...) and (T in INTERVAL) .
crl [tick] : tickOk : M : ...  => tickNotOk : M : increaseClocks(...) ...
  if  T <= mte(M, CLOCKS, NET) [nonexec] .
```

We prove in [6] that $m : cs : net \longrightarrow^*_{\mathcal{R}_0} m' : cs' : net$ iff `tickOk` $: m : cs : net \longrightarrow^*_{\mathcal{R}_1}$ `tickNotOk` $: m' : cs' : net$. While reachability is preserved, a `tick` rule application in $\mathcal{R}_1$, where time does not advance far enough for a transition to be taken, could lead to a deadlock in $\mathcal{R}_1$ which cannot happen in $\mathcal{R}_0$.

The theory $\mathcal{R}_2$ adds a "global clock", denoting how much time has elapsed in the system, to answer questions such as whether a certain state can be reached in a certain time interval, and to enable time-bounded analysis where behaviors beyond the time bound are not explored. $\mathcal{R}_2$ adds the "global time," to the state:

```
op _:_:_:_@_ :  TickState Marking ClockValues Net Time -> State [ctor] .
```

The rewrite rules are modified as expected. For instance, the rule `tick` becomes:

```
crl [tick] :   tickOk    : M : CLOCKS : NET @ GT
          =>   tickNotOk : M : increaseClocks(..., T) : NET @ GT + T
if  T <= mte(M, CLOCKS, NET) [nonexec] .
```

where `GT` is a variable of sort `Time`. For a time bound Δ, we can add a conjunct `GT + T <=` Δ in the condition of this rule to stop executing beyond the time bound.

3.3 Explicit-state Analysis of ITPNs in Maude

The theories $\mathcal{R}_0$–$\mathcal{R}_2$ cannot be directly executed in Maude, since the `tick` rule introduces a new variable `T` in its right-hand side. Following the Real-Time Maude methodology, we can "sample" system execution at *some* time points, e.g., by changing the `tick` rule to increase time by *one time unit* in each application:

```
crl [tickOne] : M : CLOCKS : NET => M : increaseClocks(M, CLOCKS, NET, 1) : NET
                if  1 <= mte(M, CLOCKS, NET) .
```

Such time sampling analysis is in general not sound and complete, since it does not cover all possible system behaviors for dense time domains. Nevertheless, if all interval bounds are natural numbers, then "all behaviors" should be covered.

We can quickly experiment with different parameter values for our model, before applying the sound and complete symbolic methods developed in Sects. 4 and 5. Our report [6] describes a wealth of such analyses, including LTL model checking and time-bounded analysis. Here we just check whether the net in Fig. 1 is 1000-safe when $a = 5$, where the term `init3` denotes the initial marking in Fig. 1. We define a function `k-safe`, where `k-safe(`n`,` m`)` holds iff the marking m does not have any place with more than n tokens:

```
op k-safe : Nat Marking -> Bool .
eq k-safe(N, empty) = true .
eq k-safe(N1, P |-> N2 ; M) = N2 <= N1 and k-safe(N1, M) .
```

We can then quickly check whether the net is 1000-safe when $a = 5$:

```
Maude> search [1] init3 : initClocks(net3(5)) : net3(5)  =>*
                  M : CLOCKS : NET such that not k-safe(1000, M) .

Solution 1 (state 83924)
M -->"p1" |-> 0 ; "p2" |-> 1001 ; "p3" |-> 0 ; "p4" |-> 1 ; "p5" |-> 1
```

The net is not 1000-safe: we reached a state with 1001 tokens in place p_2. Similar searches show that the net is 2-safe (but not 1-safe) if $a = 4$ and 1-safe if $a = 3$.

4 Parameters and Symbolic Executions

Standard explicit-state Maude analysis of $\mathcal{R}_0$–$\mathcal{R}_2$ cannot be used to analyze all behaviors of PITPNs for two reasons: **(1)** The rule `tick` introduces a new variable `T` in its right-hand side, reflecting that time can advance by *any* value `T <= mte(...)`; and **(2)** analyzing nets with *uninitialized* parameters is impossible with explicit-state Maude analysis of concrete states. (For example, the condition `T in INTERVAL` in rule `applyTransition` does not evaluate to `true` if `INTERVAL` is not a *concrete* interval, and hence the rule will never be applied.) Maude-SE analysis of *symbolic* states with SMT variables can solve both issues, by symbolically representing the time advances `T` and the uninitialized parameters.

This section defines a rewrite theory $\mathcal{R}_1^{\mathbf{S}}$ that faithfully models PITPNs and that can be symbolically executed using Maude-SE. We prove that (concrete) executions in $\mathcal{R}_1$ are captured by (symbolic) executions in $\mathcal{R}_1^{\mathbf{S}}$, and vice versa. We also show that standard folding techniques [33] in rewriting modulo SMT are not sufficient for collapsing equivalent symbolic states in $\mathcal{R}_1^{\mathbf{S}}$. We therefore propose a new folding technique that guarantees termination of the reachability analyses of $\mathcal{R}_1^{\mathbf{S}}$ when the state-class graph of the encoded PITPN is finite.

4.1 The Symbolic Rewriting Logic Semantics

We define the "symbolic" semantics of PITPNs using the rewrite theory $\mathcal{R}_1^{\mathbf{S}}$, which is the symbolic counterpart of $\mathcal{R}_1$, instead of basing it on $\mathcal{R}_0$, since a symbolic "tick" step represents all possible tick steps from a symbolic state. We therefore do not introduce deadlocks not possible in the corresponding PITPN.

$\mathcal{R}_1^{\mathbf{S}}$ is obtained from $\mathcal{R}_1$ by replacing the sort `Nat` in markings and the sort `PosRat` for clock values with the corresponding SMT sorts `Integer` and `Real`. (The former is only needed to enable reasoning with *symbolic* initial states where the number of tokens in a location is unknown). Conditions in rules (e.g., `M1 <= M2`) are replaced with the corresponding SMT expressions of sort `Boolean`. The symbolic execution of $\mathcal{R}_1^{\mathbf{S}}$ in Maude-SE will accumulate and check the satisfiability of the constraints needed for a parametric transition to happen.

We start by declaring the sort `Time` as follows:

```
sorts Time TimeInf .  subsort Real < Time < TimeInf .
op inf : -> TimeInf [ctor] .
```

where `Real` is the sort for SMT reals (constraints in rewrite rules guarantee that only non-negative numbers are considered). Intervals are defined as in $\mathcal{R}_0$. Since `Real` is a subsort of `Time`, a parametric interval $[a, b]$ in a PITPN can be represented in $\mathcal{R}_1^{\mathbf{S}}$ as the term `[a:Real : b:Real]`, where `a` and `b` are variables of sort `Real`. The definition and operations on markings, nets, and clock values are similar to those in Sect. 3.1, albeit with the appropriate SMT sorts.

The rewrite rules in $\mathcal{R}_1^{\mathbf{S}}$ act on symbolic states that may contain SMT variables. Although these rules are similar to those in $\mathcal{R}_1$, their symbolic execution is completely different. Maude-SE defines a theory transformation to implement symbolic rewriting. In the resulting theory $\widehat{\mathcal{R}_1^{\mathbf{S}}}$, when a rule is applied, the variables occurring in the right-hand side but not in the left-hand side are replaced by fresh variables. Moreover, rules in $\widehat{\mathcal{R}_1^{\mathbf{S}}}$ act on constrained terms of the form $\phi \parallel t$, where t in this case is a term of sort `State` and ϕ is a satisfiable SMT boolean expression. The constraint ϕ is obtained by accumulating the conditions in rules, thereby restricting the possible values of the variables in t.

The tick rewrite rule in $\mathcal{R}_1^{\mathbf{S}}$ is

```
crl [tick] :  tickOk    : M : CLOCKS     : NET
           =>  tickNotOk : M : increaseClocks(M, CLOCKS, NET, T) : NET
if (T >= 0/1 and mte(M, CLOCKS, NET, T)) = true .
```

The variable `T` is restricted to be a non-negative real number and to satisfy the following *predicate* `mte`, which gathers the constraints to ensure that time cannot advance beyond the point in time when an enabled transition *must* fire:

```
op mte : Marking ClockValues Net Real -> Boolean .
eq mte(M, empty, NET, T) = true .
eq mte(M, (L -> R1) ; CLOCKS, (L : PRE --> ...  in [T1 : inf]) ; NET, T)
 = mte(M, CLOCKS, NET, T) .
eq mte(M, (L -> R1) ; CLOCKS, (L : PRE --> ...  in [T1 : T2])  ; NET, T)
 = (active(M, L : ...) ? T <= T2 - R1 : true) and mte(M, CLOCKS, NET, T) .
```

This means that, for every transition `L`, if the upper bound of the interval in `L` is `inf`, no restriction on `T` is added. Otherwise, if `L` is active at marking `M`, the SMT ternary operator `C ? E1 : E2` (checking `C` to choose either `E1` or `E2`) further constrains `T` to be less than `T2 - R1`. The definition of `increaseClocks` also uses this SMT operator to represent the new values of the clocks:

```
eq increaseClocks(M, (L -> R1) ; CLOCKS, (L : PRE --> ... ) ; NET, T)
```

```
 = (L -> (active(M, L : PRE ...) ? R1 + T : R1 )) ;
   increaseClocks(M, CLOCKS, NET, T) .
```

The rule for applying a transition is defined as follows:

```
crl [applyTransition] :
    TS : M : ((L -> T) ; CLOCKS) : (L : PRE --> ...) ; NET)
 => tickOk : ((M - PRE) + POST) :  updateClocks(...) :
    (L : PRE --> ... ; NET) if (active(...) and (T in INTERVAL)) = true .
```

When applied, this rule adds new constraints asserting that the transition `L` can be fired (predicates `active` and `_in_`) and updates the state of the clocks:

```
eq updateClocks((L' -> R1) ; CLOCKS, INTERM-M, (L' : PRE --> ...); NET)
 = (L ->  PRE <= INTERM-M ? R1 : 0/1) ; updateClocks(...) .
```

Example 2. Let *net* and m_0 be the Maude terms representing, respectively, the PITPN and the initial marking shown in Fig. 1. The term *net* includes a variable `a:Real` representing the parameter a. The command

```
smt-search tickOk : m0 : initClocks(net) : net  =>*  TICK : M : CLOCKS : NET
  such that (a:Real >= 0/1 and not k-safe(1, M)) = true .
```

checks whether it is possible to reach a non-1-safe marking. Maude positively answers this question, with resulting accumulated constraint telling us that such a state is reachable (with 2 tokens in p_2) if `a:Real >= 4/1`.

Terms of sort `Marking` in $\mathcal{R}_1^{\mathbf{S}}$ may contain expressions with parameters (i.e., variables) of sort `Integer`. Let Λ_m denote the set of such parameters and $\pi_m : \Lambda_m \to \mathbb{N}$ a valuation function for them. We use m_s to denote a mapping from places to `Integer` expressions including parameter variables. Similarly, $clocks_s$ denotes a mapping from transitions to `Real` expressions (including variables). We write $\pi_m(m_s)$ to denote the ground term where the parameters in markings are replaced by the corresponding values $\pi_m(\lambda_i)$. Similarly for $\pi(clocks_s)$. We use $[\![\mathcal{N}]\!]_{\mathcal{R}_1^{\mathbf{S}}}$ to denotes the above rewriting logic representation of nets in $\mathcal{R}_1^{\mathbf{S}}$.

Recall that $t \in [\![\phi \parallel t_s]\!]$ is a ground instance, with a suitable ground substitution σ, of the constrained term $\phi \parallel t_s$. By construction, in $\mathcal{R}_1^{\mathbf{S}}$, if for all $t \in [\![\phi \parallel t_s]\!]$ all markings (sort `Integer`), clocks and parameters (`Real`) are non-negative numbers, then this is also the case for all reachable states from $\phi \parallel t_s$. Hence, there is a one-to-one correspondence for ground terms in $\mathcal{R}_1^{\mathbf{S}}$ satisfying that condition with terms in $\mathcal{R}_1$. We use $t \approx\in [\![\phi \parallel t_s]\!]$ to denote that there exists a $\mathcal{R}_1^{\mathbf{S}}$-term $t' \in [\![\phi \parallel t_s]\!]$ and t is its corresponding term in $\mathcal{R}_1$. Note that the ground substitution σ ($t' = t_s\sigma$) determines a parameter (π) and a marking (π_m) valuation consistent with the constraint ϕ ($\mathcal{I}_{\mathcal{E}_0} \models \phi\sigma$).

The following theorem states that the symbolic semantics matches all the behaviors resulting from a concrete execution of $\mathcal{R}_1$ with arbitrary parameter valuations π and π_m. Furthermore, for all symbolic executions with parameters, there exists a corresponding concrete execution where the parameters are instantiated with values consistent with the resulting accumulated constraint.

Theorem 2 (Soundness and Completeness). *Let $\mathcal{N}$ be a PITPN and ϕ be the constraint $\bigwedge_{J(t),t\in T}(0 \leq {}^{\uparrow}J(t) \leq J(t)^{\uparrow}) \wedge \bigwedge_{\lambda_i \in \Lambda_m}(0 \leq \lambda_i)$. **(1)** If $\phi \parallel t_s \rightsquigarrow^*_{\mathcal{R}_1^{\mathbf{S}}}$*

$\phi' \parallel t'_s$ *then, there exist* t' *and* $t \approx \in [\![\phi \parallel t_s]\!]$ *(and the corresponding valuations* π *and* π_m*) such that* $t \longrightarrow^*_{\mathcal{R}_1} t'$ *and* $t' \approx \in [\![\phi' \parallel t'_s]\!]$.
(2) *If* $t \longrightarrow^*_{\mathcal{R}_1} t'$ *with* $t \approx \in [\![\phi \parallel t_s]\!]$*, then there exists* $\phi' \parallel t'_s$ *such that* $t' \approx \in [\![\phi' \parallel t'_s]\!]$ *and* $\phi \parallel t_s \rightsquigarrow^*_{\mathcal{R}_1^S} \phi' \parallel t'_s$.

The symbolic counterpart $\mathcal{R}_2^{\mathbf{S}}$ of the theory $\mathcal{R}_2$ can be defined similarly.

4.2 A New Folding Method for Symbolic Reachability

Reachability analysis should terminate for both positive and negative queries for nets with finite parametric state-class graphs. However, this is not the case in analysis with $\mathcal{R}_1^{\mathbf{S}}$: the symbolic state space generated by `smt-search` is infinite even for such nets. The problem is that `smt-search` stops exploring from a symbolic state only if it has already visited the *same* state. Moreover, due to the fresh variables created in $\mathcal{R}_1^{\mathbf{S}}$, symbolic states representing the same set of concrete states are not the same, even though they are *logically* equivalent. For instance, if we use `smt-search` to try to show that the PITPN in Fig. 1 is 1-safe if $0 \leq a < 4$, such a command does not terminate. In fact, the command

```
smt-search tickOk : m0 : 0-clock(net) : net  =>*  TICK : M : CLOCKS : NET
such that (a:Real >= 0/1 and a:Real < 4 and M <= m0 and m0 <= M) = true .
```

searching for reachable states where $M = m_0$ will produce infinitely many equivalent solutions, where the state of the system is represented by different (new) variables but subject to equivalent constraints.

The usual approach for collapsing equivalent symbolic states in rewriting modulo SMT is subsumption [33]. Essentially, we stop searching from a symbolic state if, during the search, we have already encountered another state that subsumes ("contains") it. Let $U = \phi_u \parallel t_u$ and $V = \phi_v \parallel t_v$ be constrained terms. Then $U \sqsubseteq V$ if there is a substitution σ such that $t_u = t_v\sigma$ and the implication $\phi_u \Rightarrow \phi_v\sigma$ holds. In that case, $[\![U]\!] \subseteq [\![V]\!]$ and U does not need to be further explored if V has already been encountered.

Reachability analysis with folding is sound [7] but not necessarily complete (since $[\![U]\!] \subseteq [\![V]\!]$ does not imply $U \sqsubseteq V$) [33]. In fact, if we take two solutions U and V from the above `smt-search` command and use the Maude's command `match` to find the needed substitution σ, the SMT solver determines that the formula $\neg(\phi_u \Rightarrow \phi_v\sigma)$ is satisfiable (and therefore $\phi_u \Rightarrow \phi_v\sigma$ is not valid). Hence, a procedure based on checking this implication will fail to determine that $U \sqsubseteq V$.

The satisfiability witnesses of $\neg(\phi_u \Rightarrow \phi_v\sigma)$ show that the values for markings and clocks in the *current* time instant are equally constrained in ϕ_u and ϕ_v (and hence, they represent the same set of concrete states). However, since the variables representing the *current* state are different, the implication is falsifiable.

In the following, we propose a subsumption relation that solves the aforementioned problems. Let $\phi \parallel t$ be a constrained term where t is a term of sort `State`. Consider an *abstraction of built-ins* (t°, σ°) for t [47], where t° is as t but it replaces the expression e_i in markings $(p_i \mapsto e_i)$ and clocks $(l_i \rightarrow e_i)$ with new fresh variables. The substitution σ° is defined accordingly in such a way that

$t = t^\circ\sigma^\circ$. Let $\Psi_{\sigma^\circ} = \bigwedge_{x \in dom(\sigma^\circ)} x = x\sigma^\circ$, where $dom(\sigma) = \{x \in X \mid \sigma(x) \neq x\}$. We use $(\phi \parallel t) \Downarrow_{\mathtt{now}}$ to denote the constrained term $\phi \wedge \Psi_{\sigma^\circ} \parallel t^\circ$. Intuitively, $(\phi \parallel t) \Downarrow_{\mathtt{now}}$ replaces the clock values and markings with fresh variables, and the boolean expression Ψ_{σ° constrains those variables to take the values of clocks and the marking in t. From [47] we can show that $[\![\phi \parallel t]\!] = [\![(\phi \parallel t) \Downarrow_{\mathtt{now}}]\!]$.

Note that the only variables occurring in $(\phi \parallel t) \Downarrow_{\mathtt{now}}$ are those for parameters (if any) and the fresh variables in $dom(\sigma^\circ)$ (representing the symbolic state of clocks and markings). For a constrained term $\phi \parallel t$, we use $\exists(\phi \parallel t)$ to denote the formula $(\exists X)\phi$ where $X = vars(\phi) \setminus vars(t)$.

Definition 4 (Relation $\preceq$). *Let $U = \phi_u \parallel t_u$ and $V = \phi_v \parallel t_v$ be constrained terms where t_u and t_v are terms of sort* `State`*. Moreover, let $U \Downarrow_{now} = \phi'_u \parallel t'_u$ and $V \Downarrow_{now} = \phi'_v \parallel t'_v$, where $vars(t'_u) \cap vars(t'_v) = \emptyset$. We define the relation $\preceq$ on constrained terms so that $U \preceq V$ whenever there exists a substitution σ such that $t'_u = t'_v\sigma$ and the formula $\exists(U \Downarrow_{now}) \Rightarrow \exists(V \Downarrow_{now})\sigma$ is valid.*

The formula $\exists(U \Downarrow_{\mathtt{now}})$ hides the information about all the tick variables as well as the information about the clocks and markings in previous time instants. What we obtain is the information about the parameters, clocks and markings "now". Moreover, if t_u and t_v above are both `tickOk` states (or both `tickNotOk` states), and they represent two symbolic states of the same PITPN, then t'_u and t'_v always match (σ being the identity on the variables representing parameters and mapping the corresponding variables created in $V \Downarrow_{\mathtt{now}}$ and $U \Downarrow_{\mathtt{now}}$).

Theorem 3 (Soundness and Completeness) *Let U and V be constrained terms for two symbolic states of the same PITPN. Then, $[\![U]\!] \subseteq [\![V]\!]$ iff $U \preceq V$.*

We have implemented a new symbolic reachability analysis using the folding relation in Definition 4. Building on the theory transformation [47] implemented in Maude-SE, we transform $\mathcal{R}_1^{\mathbf{S}}$ into a rewrite theory $\mathcal{R}_1^{f\mathbf{S}}$ that rewrites terms of the form $S : \phi \parallel t$ where S is a set of constrained terms (the already visited states). Theory $\mathcal{R}_1^{f\mathbf{S}}$ defines an operator `subsumed`$(\phi \parallel t\ ,\ S)$ that reduces to `true`—by a call to the SMT solver Z3 for quantifier elimination and satisfiability checking—iff there exists $\phi' \parallel t' \in S$ s.t $\phi \parallel t \preceq \phi' \parallel t'$. Rules in $\mathcal{R}_1^{\mathbf{S}}$ are systematically transformed to add a further constraint: the new state on the right-hand side of the rule is not subsumed by a state in the set S.

In $\mathcal{R}_1^{f\mathbf{S}}$, for an initial constraint ϕ on the parameters, the Maude command `search [`n`,`m`] empty :` $\phi \parallel t$ `=>*` $S : \phi' \parallel t'$ `such that smtCheck(`ϕ' `and` Φ`)` d answers the question whether it is possible to reach a symbolic state that matches t' and satisfies the condition Φ. In the following, we use `init`(net, m_0, ϕ) to denote the term `empty` $: \phi \parallel$ `tickOk` $: m_0$ `:initClocks(`net`)` $: net$.

Example 3. Consider the PITPN in Fig. 1. Let m_0 be the marking in the figure and $\phi = 0 \leq a < 4$. The command

```
search init(net, m0, φ) =>* S : φ' || ( TICK : M : CLOCKS : NET )
   such that smtCheck(φ' and not k-safe(1,M)) .
```

terminates returning `No solution`, showing that the net is 1-safe if $0 \leq a < 4$.

If the set of reachable state classes in the symbolic semantics of $\mathcal{N}$ in [2] is finite, then so is the set of reachable symbolic states with the new folding method:

Corollary 1. *For any PITPN $\mathcal{N}$ and state class (M, D), if the transition system $(\mathcal{C}, (M, D), \Rightarrow)$ is finite, then so is* $\left(T_{\Sigma,State}, \mathit{init}(\mathcal{N}, M, D), \rightsquigarrow_{\mathcal{R}_1^{fs}}\right)$.

The new folding relation is applicable to any rewrite theory $\mathcal{R}$ that satisfies the requirements for rewriting with SMT [47], briefly explained in Sect. 2.

5 Parameter Synthesis and Symbolic Model Checking

This section shows how Maude-SE can be used for solving parameter synthesis problems, model checking the non-nested timed temporal logic properties supported by `Roméo` (in addition to LTL model checking), reasoning with *parametric* initial states, and analyzing nets with user-defined execution strategies.

5.1 Parameter Synthesis

A *state predicate* is a boolean expression on markings (e.g., `k-safe(1,` m`)`) and clocks (e.g., $c_1 < c_2$). EF-*synthesis* (resp. *safety synthesis* (AG$\neg\phi$)) is the problem of computing parameter values π such that *some* (resp. *no*) run of $\pi(\mathcal{N})$ reaches a state satisfying a given state predicate ϕ.

`search` in $\mathcal{R}_1^{fs}$ provides semi-decision procedures for solving these parameter synthesis problems (which are undecidable in general). As illustrated below, the resulting constraint computed by `search` can be used to synthesize the parameter values that allow such behaviors. The safety synthesis problem AG$\neg\phi$ can be solved by finding all solutions for EFϕ and negating the resulting constraint.

Example 4. The following command solves the EF-synthesis problem of finding values for a in Fig. 2 such that the net is not 1-safe, where $\phi = 0 \leq a$:

```
search [1] init(net, m0, φ) =>* S : PHI' || ( TICK : M : CLOCKS : NET )
   such that smtCheck(PHI' and not k-safe(1,M)) .
```

It returns one solution, and the resulting constraint ϕ', instantiating the pattern `PHI'`, can be used to extract the parameter values as follows. Let X be the set of SMT variables in ϕ' *not* representing parameters. A call to the quantifier elimination procedure (`qe`) of the SMT solver Z3 on the formula $\exists X.\phi'$ reduces to `a:Real >= 4/1`, giving us the desired values for the parameter a.

To solve the safety synthesis problem AG$\neg\phi$, we have used Maude's meta-programming facilities [18] to implement a command `safety-syn(`net`,` m_0`,` ϕ_0`,` ϕ`)` where m_0 is a marking, ϕ_0 a constraint on the parameters and ϕ a constraint involving the variables `M` and `CLOCKS` as in the `search` command in Example 4. This command iteratively calls `search` to find a state reachable from m_0, with

initial constraint ϕ_0, where ϕ does not hold. If such state is found, with accumulated constraint ϕ', the `search` command is invoked again with initial constraint $\phi_0 \wedge \neg\phi'$. This process stops when no more reachable states where ϕ does not hold are found, thus solving the $\mathsf{AG}\neg\phi$ synthesis problem.

Example 5. The following command synthesizes the values of the parameter a, so that $30 \leq a \leq 70$, that make the `scheduling` system in [6,50] 1-safe:

```
safety-syn(net, m0, a:Real >= 30/1 and a:Real <= 70/1, k-safe(1,M)) .
```

The first counterexample found assumes that $a \leq 48$. If $a > 48$, `search` does not find any counterexample. This is the same answer that `Roméo` found.

Since we can have `Integer` variables in initial markings, we can use Maude-SE to synthesize the initial markings that, e.g., make the net k-safe or alive:

Example 6. Consider a *parametric* initial marking m_s for the net in Fig. 1, with parameters x_1, x_2, and x_3 denoting the number of tokens in places p_1, p_2, and p_3, respectively, and the initial constraint ϕ_0 stating that $a \geq 0$ and $0 \leq x_i \leq 1$. The execution of the command execution of the command`safety-syn`$(net, m_s, \phi_0,$ `k-safe(1,M)`$)$ determines that the net is 1-safe when $x_1 = x_3 = 0$ and $0 \leq x_2 \leq 1$.

Analysis with Strategies. Maude's strategy facilities [17] allow us to analyze PITPNs whose executions follow some user-defined strategy:

Example 7. We execute the net in Fig. 1 with the following strategy `t3-first`: whenever transition t_3 and some other transition are enabled at the same time, then t_3 fires first. The following strategy definition (`sd`) specifies this strategy:

```
sd t3-first := ( applyTransition[L <- "t3"] or-else all )!
```

Running `srew init(`net`, `m_0`, `$a \geq 0$`) using t3-first` in $\mathcal{R}_1^{fS}$ finds all symbolic states reachable with this strategy, and all of them are 1-safe. Therefore, all parameter values $a \geq 0$ guarantee the desired property with this execution strategy.

5.2 Analyzing Temporal Properties

This section shows how Maude-SE can be used to analyze the temporal properties supported by `Roméo` [29], albeit in a few cases without parametric bounds in the temporal formulas. `Roméo` can analyze the following temporal properties:

$$\mathbf{Q}\,\phi\,\mathsf{U}_J\,\psi \;\mid\; \mathbf{Q}\mathsf{F}_J\,\phi \;\mid\; \mathbf{Q}\mathsf{G}_J\,\phi \;\mid\; \phi \rightsquigarrow_{\leq b} \psi$$

where $\mathbf{Q} \in \{\exists, \forall\}$, ϕ and ψ are *state predicates* on *markings*, and J is a time interval $[a, b]$, where a and/or b can be parameters and b can be ∞. For example, $\forall\mathsf{F}_{[a,b]}\,\phi$ says that in *each* path from the initial state, a marking satisfying ϕ is reachable in some time in $[a, b]$. The bounded response $\phi \rightsquigarrow_{\leq b} \psi$ says that each ϕ-marking *must* be followed by a ψ-marking within time b.

Since queries include time bounds, we use $\mathcal{R}_2^{fS}$, and `init`(net, m_0, ϕ) will denote the term `empty` $: \phi \parallel$ `tickOk` $: m_0$ `: initClocks`(net) `:` net `@ 0/1`.

State predicates, including inequalities on markings and clocks, and also a test whether the global clock is in a given interval are defined as follows:

```
ops _>=_ _>_ _<_ _<=_ _==_ : Place Integer -> Prop .
ops _>=_ _>_ _<_ _<=_ _==_ : Clock Real -> Prop .
op in-time : Interval -> Prop .
eq S : C || (TICK : M ; (P |-> N1) : CLOCKS : NET) @ G-CLOCK  |=  P >= N1'
 = smtCheck(C and N1 >= N1' ) . --- similarly for >, <=, < and ==
eq S : C || (TICK : M : CLOCKS : NET) @ G-CLOCK  |=  in-time INTERVAL
 = smtCheck(C and (G-CLOCK in INTERVAL )) .
```

Atomic propositions (`Prop`) are evaluated (`|=`) on symbolic states represented as constrained terms $S : \phi \parallel t$. Since they may contain variables, a call to the SMT solver (`smtCheck`) is needed to determine whether ϕ entails the proposition.

Some of the temporal formulas supported by `Roméo` can be easily verified using the reachability commands presented in the previous section. The property $\exists \mathsf{F}_{[a,b]}\, \psi$ can be verified using the command:

```
search [1] init(net, m0, φ) =>* S : PHI' || TICK : M : CLOCKS : NET @ G-CLOCK
   such that (STATE' |= ψ) and G-CLOCK in [a : b] .
```

where ϕ states that all parameters are non-negative numbers and $STATE'$ is the expression to the right of `=>*`. a and b can be variables representing parameters to be synthesized; and ψ can be an expression involving `CLOCKS`. For example,

```
search [1] init(net, m0, φ) =>*
 S' : PHI' || TICK : (M ; "p1" |-> P1) : (CLOCKS ; "t2" -> C2) : NET @ G-CLOCK
 such that (STATE' |= P1 > 1 /\ C2 < 2/1) and G-CLOCK in [a : b] .
```

checks whether it is possible to reach a marking, in some time in $[a, b]$, with more than one token in place p_1, when the value of the clock of transition t_2 is < 2.

The dual property $\forall \mathsf{G}_{[a,b]}\, \phi$ can be checked by analyzing $\exists \mathsf{F}_{[a,b]}\, \neg\, \phi$.

Example 8. Consider the PITPN in Example 5 with (interval) parameter $\phi = 30 \leq a \leq 70$. The property $\exists \mathsf{F}_{[b,b]}(\neg \textit{1-safe})$ can be verified with the following command, which determines that the parameter b satisfies $60 \leq b \leq 96$.

```
search [1] init(net, m0, φ) =>* S : PHI' || TICK : M : CLOCKS : NET @ G-CLOCK
  such that STATE' |= b:Real >= 0/1 and (G-CLOCK in [b:Real : b:Real])
                    and not (k-safe(1,M)) .
```

$\phi \rightsquigarrow_{\leq b} \psi$ can be verified using a simple theory transformation on $\mathcal{R}_{\mathbf{0}}^{\mathbf{S}}$ followed by reachability analysis. The theory transformation adds a new "clock," which is either `noClock` or `clock(`τ`)`, to the state. The latter represents the time (τ) since a ϕ-state was visited without having been followed by a ψ-state. The `applyTransition` rule is modified as follows: when the clock is `noClock` and the new marking satisfies $\phi \wedge \neg\psi$, this clock is set to `clock(0)`, and when a ψ-marking is reached, the clock is set to `noClock`. The `tick` rule updates `clock(T1)` to `clock(T1 + T)` and leaves `noClock` unchanged. $\phi \rightsquigarrow_{\leq b} \psi$ can be checked by searching for a "bad" state with "clock" `clock(T)` where `T` $> b$. See [6] for details.

Reachability analysis cannot be used to analyze the other properties supported by `Roméo` ($\mathbf{Q}\, \phi\, \mathsf{U}_J\, \psi$, and $\forall \mathsf{F}_J\, \phi$ and its dual $\exists \mathsf{G}_J\, \phi$). We combine Maude's explicit-state model checker and SMT solving to solve these (and other) queries.

The timed temporal operators can be defined on top of the (untimed) LTL temporal operators in Maude (`<>`, `[]` and `U`) :

```
ops  <_>_  [_]_ :  Interval Prop -> Formula . --- F_J φ and G_J φ
op _U__ : Prop Interval Prop -> Formula .     --- φ U_j ψ
eq < INTERVAL > PR1 = <> (PR1 /\ in-time INTERVAL) .
eq [ INTERVAL ] PR1 = ~ (< INTERVAL > (~ PR1)) .
eq PR1 U INTERVAL PR2 = PR1 U (PR2 /\ in-time INTERVAL) .
```

For this fragment of non-nested timed temporal logic formulas, universally quantified properties can be model checked directly by Maude; for $\exists\Phi$ it is enough to model check $\neg\Phi$: any counterexample to this is a witness for $\exists\Phi$, and vice versa.

6 Benchmarking

We have compared the performance of Maude-with-SMT analysis with that of `Roméo` on three case studies: the `producer-consumer` [52] system in Fig. 1, the `scheduling` system in [50], and the `tutorial` system taken from the `Roméo` website. We modified `tutorial` to produce two tokens in the loop-back, which leads to infinite behaviors. We compared the performance of solving the synthesis problem $\mathsf{EF}(p > n)$ (place p holds more than n tokens), for different p and n, and of checking whether the net is 1-safe. In each experiment, Maude was executed with two different SMT solvers: Yices and Z3. The benchmarking data are available in the repository [5] and in the technical report [6].

The results show that using Maude with Yices is faster than using it with Z3. For negative queries, as expected, $\mathcal{R}_{\mathbf{0}}^{\mathbf{S}}$ and $\mathcal{R}_{\mathbf{1}}^{\mathbf{S}}$ time out (set to 10 minutes), while $\mathcal{R}_{\mathbf{1}}^{f\mathbf{S}}$ (which uses folding) completes the analysis before the timeout.

Maude-SE outperforms `Roméo` in some reachability queries, and sometimes our analysis terminates when `Roméo` does not, which may happen when the search order leads `Roméo` to explore an infinite branch with an unbounded marking.

7 Related Work

Tool Support for Parametric Time Petri Nets. We are not aware of any tool for analyzing parametric time(d) Petri nets other than `Roméo` [29].

Petri Nets in Rewriting Logic. Formalizing Petri nets algebraically [34] partly inspired rewriting logic. Different kinds of Petri nets are given a rewriting logic semantics in [48], and in [40] for timed nets. In contrast to our paper, these papers focus on the semantics of such nets, and do not consider execution and analysis (or inhibitor arcs or parameters). Capra [14,15], Padberg and Schultz [45], and Barbosa et al. [12] use Maude to formalize dynamically reconfigurable Petri nets and I/O Petri nets. In contrast to our work, these papers target untimed and non-parametric nets, and do not focus on formal analysis, but only show examples of standard (explicit-state) `search` and LTL model checking.

Symbolic Methods for Real-Time Systems in Maude. We develop symbolic analysis methods for parametric time automata (PTA) in [4]. The differences with the current paper include: PTAs are very simple structures compared to PITPNs (with inhibitor arcs, no bounds on the number of tokens in a state), so the

semantics of PITPNs is more sophisticated than the one for PTAs, which does not use "structured" states, equations, or user-defined functions; defining a new rewrite theory for each PTA in [4] compared to having a single rewrite theory for all nets in this work; obtaining desired symbolic reachability properties using "standard" folding methods for PTAs compared to having to develop a new folding mechanism for PITPNs; analysis in [4] does not include temporal logic model checking; and so on. In addition, a number of real-time systems have been formally analyzed using rewriting with SMT, including PLC ST programs [26], virtually synchronous cyber-physical systems [23–25], and soft agents [35]. These papers differ from our work in that they use guarded terms [10,11] for state-space reduction instead of folding, and do not consider parameter synthesis problems.

8 Concluding Remarks

We have provided a "concrete" rewriting logic semantics for PITPNs, and proved that this semantics is bisimilar to the semantics of such nets in [49]. We then *systematically transformed* this non-executable "Real-Time Maude-style" model into a "symbolic" rewrite model which is amenable to sound and complete symbolic analysis for dense-time systems using Maude combined with SMT solving.

We have shown how almost all analysis and parameter synthesis supported by the PITPN tool `Roméo` can be done using Maude-with-SMT. We have also shown how Maude-with-SMT can provide additional capabilities for PITPNs, including synthesizing initial markings (and not just firing bounds) from *parametric* initial markings so that desired properties are satisfied, full LTL model checking, and analysis with user-defined execution strategies. We developed a new "folding" method for symbolic states, so that symbolic reachability analysis using Maude-with-SMT terminates whenever the corresponding `Roméo` analysis terminates.

Our benchmarking shows that our symbolic methods using Maude combined with the SMT solver Yices in some cases outperforms `Roméo`, whereas Maude with Z3 is significantly slower.

This paper has not only provided new features for PITPNs. It has also shown that even a model like our Real-Time Maude-inspired PITPN interpreter—with functions, equations, and unbounded markings—can easily be turned into a symbolic rewrite theory for which Maude-with-SMT provides very useful sound and complete analyses even for dense-time systems.

Acknowledgments. We thank the anonymous reviewers for their insightful comments. Arias, Olarte, Ölveczky, Petrucci, and Rømming acknowledge support from CNRS INS2I project ESPRiTS and the PHC project Aurora AESIR. Bae was supported by the NRF grants funded by the Korea government (No. 2021R1A5A1021944 and No. 2022R1F1A1074550).

References

1. AlTurki, M., Dhurjati, D., Yu, D., Chander, A., Inamura, H.: Formal specification and analysis of timing properties in software systems. In: Chechik, M., Wirsing, M. (eds.) FASE 2009. LNCS, vol. 5503, pp. 262–277. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00593-0_18
2. André, É., Pellegrino, G., Petrucci, L.: Precise robustness analysis of time Petri nets with inhibitor arcs. In: Braberman, V., Fribourg, L. (eds.) FORMATS 2013. LNCS, vol. 8053, pp. 1–15. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40229-6_1
3. Andreychenko, A., Magnin, M., Inoue, K.: Analyzing resilience properties in oscillatory biological systems using parametric model checking. Biosystems **149**, 50–58 (2016)
4. Arias, J., Bae, K., Olarte, C., Ölveczky, P.C., Petrucci, L., Rømming, F.: Rewriting logic semantics and symbolic analysis for parametric timed automata. In: Proceedings of the 8th ACM SIGPLAN International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS 2022), pp. 3–15. ACM (2022)
5. Arias, J., Bae, K., Olarte, C., Ölveczky, P.C., Petrucci, L., Rømming, F.: PITPN2Maude (2023). https://depot.lipn.univ-paris13.fr/arias/pitpn2maude
6. Arias, J., Bae, K., Olarte, C., Ölveczky, P.C., Petrucci, L., Rømming, F.: Symbolic analysis and parameter synthesis for time Petri nets using Maude and SMT solving (2023). https://doi.org/10.48550/ARXIV.2303.08929
7. Bae, K., Escobar, S., Meseguer, J.: Abstract logical model checking of infinite-state systems using narrowing. In: Rewriting Techniques and Applications (RTA 2013). LIPIcs, vol. 21, pp. 81–96. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2013)
8. Bae, K., Krisiloff, J., Meseguer, J., Ölveczky, P.C.: Designing and verifying distributed cyber-physical systems using Multirate PALS: an airplane turning control system case study. Sci. Comput. Program. **103**, 13–50 (2015). https://doi.org/10.1016/j.scico.2014.09.011
9. Bae, K., Ölveczky, P.C., Feng, T.H., Lee, E.A., Tripakis, S.: Verifying hierarchical Ptolemy II discrete-event models using Real-Time Maude. Sci. Comput. Program. **77**(12), 1235–1271 (2012)
10. Bae, K., Rocha, C.: Guarded terms for rewriting modulo SMT. In: Proença, J., Lumpe, M. (eds.) FACS 2017. LNCS, vol. 10487, pp. 78–97. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68034-7_5
11. Bae, K., Rocha, C.: Symbolic state space reduction with guarded terms for rewriting modulo SMT. Sci. Comput. Program. **178**, 20–42 (2019)
12. Barbosa, P., et al.: SysVeritas: a framework for verifying IOPT nets and execution semantics within embedded systems design. In: Camarinha-Matos, L.M. (ed.) DoCEIS 2011. IAICT, vol. 349, pp. 256–265. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19170-1_28
13. Bobba, R., et al.: Survivability: design, formal modeling, and validation of cloud storage systems using Maude. In: Assured Cloud Computing, Chap. 2, pp. 10–48. Wiley (2018)
14. Capra, L.: Canonization of reconfigurable PT nets in Maude. In: Lin, A.W., Zetzsche, G., Potapov, I. (eds.) Reachability Problems. RP 2022. LNCS, vol. 13608, pp. 160–177. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-19135-0_11

15. Capra, L.: Rewriting logic and Petri nets: a natural model for reconfigurable distributed systems. In: Bapi, R., Kulkarni, S., Mohalik, S., Peri, S. (eds.) ICDCIT 2022. LNCS, vol. 13145, pp. 140–156. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-94876-4_9
16. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Amsterdam/Cambridge (2001)
17. Clavel, M., et al.: Maude Manual (Version 3.2.1). SRI International (2022). http://maude.cs.illinois.edu
18. Clavel, M., et al.: All About Maude - A High-Performance Logical Framework. LNCS, vol. 4350. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71999-1
19. Coullon, H., Jard, C., Lime, D.: Integrated model-checking for the design of safe and efficient distributed software commissioning. In: Ahrendt, W., Tapia Tarifa, S.L. (eds.) IFM 2019. LNCS, vol. 11918, pp. 120–137. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34968-4_7
20. Grabiec, B., Traonouez, L.-M., Jard, C., Lime, D., Roux, O.H.: Diagnosis using unfoldings of parametric time Petri nets. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 137–151. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15297-9_12
21. Grov, J., Ölveczky, P.C.: Formal modeling and analysis of Google's Megastore in Real-Time Maude. In: Iida, S., Meseguer, J., Ogata, K. (eds.) Specification, Algebra, and Software. LNCS, vol. 8373, pp. 494–519. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54624-2_25
22. Jensen, K., Kristensen, L.M.: Coloured Petri Nets - Modelling and Validation of Concurrent Systems. Springer, Heidelberg (2009). https://doi.org/10.1007/b95112
23. Lee, J., Bae, K., Ölveczky, P.C.: An extension of HybridSynchAADL and its application to collaborating autonomous UAVs. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Adaptation and Learning (ISoLA 2022). LNCS, vol. 13703, pp. 47–64. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-19759-8_4
24. Lee, J., Bae, K., Ölveczky, P.C., Kim, S., Kang, M.: Modeling and formal analysis of virtually synchronous cyber-physical systems in AADL. Int. J. Software Tools Technol. Transf. **24**(6), 911–948 (2022)
25. Lee, J., Kim, S., Bae, K., Ölveczky, P.C.: HybridSynchAADL: modeling and formal analysis of virtually synchronous CPSs in AADL. In: Silva, A., Leino, K.R.M. (eds.) CAV 2021. LNCS, vol. 12759, pp. 491–504. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81685-8_23
26. Lee, J., Kim, S., Bae, K.: Bounded model checking of PLC ST programs using rewriting modulo SMT. In: Proceedings of the 8th ACM SIGPLAN International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS 2022), pp. 56–67. ACM (2022)
27. Lien, E., Ölveczky, P.C.: Formal modeling and analysis of an IETF multicast protocol. In: Seventh IEEE International Conference on Software Engineering and Formal Methods (SEFM 2009), pp. 273–282. IEEE Computer Society (2009)
28. Lime, D., Roux, O.H., Seidner, C.: Cost problems for parametric time Petri nets. Fundam. Informaticae **183**(1-2), 97–123 (2021). https://doi.org/10.3233/FI-2021-2083
29. Lime, D., Roux, O.H., Seidner, C., Traonouez, L.-M.: Romeo: a parametric model-checker for Petri nets with stopwatches. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 54–57. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00768-2_6

30. Merlin, P.M.: A study of the recoverability of computing systems. Ph.D. thesis, University of California, Irvine, CA, USA (1974)
31. Meseguer, J.: Conditional rewriting logic as a unified model of concurrency. Theor. Comput. Sci. **96**(1), 73–155 (1992)
32. Meseguer, J.: Twenty years of rewriting logic. J. Log. Algebraic Methods Program. **81**(7–8), 721–781 (2012)
33. Meseguer, J.: Generalized rewrite theories, coherence completion, and symbolic methods. J. Log. Algebraic Methods Program. 110 (2020)
34. Meseguer, J., Montanari, U.: Petri nets are monoids. Inform. Comput. **88**(2), 105–155 (1990)
35. Nigam, V., Talcott, C.L.: Automating safety proofs about cyber-physical systems using rewriting modulo SMT. In: Bae, K. (ed.) Rewriting Logic and Its Applications (WRLA 2022). LNCS, vol. 13252, pp. 212–229. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-12441-9_11
36. Ölveczky, P.C.: Semantics, simulation, and formal analysis of modeling languages for embedded systems in Real-Time Maude. In: Agha, G., Danvy, O., Meseguer, J. (eds.) Formal Modeling: Actors, Open Systems, Biological Systems. LNCS, vol. 7000, pp. 368–402. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24933-4_19
37. Ölveczky, P.C.: Real-Time Maude and its applications. In: Escobar, S. (ed.) WRLA 2014. LNCS, vol. 8663, pp. 42–79. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12904-4_3
38. Ölveczky, P.C., Boronat, A., Meseguer, J.: Formal semantics and analysis of behavioral AADL models in Real-Time Maude. In: Hatcliff, J., Zucca, E. (eds.) FMOOD-S/FORTE -2010. LNCS, vol. 6117, pp. 47–62. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13464-7_5
39. Ölveczky, P.C., Caccamo, M.: Formal simulation and analysis of the CASH scheduling algorithm in Real-Time Maude. In: Baresi, L., Heckel, R. (eds.) FASE 2006. LNCS, vol. 3922, pp. 357–372. Springer, Heidelberg (2006). https://doi.org/10.1007/11693017_26
40. Ölveczky, P.C., Meseguer, J.: Specification of real-time and hybrid systems in rewriting logic. Theor. Comput. Sci. **285**(2), 359–405 (2002)
41. Ölveczky, P.C., Meseguer, J.: Abstraction and completeness for Real-Time Maude. In: 6th International Workshop on Rewriting Logic and its Applications (WRLA 2006). Electronic Notes in Theoretical Computer Science, vol. 174, pp. 5–27. Elsevier (2006)
42. Ölveczky, P.C., Meseguer, J.: Semantics and pragmatics of Real-Time Maude. High. Order Symb. Comput. **20**(1–2), 161–196 (2007)
43. Ölveczky, P.C., Meseguer, J.: The Real-Time Maude tool. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 332–336. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_23
44. Ölveczky, P.C., Meseguer, J., Talcott, C.L.: Specification and analysis of the AER/NCA active network protocol suite in Real-Time Maude. Formal Methods Syst. Des. **29**(3), 253–293 (2006)
45. Padberg, J., Schulz, A.: Model checking reconfigurable Petri nets with Maude. In: Echahed, R., Minas, M. (eds.) ICGT 2016. LNCS, vol. 9761, pp. 54–70. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40530-8_4
46. Parquier, B., et al.: Applying parametric model-checking techniques for reusing real-time critical systems. In: Artho, C., Ölveczky, P.C. (eds.) FTSCS 2016. CCIS, vol. 694, pp. 129–144. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-53946-1_8

47. Rocha, C., Meseguer, J., Muñoz, C.A.: Rewriting modulo SMT and open system analysis. J. Log. Algebraic Methods Program. **86**(1), 269–297 (2017)
48. Stehr, M.-O., Meseguer, J., Ölveczky, P.C.: Rewriting logic as a unifying framework for Petri nets. In: Ehrig, H., Padberg, J., Juhás, G., Rozenberg, G. (eds.) Unifying Petri Nets. LNCS, vol. 2128, pp. 250–303. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45541-8_9
49. Traonouez, L.-M., Lime, D., Roux, O.H.: Parametric model-checking of time Petri nets with stopwatches using the state-class graph. In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 280–294. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85778-5_20
50. Traonouez, L., Lime, D., Roux, O.H.: Parametric model-checking of stopwatch Petri nets. J. Univers. Comput. Sci. **15**(17), 3273–3304 (2009)
51. Vernadat, F., Berthomieu, B.: State space abstractions for time Petri nets. In: Son, S.H., Lee, I., Leung, J.Y. (eds.) Handbook of Real-Time and Embedded Systems. Chapman and Hall/CRC (2007)
52. Wang, J.: Time Petri nets. In: Timed Petri Nets: Theory and Application, pp. 63–123. Springer, Cham (1998)
53. Yu, G., Bae, K.: Maude-SE: a tight integration of Maude and SMT solvers. In: Preliminary Proceedings of WRLA@ETAPS, pp. 220–232 (2020)