Linear Parametric Model Checking of Timed Automata*

Thomas Hune[1], Judi Romijn[2], Mariëlle Stoelinga[2], and Frits Vaandrager[2]

[1] BRICS, University of Århus, Denmark
baris@brics.dk
[2] Computing Science Institute, University of Nijmegen
[judi,marielle,fvaan]@cs.kun.nl

Abstract. We present an extension of the model checker UPPAAL capable of synthesize linear parameter constraints for the correctness of parametric timed automata. The symbolic representation of the (parametric) state-space is shown to be correct. A second contribution of this paper is the identification of a subclass of parametric timed automata (L/U automata), for which the emptiness problem is decidable, contrary to the full class where it is know to be undecidable. Also we present a number of lemmas enabling the verification effort to be reduced for L/U automata in some cases. We illustrate our approach by deriving linear parameter constraints for a number of well-known case studies from the literature (exhibiting a flaw in a published paper).

1 Introduction

During the last decade, there has been enormous progress in the area of timed model checking. Tools such as UPPAAL[11], KRONOS [5], and PMC [12] are now routinely used for industrial case studies. A disadvantage of the traditional approaches is, however, that they can only be used to verify concrete timing properties: one has to provide the values of all timing parameters that occur in the system. For practical purposes, one is often interested in deriving the (symbolic) constraints on the parameters that ensure correctness. The process of manually finding and proving such results is very time consuming and error prone (we have discovered minor errors in the two examples we have been looking at). Therefore tool support for deriving the constraints *automatically* is very important.

In this paper, we study a parameterized extension of timed automata, as well as a corresponding extension of the forward reachability algorithm. We show the theoretical correctness of our approach, and its feasibility by application to some non-trivial case studies. For this purpose, we have implemented a prototype extension of UPPAAL, an efficient real-time model checking tool [11]. The algorithm we propose and have implemented is a semi-decision algorithm which will not terminate in all cases. In [2] the problem of synthesizing values for parameters

* Research supported by Esprit Project 26270, Verification of Hybrid Systems (VHS), and PROGRESS Project TES4199, Verification of Hard and Softly Timed Systems (HaaST). This work was initiated during a visit of the first author to Nijmegen.

such that a property is satisfied, was shown to be undecidable, so this is the best we can hope for.

A second contribution of this paper is the identification of a subclass of parameterized timed automata, called *lower bound/upper bound (L/U) automata*, which appears to be sufficiently expressive from a practical perspective, while it also has nice theoretical properties. Most importantly, we show that the emptyness question for parametric timed automata shown to be undecidable in [2], is decidable for L/U automata. We also establish a number of lemmas which allow one to reduce the number of parameters when tackling specific verification questions for L/U automata. The application of these lemmas has already reduced the verification effort drastically in some of our experiments.

Our attempt at automatic verification of parameterized real-time models is not the only one. Henzinger et al. aim at solving a more general problem with HyTech [9], a tool for model checking hybrid automata, exploring the state-space either by partition refinement, or forward reachability. The tool has been applied successfully on relatively small examples such as a railway gate controller. Experience so far has shown that HyTech cannot cope with larger examples, such as the ones considered in this paper.

Toetenel et al. [12] have made an extension of the PMC real-time model checking tool [4] called LPMC. LPMC is restricted to linear parameter constraints as is our approach, and uses the partition refinement method, like HyTech. Other differences with our approach are that LPMC also allows for the comparison of non-clock variables to parameter constraints, and for more general specification properties (full TCTL with fairness assumptions). Since LPMC is a quite recent tool, not many applications have been presented yet. However, a model of the IEEE 1394 root contention protocol inspired by [13] has been successfully analyzed in [4].

A more general attempt than LPMC and our Uppaal extension has been made by Annichini et al. [3]. They have constructed and implemented a method which allows for non-linear parameter constraints, and uses heavier, third-party, machinery to solve the arising non-linear constraint comparisons. Independently, we have used the same data-structure (a direct extension of DBMs [8]) for the symbolic representation of the state space, as in [3]. For speeding up the exploration, a method for guessing the effect of control loops in the model is presented. It appears that this helps termination of the method, but it is unclear under what circumstances this technique can or cannot be used. The feasibility of this approach has been shown on a few rather small case studies.

The remainder of this paper is organized as follows. Section 2 introduces the notion of parametric timed automata. Section 3 gives the symbolic semantics, which is the basis for our model checking algorithm, presented in Section 3.5. Section 4 is an intermezzo that states some helpful lemmas and decidability results on an interesting subclass. Finally, Section 5 reports on experiments with our tool. For lack of space, some technical details and all proofs have been omitted, which can be found in the full version of this paper [10].

2 Parametric Timed Automata

2.1 Parameters and Constraints

Throughout this paper, we assume a fixed set of *parameters* $P = \{p_1, \ldots, p_n\}$. A *linear expression* e is either an expression of the form $t_1 p_1 + \cdots + t_n p_n + t_0$, where $t_0, \ldots, t_n \in \mathsf{Z}$, or ∞. We write E to denote the set of all linear expressions. A *constraint* is an inequality of the form $e \sim e'$, with e, e' linear expressions and $\sim \in \{<, \leq, >, \geq\}$. The *negation* of constraint c, notation $\neg c$, is obtained by replacing relation signs $<$, $\leq$, $>$, $\geq$ by $\geq$, $>$, $\leq$, $<$, respectively. A *(parameter) valuation* is a function $v : P \to \mathsf{R}^{\geq 0}$ assigning a nonnegative real value to each parameter. There is a one-to-one correspondence between valuations and points in $(\mathsf{R}^{\geq 0})^n$. In fact we often identify a valuation v with the point $(v(p_1), \ldots, v(p_n)) \in (\mathsf{R}^{\geq 0})^n$.

If e is a linear expression and v is a valuation, then $e[v]$ denotes the expression obtained by replacing each parameter p in e with $v(p)$. Likewise, we define $c[v]$ for c a constraint. Valuation v *satisfies* constraint c, notation $v \models c$, if $c[v]$ evaluates to true. The *semantics* of a constraint c, notation $[\![c]\!]$, is the set of valuations (points in $(\mathsf{R}^{\geq 0})^n$) that satisfy c. A finite set of constraints C is called a *constraint set*. A valuation *satisfies* a constraint set if it satisfies each constraint in the set. The *semantics* of a constraint set C is given by $[\![C]\!] := \bigcap_{c \in C} [\![c]\!]$. We write $\top$ to denote any constraint set with $[\![\top]\!] = (\mathsf{R}^{\geq 0})^n$, for instance the empty set. We use $\perp$ to denote any constraint set with $[\![\perp]\!] = \emptyset$, for instance the constraint set $\{c, \neg c\}$, for some arbitrary c.

Constraint c *covers* constraint set C, notation $C \models c$, iff $[\![C]\!] \subseteq [\![c]\!]$. Constraint set C *is split by* constraint c iff neither $C \models c$ nor $C \models \neg c$.

During the analysis questions arise of the kind: given a constraint set C and a constraint c, does c hold, i.e., does constraint c cover C? A split occurs when c holds for some valuations in the semantics of C and $\neg c$ holds for some other valuations. We will not discuss methods for answering such questions: in our implementation we use an oracle to compute the following function.

$$\mathcal{O}(c, C) = \begin{cases} \text{yes} & \textit{if } C \models c \\ \text{no} & \textit{if } C \models \neg c \\ \text{split} & \textit{otherwise} \end{cases}$$

Observe that using the oracle, we can easily decide semantic inclusion between constraint sets: $[\![C]\!] \subseteq [\![C']\!]$ iff $\forall c' \in C' : \mathcal{O}(c', C) = \text{yes}$. The oracle that we use is a linear programming (LP) solver that was kindly provided to us by the authors of [4], who built it for their LPMC model checking tool.

2.2 Parametric Timed Automata

Throughout this paper, we assume a fixed set of clocks $X = \{x_0, \ldots, x_m\}$ and a fixed set of actions $A = \{a_1, \ldots, a_k\}$. The special clock x_0, which is called the *zero clock*, always has the value 0.

A *simple guard* is an expression f of the form $x_i - x_j \prec e$, where x_i, x_j are clocks, $\prec \in \{<, \leq\}$, and e is a linear expression. We say that f is *proper* if $i \neq j$. We define a *guard* to be a (finite) conjunction of simple guards. We let g range over guards and write G to denote the set of guards. A *clock valuation* is a function $w : X \to \mathsf{R}^{\geq 0}$ assigning a nonnegative real value to each clock, such that $w(x_0) = 0$. We will identify a clock valuation w with the point $(w(x_0), \ldots, w(x_m)) \in (\mathsf{R}^{\geq 0})^{m+1}$. Let g be a guard, v a parameter valuation, and w a clock valuation. Then $g[v, w]$ denotes the expression obtained by replacing each parameter p with $v(p)$, and each clock x with $w(x)$. A pair (v, w) of a parameter valuation and a clock valuation *satisfies* a guard g, notation $(v, w) \models g$, if $g[v, w]$ evaluates to true. The *semantics* of a guard g, notation $[\![g]\!]$, is the set of pairs (v, w) such that $(v, w) \models g$.

A *reset* is an expression of the form, $x_i := b$ where $i \neq 0$ and $b \in \mathsf{N}$. A *reset set* is a set of resets containing at most one reset for each clock. The set of reset sets is denoted by R.

We now define an extension of timed automata [1,15] called parametric timed automata. Similar models have been presented in [2,3,4].

Definition 1 (PTA). *A* parametric timed automaton (PTA) *over set of clocks X, set of actions A, and set of parameters P, is a quadruple $\mathcal{A} = (Q, q_0, \to, I)$, where Q is a finite set of* locations, *$q_0 \in Q$ is the* initial location, *$\to \subseteq Q \times A \times G \times R \times Q$ is a finite* transition relation, *and function $I : Q \to G$ assigns an* invariant *to each location. We abbreviate a $(q, a, g, r, q') \in \to$ consisting of a source location, an action, a guard, a reset set, and a target location as $q \xrightarrow{a,g,r} q'$. For a simple guard $x_i - x_j \prec e$ to be used in an invariant it must be the case that $x_j = x_0$, that is, the simple guard represents an upper bound on a clock.*

Example 1. A parametric timed automaton with clocks x, y and parameters p, q can be seen in Fig. 1. The initial state is $S0$ which has invariant $x \leq p$, and the transition from the initial location to $S1$ has guard $y \geq q$ and reset set $x := 0$. There are no actions on the transitions. Initially the transition from $S0$ to $S1$ is only enabled if $p \leq q$, otherwise the system will be deadlocked.

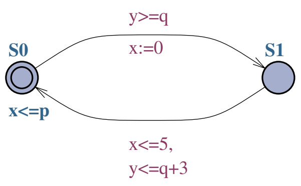

Fig. 1. A parametric timed automaton

To define the semantics of PTAs, we require two auxiliary operations on clock valuations. For clock valuation w and nonnegative real number d, $w + d$ is the clock valuation that adds to each clock (except x_0) a delay d. For clock valuation

w and reset set r, $w[r]$ is the clock valuation that resets clocks according to r.

$$(w+d)(x) = \begin{cases} 0 & \text{if } x = x_0 \\ w(x)+d & \text{otherwise} \end{cases} \qquad (w[r])(x) = \begin{cases} b & \text{if } x := b \in r \\ w(x) & \text{otherwise.} \end{cases}$$

Definition 2 (Concrete semantics). *Let* $\mathcal{A} = (Q, q_0, \rightarrow, I)$ *be a PTA and* v *be a parameter valuation. The* concrete semantics of $\mathcal{A}$ under v, *notation* $[\![\mathcal{A}]\!]_v$, *is the labeled transition system (LTS)* $(S, S_0, \rightarrow)$ *over* $A \cup \mathsf{R}^{\geq 0}$ *where*

$$S = \{(q,w) \in Q \times (X \rightarrow \mathsf{R}^{\geq 0}) \mid w(x_0) = 0 \wedge (v,w) \models I(q)\},$$
$$S_0 = \{(q,w) \in S \mid q = q_0 \wedge w = \lambda x.0\},$$

and transition predicate $\rightarrow$ *is specified by the following two rules, for all* (q,w), $(q',w') \in S$, $d \geq 0$ *and* $a \in A$,

- $(q,w) \xrightarrow{d} (q',w')$ *if* $q = q'$ *and* $w' = w + d$.
- $(q,w) \xrightarrow{a} (q',w')$ *if* $\exists g, r : q \xrightarrow{a,g,r} q' \wedge (v,w) \models g \wedge w' = w[r]$.

2.3 The Problem

In its current version, UPPAAL is able to check for reachability properties, in particular whether certain combinations of locations and constrains on clock variables are reachable from the initial configuration. Our parameterized extension of UPPAAL handles exactly the same properties. However, rather than just telling whether a property holds or not, our tool looks for constraints on the parameters which ensure that the property holds.

Definition 3 (Properties). *The sets of* system properties *and* state formulas *are defined by, respectively,*

$$\psi ::= \forall\Box\phi \mid \exists\Diamond\phi \qquad\qquad \phi ::= x - y \prec b \mid q \mid \neg\phi \mid \phi \wedge \phi$$

where $x, y \in X$, $b \in \mathsf{N}$ *and* $q \in Q$. *Let* $\mathcal{A}$ *be a PTA,* v *a parameter valuation,* s *a state of* $[\![\mathcal{A}]\!]_v$, *and* ϕ *a state formula. We write* $s \models \phi$ *if* ϕ *holds in state* s, *we write* $[\![\mathcal{A}]\!]_v \models \forall\Box\phi$ *if* ϕ *holds in all reachable states of* $[\![\mathcal{A}]\!]_v$, *and we write* $[\![\mathcal{A}]\!]_v \models \exists\Diamond\phi$ *if* ϕ *holds for some reachable state of* $[\![\mathcal{A}]\!]_v$.

The problem that we address in this paper can now be stated as follows: *Given a parametric timed automaton* $\mathcal{A}$ *and a system property* ψ, *compute the set of parameter valuations* v *for which* $[\![\mathcal{A}]\!]_v \models \psi$.

Timed automata [1,15] arise as a special case of PTAs for which the set P of parameters is empty. If $\mathcal{A}$ is a PTA and v is a parameter valuation, then the structure $\mathcal{A}[v]$ that is obtained by replacing all linear expressions e that occur in $\mathcal{A}$ by $e[v]$ is a timed automaton.[1] It is easy to see that in general $[\![\mathcal{A}]\!]_v = [\![\mathcal{A}[v]]\!]$. Since the reachability problem for timed automata is decidable [1], this implies that, for any $\mathcal{A}$, integer valued v and ψ, $[\![\mathcal{A}]\!]_v \models \psi$ is decidable.

[1] Strictly speaking, $\mathcal{A}[v]$ is only a timed automaton if v assigns an integer to each parameter.

3 Symbolic State Exploration

Our aim is to use basically the same algorithm for parametric time model checking as for timed model checking. We represent sets of states symbolically in a similar way and support the same operations used for timed model checking. In the nonparametrized case, sets of states can be efficiently represented using matrices [8]. Similarly, in this paper we represent sets of states symbolically as *(constrained) parametric difference-bound matrices.*

3.1 Parametric Difference-Bound Matrices

In the nonparametrized case, a difference-bound matrix is a $(m+1) \times (m+1)$ matrix whose entries are elements from $(\mathsf{Z} \cup \{\infty\}) \times \{0,1\}$. An entry $(c,1)$ for D_{ij} denotes a nonstrict bound $x_i - x_j \leq c$, whereas an entry $(c,0)$ denotes a strict bound $x_i - x_j < c$. Here, instead of using integers in the entries, we will use linear expressions over the parameters. Also, we find it convenient to view the matrix slightly more abstractly as a set of guards.

Definition 4 (PDBM). *A* parametric difference-bound matrix (PDBM) *is a set D which contains, for all $0 \leq i,j \leq m$, a simple guard D_{ij} of the form $x_i - x_j \prec_{ij} e_{ij}$. We require that, for all i, D_{ii} is of the form $x_i - x_i \leq 0$. Given a parameter valuation v, the* semantics *of D is defined by $[\![D]\!]_v = [\![\bigwedge_{i,j} D_{ij}]\!]_v$. We say that D is* satisfiable *for v if $[\![D]\!]_v$ is nonempty. If f is a proper guard of the form $x_i - x_j \prec e$ then we write $D[f]$ for the PDBM obtained from D by replacing D_{ij} by f. If i,j are indices then we write D^{ij} for the pair $(e_{ij}, \prec_{ij})$; we call D^{ij}* a bound *of D. Clearly, a PDBM is fully determined by its bounds.*

Definition 5 (Constrained PDBM). *A* constrained PDBM *is a pair (C,D) where C is a constraint set and D is a PDBM. The* semantics *of a constrained PDBM is defined by $[\![C,D]\!] = \{(v,w) \mid v \in [\![C]\!] \wedge w \in [\![D]\!]_v\}$.*

PDBMs with the tightest possible bounds are called *canonical.* To formalize this notion, we view Boolean connectives as operations on relation symbols $\leq$ and $<$ by identifying $\leq$ with 1 and $<$ with 0. Thus we have, for instance, $(\leq \wedge \leq) = \leq$, $(\leq \wedge <) = <$ and $(\leq \implies <) = <$. Our definition of a canonical form of a constrained PDBM is essentially equivalent to the one for standard DBMs.

Definition 6 (Canonical Form). *A constrained PDBM (C,D) is* in canonical form *iff for all i,j,k, $C \models e_{ij}\ (\prec_{ij} \implies \prec_{ik} \wedge \prec_{kj})\ e_{ik} + e_{kj}$.*

The next important lemma, which basically carries over from the unparametrized case, states that canonicity of a constrained PDBM guarantees satisfiability.

Lemma 1. *Suppose (C,D) is a constrained PDBM in canonical form and $v \in [\![C]\!]$. Then D is satisfiable for v.*

Also the following lemma essentially carries over from the unparametrized case, see for instance [8]. As a direct consequence, semantic inclusion of constrained PDBMs is decidable for canonical PDBMs (using the oracle function).

Lemma 2. *Suppose $(C,D), (C',D')$ are constrained PDBMs and (C,D) is canonical. Then $[\![C,D]\!] \subseteq [\![C',D']\!] \Leftrightarrow ([\![C]\!] \subseteq [\![C']\!] \wedge \forall i,j : C \models e_{ij}(\prec_{ij} \implies \prec'_{ij})e'_{ij})$.*

3.2 Operations on PDBMs

Our algorithm requires basically four operations to be implemented on constrained PDBMs: adding guards, canonicalization, resetting clocks and computing time successors.

Adding Guards In the case of DBMs, adding a guard is a simple operation. It is implemented by taking the conjunction of a DBM and the guard (which is also viewed as a DBM). The conjunction operation just takes the pointwise minimum of the entries in both matrices. In the parametric case, adding a guard to a constrained PDBM may result in a set of constrained PDBMs. We define a relation $\leadsto$ which relates a constrained PDBM and a guard to a collection of constrained PDBMs that satisfy this guard. For this we need an operation $\mathcal{C}$ that takes a PDBM and a simple guard, and produces a constraint stating that the bound imposed by the guard is larger than the corresponding bound in the PDBM, so let $D^{ij} = (e_{ij}, \prec_{ij})$ then $\mathcal{C}(D, x_i - x_j \prec e) = e_{ij}\ (\prec_{ij} \implies \prec)\ e$. Relation $\leadsto$ is defined as the smallest relation that satisfies the following rules:

$$(R1)\ \frac{\mathcal{O}(\mathcal{C}(D,f),C) = \text{yes}}{(C,D) \overset{f}{\leadsto} (C,D)} \qquad (R2)\ \frac{\mathcal{O}(\mathcal{C}(D,f),C) = \text{no},\ f \text{ proper}}{(C,D) \overset{f}{\leadsto} (C,D[f])}$$

$$(R3)\ \frac{\mathcal{O}(\mathcal{C}(D,f),C) = \text{split}}{(C,D) \overset{f}{\leadsto} (C \cup \{\mathcal{C}(D,f)\}, D)} \qquad (R4)\ \frac{\mathcal{O}(\mathcal{C}(D,f),C) = \text{split},\ f \text{ proper}}{(C,D) \overset{f}{\leadsto} (C \cup \{\neg\mathcal{C}(D,f)\}, D[f])}$$

$$(R5)\ \frac{(C,D) \overset{g}{\leadsto} (C',D')\ ,\ (C'D') \overset{g'}{\leadsto} (C'',D'')}{(C,D) \overset{g \wedge g'}{\leadsto} (C'',D'')}$$

Lemma 3. $[\![C,D]\!] \cap [\![g]\!] \;=\; \bigcup\{[\![C',D']\!] \mid (C,D) \overset{g}{\leadsto} (C',D')\}$.

Canonicalization Each DBM can be brought into canonical form using classical algorithms for computing all-pairs shortest paths, for instance the Floyd-Warshall (FW) algorithm [6]. In the parametric case, we also apply this approach except that now we run FW *symbolically*. Below, we describe the computation steps of the symbolic FW algorithm in SOS style. Recall that the FW algorithm consists of three nested for-loops, for indices k, i and j, respectively. Correspondingly, in the SOS description of the symbolic version, we use configurations of the form (k, i, j, C, D), where (C, D) is a constrained PDBM and $k, i, j \in [0, m+1]$ record the values of indices. In the rules below, k, i, j range over $[0, m]$.

$$\frac{(C,D) \overset{x_i - x_j\ \prec_{ik} \wedge \prec_{kj}\ e_{ik}+e_{kj}}{\leadsto} (C',D')}{(k,i,j,C,D) \rightarrow_{FW} (k,i,j+1,C',D')}$$

$$(k,i,m+1,C,D) \rightarrow_{FW} (k,i+1,0,C,D)$$

$$(k,m+1,0,C,D) \rightarrow_{FW} (k+1,0,0,C,D)$$

We write $(C, D) \rightarrow_c (C', D')$ if there exists a sequence of $\rightarrow_{FW}$ steps leading from configuration $(0, 0, 0, C, D)$ to configuration $(m+1, 0, 0, C', D')$. In this case, we say that (C', D') is an *outcome* of the symbolic Floyd-Warshall algorithm on (C, D). If the semantics of (C, D) is empty, then the set of outcomes is also empty. We write $(C, D) \overset{g}{\rightsquigarrow}_c (C', D')$ iff $(C, D) \overset{g}{\rightsquigarrow} (C'', D'') \rightarrow_c (C', D')$, for some C'', D''.

The following lemma says that if we run the symbolic Floyd-Warshall algorithm, the union of the semantics of the outcomes equals the semantics of the original constrained PDBM.

Lemma 4. $[\![C, D]\!] = \bigcup\{[\![C', D']\!] \mid (C, D) \rightarrow_c (C', D')\}$.

Resetting Clocks A third operation on PDBMs that we need is resetting clocks. Since we do not allow parameters in reset sets, the reset operation on PDBMs is essentially the same as for DBMs, see [15]. The following lemma characterizes the reset operation semantically.

Lemma 5. *Let (C, D) be a constrained PDBM in canonical form, $v \in [\![C]\!]$, and w a clock valuation. Then $w \in [\![D[r]]\!]_v$ iff $\exists w' \in [\![D]\!]_v : w = w'[r]$.*

Time Successors Finally, we need to transform PDBMs for the passage of time, notation $D\!\uparrow$. As in the DBMs case [8], this is done by setting the $x_i - x_0$ bounds to $(\infty, <)$, for each $i \neq 0$, and leaving all other bounds unchanged. We have the following lemma.

Lemma 6. *Suppose (C, D) is a constrained PDBM in canonical form, $v \in [\![C]\!]$, and w a clock valuation. Then $w \in [\![D\!\uparrow]\!]_v$ iff $\exists d \geq 0\ \exists w' \in [\![D]\!]_v : w' + d = w$.*

3.3 Symbolic Semantics

With the four operations on PDBMs, we can describe the semantics of a parametric timed automaton symbolically.

Definition 7 (Symbolic semantics). *The* symbolic semantics *of PTA $\mathcal{A} = (Q, q_0, \rightarrow, I)$ is an LTS. The states are triples (q, C, D) with q a location from Q and (C, D) a constrained PDBM in canonical form. Let E be the PDBM with $\mathsf{E}^{ij} = (0, \leq)$, for all i, j. The set of initial states is $\{(q_0, C, D) \mid (\top, \mathsf{E}\!\uparrow) \overset{I(q_0)}{\rightsquigarrow}_c (C, D)\}$. The transitions are defined by the following rule:*

$$\frac{q \xrightarrow{a,g,r} q' \ ,\ (C, D) \overset{g}{\rightsquigarrow}_c (C'', D'')\ ,\ (C'', D''[r]\!\uparrow) \overset{I(q')}{\rightsquigarrow}_c (C', D')}{(q, C, D) \rightarrow (q', C', D')}.$$

Using Lemma 3 and Lemma 4, it follows by a simple inductive argument that if state (q, C, D) is reachable in the symbolic semantics and $(v, w) \in [\![C, D]\!]$ then $(v, w) \models I(q)$. It is also easy to see that the symbolic semantics of a PTA is a finitely branching transition system. It may have infinitely many reachable states though. Our search algorithm explores the symbolic semantics in an "intelligent" manner, and for instance stops whenever it reaches a state whose semantics is contained in the semantics of a state that has been encountered before. Despite this, our algorithm need not terminate.

Each run in the symbolic semantics can be simulated by a run in the concrete semantics.

Proposition 1. *For each parameter valuation v and clock valuation w, if there is a run in the symbolic semantics of $\mathcal{A}$ reaching state (q, C, D), with $(v, w) \in [\![C, D]\!]$, then this run can be simulated by a run in the concrete semantics $[\![\mathcal{A}]\!]_v$ reaching state (q, w).*

For each path in the concrete semantics, we can find a path in the symbolic semantics such that the final state of the first path is semantically contained in the final state of the second path.

Proposition 2. *For each parameter valuation v and clock valuation w, if there is a run in the concrete semantics $[\![\mathcal{A}]\!]_v$ reaching a state (q, w), then this run can be simulated by a run in the symbolic semantics reaching a state (q, C, D) such that $(v, w) \in [\![C, D]\!]$.*

3.4 Evaluating Properties

We will now explain the relation $\overset{\phi}{\models\!\!\Rightarrow}$ which relates a symbolic state and a state formula ϕ to a collection of symbolic states that satisfy ϕ. For lack of space, we do not give the full formal definition.

In order to check whether a property holds, we break it down into the small basic formulas, namely checking locations and clock guards. Checking that a clock guard holds relies on the definition given earlier, of adding that clock guard to the constrained PDBM. We rely on a special normal form of the state formula, in which all $\neg$ signs have been pushed down to the basic formulas.

The following lemma gives the soundness of relation $\overset{\phi}{\models\!\!\Rightarrow}$.

Lemma 7. *Let $[\![\phi, q]\!]$ denote the set $\{(v, w) \mid (w, q) \models \phi\}$. Then for all properties ϕ in normal form $[\![C, D]\!] \cap [\![\phi, q]\!] \;=\; \bigcup \{[\![C', D']\!] \mid (q, C, D) \overset{\phi}{\models\!\!\Rightarrow} (q, C', D')\}$.*

3.5 Algorithm

We are now in a position to present our model checking algorithm for parametric timed automata. The following algorithm describes how our tool explores the symbolic state space and searches for constraints on the parameters for which a reachability formula $\exists\Diamond\phi$ holds in a PTA $\mathcal{A}$.

```
**algorithm** Reachable($\mathcal{A}$, $\phi$)
  RESULT := $\emptyset$, PASSED := $\emptyset$, WAITING := $\{(q_0, C, D) \mid (\top, \mathsf{E}\!\uparrow) \overset{I(q_0)}{\rightsquigarrow}_c (C, D)\}$
  **while** WAITING $\neq \emptyset$ **do**
    select $(q, C, D)$ from WAITING
    RESULT := RESULT $\cup \{(q', C', D') \mid (q, C, D) \overset{\phi}{\models\!\!\Rightarrow} (q', C', D')\}$
    FALSE := $\{(q', C', D') \mid (q, C, D) \overset{\neg\phi}{\models\!\!\Rightarrow} (q', C', D')\}$
    **for each** $(q', C', D')$ in FALSE **do**
      **if** for all $(q'', C'', D'')$ in PASSED: $(q', C', D') \not\sqsubseteq (q'', C'', D'')$ **then**
        add $(q', C', D')$ to PASSED
        **for each** $(q'', C'', D'')$ such that $(q', C', D') \to (q'', C'', D'')$ **do**
          WAITING := WAITING $\cup \{(q'', C'', D'')\}$
  **return** RESULT
```

The result returned by the algorithm is a set of symbolic states, all of which satisfy ϕ, for any valuation of the parameters and clocks in the state. For invariance properties $\forall\Box\phi$, the tool performs the algorithm on $\neg\phi$, and the result is then a set of symbolic states, none of which satisfies ϕ. The answer to the model checking problem, stated in Section 2.2, is obtained by taking the union of the constraint sets from all symbolic states in the result of the algorithm; in the case of an invariance property we take the complement of this set.

Some standard operations on symbolic states that help in exploring as little as possible, have also been implemented in our tool for parametric symbolic states. We give a short explanation here, and refer to the full version of this paper for the complete story with technical details. Before starting the state space exploration, our implementation determines the *maximal constant* for each clock. This is the maximal value to which the clock is compared in any guard or invariant in the PTA. When the clock value grows beyond this value, we can ignore its real value. This enables us to identify many more symbolic states, and helps termination.

4 Reducing the Complexity

This section introduces the class of lower bound/upper bound automata and describes several (rather intuitive) observations that simplify the model checking of PTAs in this class. Our results allow us to eliminate parameters in certain cases. Since the complexity of parametric model checking grows very fast in the number of parameters, this is a relevant issue. Secondly, our observations yield a decidability result for lower bound/upper bound automata whereas the corresponding problem for general PTAs is undecidable.

Informally, a positive occurrence of a parameter in a PTA enforces (or contributes to) an upper bound on a clock difference, for instance p in $x - y < 2p$. A negative occurrence of a parameter contributes to a lower bound on a clock difference, for instance q and q' in $y - x > q + 2q'$ ($\equiv x - y < -q - 2q'$) and in $x - y < 2p - q - 2q'$.

Definition 8. *A parameter $p_i \in P$ is said to* occur *in the linear expression $e = t_0 + t_1 \cdot p_1 + \cdots t_n \cdot p_n$ if $t_i \neq 0$; p_i* occurs positively *in e if $t_i > 0$ and*

p_i occurs negatively *in* e *if* $t_i < 0$. *A* lower bound parameter *of a PTA* $\mathcal{A}$ *is a parameter that only occurs negatively in the expressions of* $\mathcal{A}$ *and an* upper bound parameter *of* $\mathcal{A}$ *a parameter that only occurs positively in* $\mathcal{A}$*. We call* $\mathcal{A}$ *a* lower bound/upper bound (L/U) automaton *if every parameter is either a lower bound parameter or an upper bound parameter of* $\mathcal{A}$*, but not both.*

From now on, we work with a fixed set $L = \{l_1, \ldots l_K\}$ of lower bound parameters and a fixed set $U = \{u_1, \ldots u_M\}$ of upper bound parameters with $L \cap U = \emptyset$ and $L \cup U = P$.

We consider, apart from parameter valuations, also *extended parameter valuations.* Intuitively, an extended parameter valuation is a parameter valuation with values in $\mathsf{R}^{\geq 0} \cup \{\infty\}$, rather than in $\mathsf{R}^{\geq 0}$. We denote an extended valuation of an L/U automaton by a pair (λ, μ), which equals the function λ on the set L and μ on U and require that λ and μ do not both assign the value ∞ to a parameter. Then we can extend the notions defined for parameter valuations (Section 2) to extended valuations in the obvious way. We write 0 and ∞ for the functions assigning respectively 0 and ∞ to each parameter.

The following proposition is based on the fact that weakening the guards in $\mathcal{A}$ (i.e. decreasing the lower bounds and increasing the upper bounds) yields an automaton whose reachable states include those of $\mathcal{A}$. Dually, strengthening the guards in $\mathcal{A}$ (i.e. increasing the lower bounds and decreasing the upper bounds) yields an automaton whose reachable states are a subset of those of $\mathcal{A}$. We claim that this proposition, formulated for L/U automata, can be generalized to lower bound and upper bound parameters present in general PTAs. It is however crucial that (by definition) state formulae do not contain parameters.

Proposition 3. *Let* $\mathcal{A}$ *be an L/U automaton and* ϕ *a state formula. Then*

1. $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \models \exists\Diamond\phi \iff \forall\lambda' \leq \lambda, \mu \leq \mu' : [\![\mathcal{A}]\!]_{(\lambda',\mu')} \models \exists\Diamond\phi$.
2. $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \models \forall\Box\phi \iff \forall\lambda \leq \lambda', \mu' \leq \mu : [\![\mathcal{A}]\!]_{(\lambda',\mu')} \models \forall\Box\phi$.

The following example illustrates how Proposition 3 can be used to eliminate parameters from L/U automata.

Example 2. The automaton in Fig. 2 is an L/U automaton. Its location S_1 is reachable irrespective of the parameter values. By setting the parameter *min* to ∞ and *max* to 0, one checks with a non-parametric model checker that $\mathcal{A}[(\infty, 0)] \models \exists\Diamond S_1$. Then Proposition 3 (together with $[\![\mathcal{A}]\!]_v = \mathcal{A}[v]$) yields that S_1 is reachable in $[\![\mathcal{A}]\!]_{(\lambda,\mu)}$ for all extended parameter valuations $0 \leq \lambda, \mu \leq \infty$.

Clearly, $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \models \exists\Diamond S_2$ iff $\lambda(min) \leq \mu(max) \wedge \lambda(min) < \infty$. We will see in this running example how we can verify this property completely by non-parametric model checking. Henceforth, we construct the automaton $\mathcal{A}'$ from $\mathcal{A}$ by substituting the parameter *max* by the parameter *min* yielding an (non L/U) automaton with one parameter, *min*. If we show that $[\![\mathcal{A}']\!]_v \models \exists\Diamond S_2$ for all valuations v, this essentially means that $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \models \exists\Diamond S_2$ for all λ, μ such that $\mu(max) = \lambda(min) < \infty$ and then Proposition 3 implies that $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \models \exists\Diamond S_2$ for all λ, μ with $\lambda(min) \leq \mu(max)$ and $\lambda(min) < \infty$.

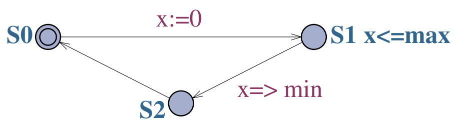

Fig. 2. Reducing parametric to non-parametric model checking

The question whether there exists a (non-extended) parameter valuation such that a given (final) location q is reachable, is known as the *emptiness problem* for PTAs. In [2], it is shown that the emptiness problem is undecidable for PTAs with three clocks or more. Proposition 3 implies $\exists\lambda,\mu : [\![\mathcal{A}]\!]_{\lambda,\mu} \models \exists\Diamond q$ iff $\mathcal{A}[0,\infty] \models \exists\Diamond q$. Here, (λ,μ) range over extended parameter valuations, but is not difficult to see that the statement also holds for (λ,μ) just valuations. Since $\mathcal{A}[(0,\infty)]$ is a non-parametric timed automaton and reachability is decidable for timed automata ([1]), the emptiness problem is decidable for L/U automata. Then it follows that the dual problem is also decidable for L/U automata. This is the *universality problem* for invariance properties, asking whether an invariance property holds for all parameter valuations.

Corollary 1. *The emptiness problem is decidable for L/U automata.*

Definition 9. *A PTA $\mathcal{A}$ is* fully parametric *if clocks are only reset to* 0 *and every linear expression in $\mathcal{A}$ of the form $t_1 \cdot p_1 + \cdots + t_n \cdot p_n$, where $t_i \in \mathsf{Z}$.*

The following proposition is basically the observation in [1], that multiplication of each constant in a timed automaton and in a system property with the same positive factor preserves satisfaction.

Proposition 4. *Let $\mathcal{A}$ be fully parametric PTA. Then*

$$[\![\mathcal{A}]\!]_v \models \psi \iff \forall t \in \mathsf{R}^{>0} : [\![\mathcal{A}]\!]_{t\cdot v} \models t \cdot \psi,$$

where $t \cdot v$ denotes the valuation $p \mapsto t \cdot v(p)$ and $t \cdot \psi$ the formula obtained from ψ by multiplying each number in ψ by t.

Then for fully parametric PTAs with one parameter and system properties ψ without constants (except for 0), we have $[\![\mathcal{A}]\!]_v \models \psi$ for all valuations v of P if and only if both $\mathcal{A}[0] \models \psi$ and $\mathcal{A}[1] \models \psi$.

Corollary 2. *For fully parametric PTAs with one parameter and properties ψ without constants (except 0), it is decidable whether $\forall v \in [\![C]\!] : [\![\mathcal{A}]\!]_v \models \psi$.*

Example 3. The PTA $\mathcal{A}'$ mentioned in Example 2 is a fully parametric automaton and the property $\exists\Diamond S_2$ is without constants. We establish that $\mathcal{A}'[0] \models \exists\Diamond S_2$ and $\mathcal{A}'[1] \models \exists\Diamond S_2$. Then Proposition 4 implies that $\mathcal{A}'[v] \models \exists\Diamond S_2$ for all v. As shown in Example 2, this implies that $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \models \exists\Diamond S_2$ for all λ, μ with $\lambda(min) = \mu(max) < \infty$.

In the running example, we would like to use the above methods to verify that $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \not\models \exists\Diamond S_2$ if $\lambda(min) > \mu(max)$. We can in this case not fill in for $min = max$, since the bound in the constraint is strict. The following definition and result allows us to move the strictness of a constraint into the PTA.

Definition 10. *Define $\mathcal{A}^{<}$ as the automaton obtained by replacing every inequality $x - y \leq e$ in $\mathcal{A}$ by a strict inequality $x - y < e$, provided that e contains at least one parameter.*

Proposition 5. *Let $\mathcal{A}$ be an L/U automaton. Then*

1. $[\![\mathcal{A}^{<}]\!]_{(\lambda,\mu)} \models \forall\Box\phi \iff \forall\lambda < \lambda', \mu' < \mu : [\![\mathcal{A}]\!]_{(\lambda',\mu')} \models \forall\Box\phi.$
2. $[\![\mathcal{A}^{<}]\!]_{(\lambda,\mu)} \models \exists\Diamond\phi \implies \forall\lambda' < \lambda, \mu < \mu' : [\![\mathcal{A}]\!]_{(\lambda',\mu')} \models \exists\Diamond\phi.$

We claim that we can extend the result above to a more general construction $\mathcal{A}^{<}_{P'}$, where we replace a guard $x-y \leq e$ by $x-y < e$ by if and only if a parameter p from P' occurs in e. Then the proposition generalizes to $\mathcal{A}^{<}_{P'}$, provided that we replace $\lambda < \lambda'$ by $\lambda <_{P'} \lambda'$ (and similar replacements for $\lambda' < \lambda$, $\mu < \mu'$, $\mu' < \mu$). Here, $v <_{P'} v'$ is defined as $v(p) < v'(p)$ if $p \in P'$ and $v(p) = v'(p)$ otherwise.

Example 4. Consider the PTA $\mathcal{A}^{<}$, which equals the PTA in Fig. 2, except that $x \leq max$ has been replace by $x < max$ and $x \geq min$ by $x > min$. Now, we construct the automaton $\mathcal{A}'$ from $\mathcal{A}^{<}$ by substituting the parameter max by min. By checking that $\mathcal{A}'[0] \models \forall\Box\neg S_2$ and $\mathcal{A}'[1] \models \forall\Box\neg S_2$, Proposition 4 yields that $\mathcal{A}'[v] \models \forall\Box\neg S_2$ for all valuations v. Then we know by Proposition 3 that $[\![\mathcal{A}']\!]_{(\lambda,\mu)} \models \forall\Box\neg S_2$ if $\infty > \lambda(min) \geq \mu(max)$. Now, Proposition 5 concludes that if $\infty > \lambda(min) > \mu(max)$ then $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \models \forall\Box\neg S_2$ i.e. $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \not\models \exists\Diamond S_2$. Combining the results from the examples in this section yields $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \models \exists\Diamond S_2$ if and only if $\lambda(min) \leq \mu(max) \wedge \lambda(min) < \infty$.

5 Experiments

In this section, we report on the results of experimenting with a prototype extension of Uppaal described in the previous sections. For lack of space, we give a short impression of the experiments, which are described in greater detail in the full version [10].

The Root Contention Protocol The root contention protocol is part of a leader election protocol in the physical layer of the IEEE 1394 standard (FireWire/i-Link), which is used to break symmetry between two nodes contending to be the root of a tree, spanned in the network topology.

We use the Uppaal models of [14,13], turn the constants used into parameters, and experiment with our prototype implementation (see Fig. 3 for results[2]).

[2] All experiments were performed on a 366 MHz Celeron, except the liveness property which was performed in a 333 MHz SPARC Ultra Enterprise.

In both models, there are five constants, all of which are parameters in our experiments. We have checked for safety and liveness on the parametric models, and have applied reductions as proposed in Section 4 where this was possible, to reduce the verification effort. In some cases, we could even derive the parametric conclusions by non-parametric model checking, which we have done with standard UPPAAL.

model from	*initial constraints?*	*reduced?*	*property*	UPPAAL	*time*	*memory*
[14]	yes	no	safety	param	2.9 h	185 Mb
[14]	yes	yes	safety	std	1 s	800 Kb
[13]	yes	no	safety	param	1.6 m	36 Mb
[13]	yes	partly	safety	param	11 s	13 Mb
[13]	yes	completely	safety	std	1 s	800 Kb
[13]	yes	no	liveness	param	2.6 h	308 Mb

Fig. 3. Experimental results for the root contention protocol

The Bounded Retransmission Protocol This protocol was designed by Philips for communication between remote controls and audio/video/TV equipment. In [7] constraints for the correctness of the protocol are derived by hand, and some instances are checked using UPPAAL. Based on the models in [7], an automatic parametric analysis is performed in [3], however, no further results are given.

model from	*initial constraints*	*property*	UPPAAL	*time*	*memory*
[7]	yes	safety1	param	1.3 m	34 Mb
[7]	no	safety2	param	11 m	180 Mb
[7]	yes	safety2	param	3.5 m	64 Mb

Fig. 4. Experimental results for the bounded retransmission protocol

For our analysis we have also used the timed automata models from [7]. In [7] three different constraints are presented based on three properties which are needed to satisfy the safety specification of the protocol. We are only able to check two of these since one of the properties contain a parameter which our prototype version of UPPAAL is not able to handle yet. The results can be found in Fig. 4[3]. Note that out of the four constants in the model which are candidates for parameters, the model checked for property 'safety1' and 'safety2' uses two and one as parameters respectively. A minor error in [7] was found while checking 'safety 1', which has been corrected by the authors of [7].

[3] All experiments run on a 333 MHz SPARC Ultra Enterprise.

References

1. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
2. R. Alur, T.A. Henzinger, and M.Y. Vardi. Parametric real-time reasoning. In *Proc. 25th Annual Symp. on Theory of Computing*, pages 592–601. ACM Press, 1993.
3. A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *Proc. 12th Int. Conference on Computer Aided Verification*, LNCS 1855, pages 419–434. Springer-Verlag, 2000.
4. G. Bandini, R. Lutje Spelberg, and H. Toetenel. Parametric verification of the IEEE 1394a root contention protocol using LPMC. http://tvs.twi.tudelft.nl/, July 2000. Submitted.
5. M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A Model-Checking Tool for Real-Time Systems. In *Proc. 10th Int. Conference on Computer Aided Verification*, LNCS 1427, pages 546–550. Springer-Verlag, June/July 1998.
6. T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* McGraw-Hill, Inc., 1991.
7. P.R. D'Argenio, J.-P. Katoen, T.C. Ruys, and J. Tretmans. The bounded retransmission protocol must be on time! In *Proc. Third Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 1217, pages 416–431. Springer-Verlag, April 1997.
8. D. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proc. Int. Workshop on Automatic Verification Methods for Finite State Systems*, LNCS 407, pages 197–212. Springer-Verlag, 1990.
9. T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A Model Checker for Hybrid Systems. In *Proc. 9th Int. Conference on Computer Aided Verification*, LNCS 1254, pages 460–463. Springer-Verlag, 1997.
10. T.S. Hune, J.M.T. Romijn, M.I.A. Stoelinga, and F.W. Vaandrager. Linear parametric model checking of timed automata. Report CSI-R0102, CSI, University of Nijmegen, January 2001.
11. K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.
12. R.F. Lutje Spelberg, W.J. Toetenel, and M. Ammerlaan. Partition refinement in real-time model checking. In *Proc. FTRTFT'98*, LNCS 1486, pages 143–157. Springer-Verlag, 1998.
13. D.P.L. Simons and M.I.A. Stoelinga. Mechanical verification of the IEEE 1394a root contention protocol using Uppaal2k. Technical Report CSI-R0009, CSI, University of Nijmegen, May 2000. Conditionally accepted for *STTT*.
14. M.I.A. Stoelinga and F.W. Vaandrager. Root contention in IEEE 1394. In *Proc. 5th Int. AMAST Workshop on Formal Methods for Real-Time and Probabilistic Systems*, LNCS 1601, pages 53–74. Springer-Verlag, 1999.
15. S. Yovine. Model checking timed automata. In *Lectures on Embedded Systems*, LNCS 1494, pages 114–152. Springer-Verlag, October 1998.