CHAPTER 6

Linear Parametric Model Checking of Timed Automata

Time present and time past
Are both perhaps present in time future
time future contained in time past.
If all time is eternally present
All time is unredeemable.
What might have been is an abstraction
Remaining a perpetual possibility
Only in a world of speculation.
What might have been and what has been
Point to one end, which is always present.
Footfalls echo in the memory
Down the passage which we did not take
Towards the door we never opened
Into the rose-garden. My words echo
Thus, in your mind.

T. S. Eliot, *Four Quartets*

Abstract We present an extension of the model checker Uppaal, capable of synthesizing linear parameter constraints for the correctness of parametric timed automata. A symbolic representation of the (parametric) state space in terms of parametric difference bound matrices is shown to be correct. A second contribution of this chapter 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 known to be undecidable. Also, we present a number of lemmas that reduce the verification effort for L/U automata in certain 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).

6.1 Introduction

Model checking is emerging as a practical tool for automated debugging of complex reactive systems such as embedded controllers and network protocols. In model checking, a high-level description of a system is compared against a logical correctness requirement to discover inconsistencies. Traditional techniques for model checking do not admit an explicit modeling of time, and are thus unsuitable for analysis of real-time systems whose correctness depends on relative magnitudes of different delays. Consequently, Alur and Dill [AD94] proposed timed automata as a formal notion to model the behavior of real-time systems. Timed

automata are state-transition diagrams annotated with timing constraints using finitely many real-valued clock variables. During the last decade, there has been enormous progress in the area of timed model checking. We refer to [Alu98, CGP99, LPY97, Yov98] for overviews of the underlying theory and references to applications. Timed automata tools such as Uppaal [LPY97], Kronos [BDM+98], and PMC [LTA98] 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. Typical examples of such parameters are upper and lower bounds on computation times, message delays and timeouts. 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 chapter, we study a parametric extension of timed automata, as well as a corresponding extension of the (forward) state space exploration algorithm for timed automata. We show the theoretical correctness of our approach, and its feasibility by application to two non-trivial case studies. For this purpose, we have implemented a prototype extension of Uppaal, an efficient real-time model checking tool [LPY97]. The algorithm we propose and have implemented fundamentally relies on parametric difference bound matrices (PDBMs) and operations on these. PDBMs constitute a data type that extends the difference bound matrices (DBMs, [Dil98]) in a natural way. The latter are used for recording clock differences when model checking (non-parametric) timed automata. PDBMs are basically DBMs, where the matrix entries are parameter expressions rather than constants. Our algorithm is a semi-decision algorithm which will not terminate in all cases. In [AHV93], the problem of synthesizing values for parameters 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 chapter is the identification of a subclass of parametric 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 emptiness problem, in [AHV93] shown to be undecidable for parametric timed automata, 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 several of our experiments.

Related work There are currently several other tools available that can do parametric model checking, namely LPMC, HyTech and TReX.

LPMC [LTA98] is a parametric extension of the timed model checker PMC [BST00]. The model checking algorithm implemented in LPMC differs from ours: it represents the state space of a system as an unstructured set of constraints, whereas we use PDBMs. Moreover, LPMC implements a partition refinement technique, whereas we use forward reachability. 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).

The model checker HyTech [HHW97] is a tool for linear hybrid automata. These are more general than parametric timed automata, since they allow the modeling of continuous

behavior via linear differential equations. The HyTech implementation uses polyhedra as its basic data type. It can explore the state space by using either forward reachability, as we do, or partition refinement, as in LPMC. The tool has been applied successfully to 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 chapter, see [CS01].

The tool TReX [AAB00, ABS01] is currently the only one that can deal with non-linear parameter constraints. Moreover, TReX has a clever method for guessing the effect of control loops in a model, based on widening principles, which increases chances of termination. Independently, [AAB00] developed the same data structure as we did (PDBMs) and implemented some similar operations on these. However, the details of this were not worked out in the research literature. Hence, we believe that our contribution over [AAB00] consists of the following. Our work presents an extensive elaboration of the theory behind our implementation. In particular, we present a correctness proof of the model checking algorithm we implemented. That is, we prove that the symbolic semantics in terms of PDBMs is sound and complete for the concrete semantics in terms of single states and transitions. These proofs rely on a number of non-trivial generalizations of results for DBMs.

Each of the tools above has been applied to the IEEE 1394 Root Contention Protocol [CS01, BST00]. In Chapter 7, section 7.3, we present a comparison of the results of these case studies. An important outcome was that each of the verification approaches has it own merits, where our verification was the fastest. Recent experiments have shown that this also holds if we generate the constraints, rather than providing them to the tool and checking their correctness.

Organization of the chapter

The remainder of this chapter is organized as follows. Section 6.2 introduces the notion of parametric timed automata. Section 6.3 gives the symbolic semantics in terms of PDBMs and is the basis for the model checking algorithm presented in Section 6.3.5. In Section 6.4, we introduce the class of L/U automata. Section 6.5 reports on several experiments with our tool. Finally, Section 6.6 presents some conclusions.

Acknowledgements We thank the reviewers for their reports, in particular Reviewer 3 who gave many comments that helped us to improve our paper and pointed out the necessity of imposing nonnegative lower bounds on clocks in constrained PDBMs.

6.2 Parametric Timed Automata

Parametric timed automata were first defined in [AHV93]. They generalize the timed automata of [AD94]. The definition of parametric timed automata that we present in this section is very similar to the definition in [AHV93], except that progress is ensured via location invariants rather than via accepting states. This difference is not essential.

6.2.1 Parameters and Constraints

Throughout this chapter, we assume a fixed set of *parameters* $P = \{p_1, \ldots, p_n\}$.

Definition 6.2.1 (Constraints) A *linear expression* e is either an expression of the form $t_1 p_1 + \cdots + t_n p_n + t_0$, where $t_0, \ldots, t_n \in \mathbb{Z}$, 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, denoted $\neg c$, is obtained by replacing relation symbols $<, \leq, >, \geq$ by $\geq, >, \leq, <$, respectively. A *(parameter) valuation* is a function $v : P \to \mathbb{R}^{\geq 0}$ assigning a nonnegative real value to each parameter. There is a one-to-one correspondence between valuations and points in $(\mathbb{R}^{\geq 0})^n$. In fact we often identify a valuation v with the point $(v(p_1), \ldots, v(p_n)) \in (\mathbb{R}^{\geq 0})^n$.

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, denoted $v \models c$, if $c[v]$ evaluates to true. The *semantics* of a constraint c, denoted $[\![c]\!]$, is the set of valuations 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 say that C is *satisfiable* if $[\![C]\!]$ is nonempty.

Constraint c *covers* constraint set C, denoted $C \models c$, 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? There are three possible answers to this, *yes*, *no*, and *split*. A split occurs when c holds for some valuations in the semantics of C and $\neg c$ holds for some other valuations. Here will not discuss in detail methods for answering such questions: in the remainder of this paper we just assume the presence of the following "oracle" function.

Definition 6.2.2 (Oracle)

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

The oracle function can be computed in polynomial time using linear programming (LP) solvers. Suppose we want to compute $\mathcal{O}(c, C)$, where c takes the form $e \leq e'$. Then we first maximize the linear function $e' - e$ subject to the set C of linear inequalities. This is a linear programming problem. If the outcome is negative then $\mathcal{O}(c, C) = \text{no}$. Otherwise we maximize $e - e'$ subject to C. If the outcome is less than or equal to 0 then $\mathcal{O}(c, C) = \text{yes}$. Otherwise $\mathcal{O}(c, C) = \text{split}$. In our implementation we use an LP solver that was kindly provided to us by the authors of [BST00], who built it for their LPMC model checking tool. This LP solver is geared to perform well on small, simple sets of constraints rather than large, complicated ones.

Observe that using the oracle, we can easily decide semantic inclusion between constraint sets: $[\![C]\!] \subseteq [\![C']\!]$ iff $\forall c' \in C' : \mathcal{O}(c', C) = \text{yes}$.

6.2.2 Parametric Timed Automata

Throughout this chapter, 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 (and hence does not increase with time).

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 $w : X \to \mathbb{R}^{\geq 0}$ assigning a nonnegative real value to each clock, such that $w(x_0) = 0$. We will identify a clock valuation w with the point $(w(x_0), \ldots, w(x_m)) \in (\mathbb{R}^{\geq 0})^{m+1}$. 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, denoted $(v, w) \models g$, if $g[v, w]$ evaluates to true. The *semantics* of a guard g, denoted $[\![g]\!]$, is the set of pairs (v, w) such that $(v, w) \models g$. Given a parameter valuation v, we write $[\![g]\!]_v$ for the set of clock valuations $\{w \mid (v, w) \models g\}$.

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 [AD94, Yov98] called parametric timed automata. Similar models have been presented in [AHV93, AAB00, BST00].

Definition 6.2.3 (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 q, an action a, a guard g, a reset set r, and a target location q' 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 $j = 0$, that is, the simple guard represents an upper bound on a clock.[1]

Example 6.2.4 A parametric timed automaton with clocks x, y and parameters p, q can be seen in Fig. 6.1. The initial location is $S0$ and has invariant $x \leq p$. There is a transition from

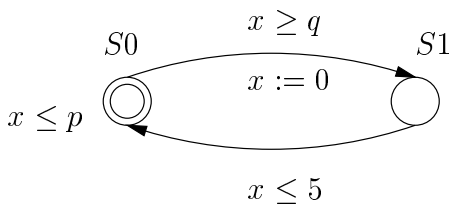

Figure 6.1: A parametric timed automaton

the initial location to $S1$, which has guard $y \geq q$ and reset set $\{x := 0\}$. There are no actions on the transitions. The transition from $S0$ to $S1$ can only become enabled if $p \leq q$, otherwise the system will end up in a deadlock.

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

[1] There is no fundamental reason to impose this restriction on invariants; our whole theory can be developed without it. However, technically the restriction makes our lives a bit easier, see for instance Proposition 6.3.17. In practice the condition is (as far as we are aware) always met.

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 6.2.5 (LTS) A *labeled transition system (LTS)* over a set of symbols Σ is a triple $Ł = (S, S_0, \rightarrow)$, with S a set of *states*, $S_0 \subseteq S$ a set of *initial states*, and $\rightarrow \subseteq S \times \Sigma \times S$ a *transition relation*. We write $s \xrightarrow{a} s'$ for $(s, a, s') \in \rightarrow$. A *run* of Ł is a finite alternating sequence $s_0 a_1 s_1 a_2 \cdots s_n$ of states $s_i \in S$ and symbols $a_i \in \Sigma$ such that $s_0 \in S_0$ and, for all $i < n$, $s_i \xrightarrow{a_{i+1}} s_{i+1}$. A state is *reachable* if it is the last state of some run.

Definition 6.2.6 (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*, denoted $[\![\mathcal{A}]\!]_v$, is the labeled transition system (LTS) $(S, S_0, \rightarrow)$ over $A \cup \mathbb{R}^{\geq 0}$ where

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

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 $\exists g, r : q \xrightarrow{a,g,r} q'$ and $(v, w) \models g$ and $w' = w[r]$.

Note that the LTS $[\![\mathcal{A}]\!]_v$ has at most one initial state. It has no initial state if the invariant which the initial location must satisfy is unsatisfiable.

6.2.3 The Parametric Model Checking Problem

In its current version, Uppaal is able to check for reachability properties, in particular whether certain combinations of locations and constraints on clock variables are reachable from the initial configuration. Our parametric 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 6.2.7 (Properties) Let $\mathcal{A} = (Q, q_0, \rightarrow, I)$ be a PTA. The sets of *system properties* and *state formulas* for $\mathcal{A}$ are defined by, respectively,

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

where $x, y \in X$, $b \in \mathbb{N}$ 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 $s \models_v \phi$ if ϕ holds in state s of $[\![\mathcal{A}]\!]_v$, 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 chapter 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 $[\![\mathcal{A}]\!]_v \models \psi$.

Remark 6.2.8 Timed automata [AD94, Yov98] 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.[2] It is easy to see that in general $[\![\mathcal{A}]\!]_v = [\![\mathcal{A}[v]]\!]$. Since the reachability problem for timed automata is decidable [AD94], this implies that, for any $\mathcal{A}$, integer valued v and ψ, $[\![\mathcal{A}]\!]_v \models \psi$ is decidable.

6.2.4 Example: Fischer's Mutual Exclusion Algorithm

Figure 6.3 shows a PTA model of Fischer's mutual exclusion algorithm [Lam87]. The purpose of this algorithm is to guarantee mutually exclusive access to a critical section among competing processes P_1, $P_2, \ldots$ P_n. In the algorithm, a shared variable lock is used for communication between the processes, with each process P_i running the code of Figure 6.2.

```
FISCHER (P_i)
lock := 0
repeat
    while lock ≠ 0 do skip od
    lock := i
    delay
until lock = i
critical section
lock := 0
```

Figure 6.2: Fischer's mutual exclusion algorithm

The correctness of this algorithm crucially depends on the timing of the operations. The key idea for the correctness is that any process P_i that sets lock $:= i$ is made to wait long enough before checking lock $= i$ to ensure that any other process P_j that tested lock $= 0$, before P_i set lock to its index, has already set lock to its index j, when P_i finally checks lock $= i$.

Assume that read/write access to the global variable (in the operations lock $= i$ and lock $:= 0$) takes between min_rw and max_rw time units and assume that the delay operation (including the time needed for the assignment lock $:= i$) takes between min_delay and max_delay time units. If we assume the basic parameter constraint $0 \leq min_rw < max_rw \wedge 0 \leq min_delay < max_delay$, then mutual exclusion is guaranteed if and only if $max_rw \leq min_delay$.

Now consider the PTA in Fig. 6.3. (Several different models of this algorithm exist [AL92, AHV93, Lyn96, KLL+97]; our model is closest to the one in [Lyn96].) It consists of four locations $start$ (which is initial), set, try_enter and cs; four parameters, min_rw, max_rw, min_delay and max_delay; one clock x and a shared variable $lock$. By convention, x and $lock$ are initially 0. Note that the process can remain in the locations $start$ and set for at least min_rw and strictly less than max_rw time units. Similarly, the process can remain in try_enter for a time in the interval $[min_delay, max_delay)$.

The shared variable, which is not a part of the definition of PTAs, is syntactic sugar which allows for an efficient encoding of the algorithm as a PTA. Also the notion of parallel

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

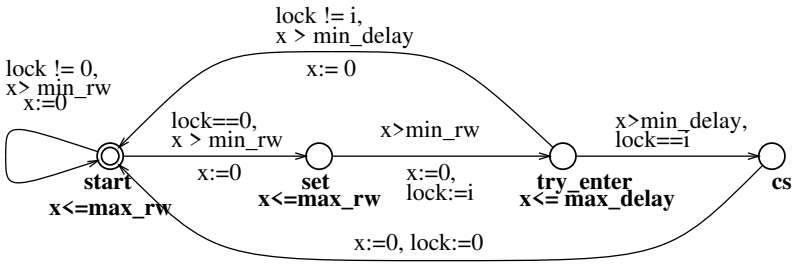

Figure 6.3: A PTA model of Fischer's mutual exclusion algorithm

composition for PTAs is standard, see for instance [LPY97] for their definitions.

6.3 Symbolic State Space Exploration

Our aim is to use basically the same algorithm for parametric timed 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 nonparametric case, sets of states can be efficiently represented using matrices [Dil98]. Similarly, in this chapter we represent sets of states symbolically as *(constrained) parametric difference bound matrices*.

A very similar approach was followed in [AAB00], although not worked out in detail. New in our presentation is the systematic use of structural operational semantics to deal with the nondeterministic computation that takes place in the parametric case.

6.3.1 Parametric difference bound matrices

In the nonparametric 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 $x_i - x_j < c$. In the parametric case, 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 6.3.1 (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 given by $[\![D]\!]_v = [\![\bigwedge_{i,j} D_{ij}]\!]_v$. PDBM D is *satisfiable* for v if $[\![D]\!]_v$ is nonempty. If f is a guard of the form $x_i - x_j \prec e$ with $i \neq j$ (i.e., a proper guard), then $D[f]$ denotes the PDBM obtained from D by replacing D_{ij} by f. If i,j are indices then D^{ij} denotes the pair $(e_{ij}, \prec_{ij})$; we call D^{ij} a *bound* of D. Clearly, a PDBM is fully determined by its bounds.

Definition 6.3.2 (Constrained PDBM) A *constrained PDBM* is a pair (C,D) where C is a constraint set and D is a PDBM. We require that $C \models p \geq 0$, for each p, and $C \models e_{0i} \leq 0$, for each i. The *semantics* of (C,D) is given by $[\![C,D]\!] = \{(v,w) \mid v \in [\![C]\!] \wedge w \in [\![D]\!]_v\}$. We call (C,D) *satisfiable* if $[\![C,D]\!]$ is nonempty.

Condition $C \models p \geq 0$ expresses that parameter p may only take nonnegative values. The condition $C \models e_{0i} \leq 0$ ensures a nonnegative lower bound on the value of clock x_i. Such a condition is required since clocks in a PTA only take nonnegative values. A similar condition occurs in [Yov98]. In the setting of [Dil98] the condition of nonnegative lower bounds is not needed since in this paper clocks (called timers) may take values in $\mathbb{R}$. In [LLPY97, Alu98, CGP99, AAB00] the condition (or something similar) is needed but not mentioned.[3]

PDBMs with the tightest possible bounds are called *canonical*. To formalize this notion, we define an addition operation on linear expressions by

$$\begin{array}{rcl}(t_1p_1 + \cdots + t_np_n + t_0) & + & (t'_1p_1 + \cdots + t'_np_n + t'_0) \\ & \triangleq & (t_1 + t'_1)p_1 + \cdots + (t_n + t'_n)p_n + (t_0 + t'_0).\end{array}$$

Also, we view Boolean connectives as operations on relation symbols $\leq$ and $<$ by identifying $\leq$ with 1 and $<$ with 0. Thus we have, for instance, $(\leq \wedge \leq) = \leq$, $(\leq \wedge <) = <$, $\neg \leq = <$, and $(\leq \Longrightarrow <) = <$.

Our definition of a canonical form of a constrained PDBM is essentially equivalent to the one for standard DBMs.

Definition 6.3.3 (Canonical Form) A constrained PDBM (C, D) is *in canonical form* iff for all i, j, k, $C \models e_{ij}\ (\prec_{ij} \Longrightarrow \prec_{ik} \wedge \prec_{kj})\ e_{ik} + e_{kj}$.

The proof of the following technical lemma is immediate from the definitions.

Lemma 6.3.4

1. *If* $v \models e \prec e'$ *and* $v \models e' \prec' e''$ *then* $v \models e\ (\prec \wedge \prec')\ e''$.
2. *If* $(v, w) \models x - y \prec e$ *and* $v \models e \prec' e'$ *then* $(v, w) \models x - y\ (\prec \wedge \prec')\ e'$.
3. *If* $v \models e\ (\prec \wedge \prec')\ e'$ *then* $v \models e \prec e'$.
4. *If* $(v, w) \models x - y\ (\prec \wedge \prec')\ e$ *then* $(v, w) \models x - y \prec e$.
5. *If* $(v, w) \models x - y \prec e$ *and* $(v, w) \models y - z \prec' e'$ *then* $(v, w) \models x - z\ (\prec \wedge \prec')\ e + e'$.
6. $v \models \neg(e \prec e')$ *iff* $v \models e'\ (\neg \prec)\ e$.

The next lemma states that canonicity of a constrained PDBM guarantees satisfiability

Lemma 6.3.5 *Suppose* (C, D) *is a constrained PDBM in canonical form and* $v \in [\![C]\!]$. *Then* D *is satisfiable for* v.

PROOF: By induction on i, with $0 \leq i \leq m$, we construct a valuation $(t_0, \ldots, t_i)$ for clock variables $(x_0, \ldots, x_i)$ such that all constraints D_{jk} for $0 \leq j, k \leq i$ are met.

To begin with, we set $t_0 = 0$. Then, trivially, $(v, x_0 \mapsto t_0) \models D_{00}$.

For the induction step, suppose that for some $i < m$ we have a valuation $(t_0, \ldots, t_i)$ for variables $(x_0, \ldots, x_i)$ such that all constraints D_{jk} for $0 \leq j, k \leq i$ are met. In order to

[3] For instance, in [CGP99] it is claimed on page 289: "If the clock zone is empty or unsatisfiable, there will be at least one negative entry in the main diagonal." This claim is incorrect. A counterexample is the canonical form of a DBM that contains as the only nontrivial guard $x_1 - x_0 \leq -1$.

extend this valuation to x_{i+1}, we have to find a value t_{i+1} such that the following simple guards hold for valuation $(v, x_0 \mapsto t_0, \ldots, x_{i+1} \mapsto t_{i+1})$:

$$D_{i+1,0} \quad \cdots \quad D_{i+1,i} \quad D_{0,i+1} \quad \cdots \quad D_{i,i+1} \quad D_{i+1,i+1} \tag{6.1}$$

Here the first $i+1$ simple guards give upper bounds for t_{i+1}, the second $i+1$ simple guards give lower bounds for t_{i+1}, and the last simple guard is trivially met by any choice for t_{i+1}. We claim that each of the upper bounds is larger than each of the lower bounds. In particular, the minimum of the upper bounds is larger than the maximum of the lower bounds. This gives us a nonempty interval of possible values for t_{i+1} to choose from. Formally, we claim that, for all $0 \leq j, k < i+1$, the following formula holds for valuation $(v, x_0 \mapsto t_0, \ldots, x_i \mapsto t_i)$:

$$x_j - e_{j,i+1} \quad \prec_{j,i+1} \wedge \prec_{i+1,k} \quad x_k + e_{i+1,k} \tag{6.2}$$

To see why (6.2) holds, observe that by induction hypothesis $(v, x_0 \mapsto t_0, \ldots, x_i \mapsto t_i) \models$

$$x_j - x_k \quad \prec_{jk} \quad e_{jk} \tag{6.3}$$

Furthermore, since (C, D) is canonical and $v \in [\![C]\!]$, $v \models$

$$e_{jk} \quad (\prec_{jk} \Longrightarrow \prec_{j,i+1} \wedge \prec_{i+1,k}) \quad e_{j,i+1} + e_{i+1,k} \tag{6.4}$$

Combination of (6.3) and (6.4), using Lemma 6.3.4(2), gives $(v, x_0 \mapsto t_0, \ldots, x_i \mapsto t_i) \models$

$$x_j - x_k \quad \prec_{j,i+1} \wedge \prec_{i+1,k} \quad e_{j,i+1} + e_{i+1,k}$$

which is equivalent to (6.2). This means that we can choose t_{i+1} in accordance with all the guards of (6.1). In particular, guard $D_{0,i+1}$ holds, which by the assumption that lower bounds on clocks are nonnegative implies that t_{i+1} is nonnegative. This completes the proof of the induction step and thereby of the lemma. □

The following lemma essentially carries over from the nonparametric case too, see for instance [Dil98]. As a direct consequence, semantic inclusion of constrained PDBMs is decidable for canonical PDBMs (using the oracle function).

Lemma 6.3.6 *Suppose $(C, D), (C', D')$ are constrained PDBMs and (C, D) is canonical. Then $[\![C, D]\!] \subseteq [\![C', D']\!] \Leftrightarrow ([\![C]\!] \subseteq [\![C']\!] \wedge \forall i, j : C \models e_{ij}(\prec_{ij} \Longrightarrow \prec'_{ij})e'_{ij})$.*

6.3.2 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 $\Leftarrow\!\!|$ 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 weaker than the corresponding bound in the PDBM. Let $D^{ij} = (e_{ij}, \prec_{ij})$. Then

$$\mathcal{C}(D, x_i - x_j \prec e) \quad = \quad e_{ij}\ (\prec_{ij} \Longrightarrow \prec)\ e.$$

Relation $\Leftarrow\!\!\!|$ is defined as the smallest relation that satisfies the following rules:

$$(R1)\ \frac{\mathcal{O}(\mathcal{C}(D,f),C) = \text{yes}}{(C,D) \overset{f}{\Leftarrow\!\!\!|} (C,D)} \qquad (R2)\ \frac{\mathcal{O}(\mathcal{C}(D,f),C) = \text{no},\ f \text{ proper}}{(C,D) \overset{f}{\Leftarrow\!\!\!|} (C,D[f])} \tag{6.5}$$

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

$$(R5)\ \frac{(C,D) \overset{g}{\Leftarrow\!\!\!|} (C',D')\ ,\ (C'D') \overset{g'}{\Leftarrow\!\!\!|} (C'',D'')}{(C,D) \overset{g \wedge g'}{\Leftarrow\!\!\!|} (C'',D'')}$$

If the oracle replies "yes" then adding a simple guard will not change the constrained PDBM. If the answer is "no" then we tighten the bound in the PDBM. With the answer "split" there are two possibilities and two PDBMs with updated constraint systems are returned. Thus the result of the operation of adding a guard is a *set* of constrained PDBMs. The side condition "f proper" in $R2$ and $R4$ rules out guards of the form $x_i - x_i \prec e$ and thereby ensures that the diagonal bounds in the PDBM always remain equal to $(0, \leq)$. It is routine to check, using Lemma 6.3.4, that relation $\Leftarrow\!\!\!|$ is well-defined in the sense that $(C,D) \overset{g}{\Leftarrow\!\!\!|} (C',D')$ implies that (C',D') is a constrained PDBM. In particular the condition that clocks have nonnegative lower bounds is met. Note that if we update a bound in D the semantics of the PDBM may become empty: a typical situation occurs when D contains a constraint $x \geq 5$ and we add a guard $x \leq 3$. Note however that $(C,D) \overset{g}{\Leftarrow\!\!\!|} (C',D')$ and $[\![C]\!] \neq \emptyset$ implies $[\![C']\!] \neq \emptyset$. The following lemma characterizes $\Leftarrow\!\!\!|$ semantically.

Lemma 6.3.7 $[\![C,D]\!] \cap [\![g]\!] \ = \ \bigcup\{[\![C',D']\!] \mid (C,D) \overset{g}{\Leftarrow\!\!\!|} (C',D')\}$.

PROOF: "$\subseteq$". Assume $(v,w) \in [\![C,D]\!] \wedge (v,w) \models g$. By structural induction on g we prove that there exists a constrained PDBM (C',D') such that $(C,D) \overset{g}{\Leftarrow\!\!\!|} (C',D')$ and $(v,w) \in [\![C',D']\!]$.

For the induction basis, suppose g is of the form $x_i - x_j \prec e$. We consider four cases:

- $\mathcal{O}(\mathcal{C}(D,g),C) = \text{yes}$. Let $C' = C$ and $D' = D$. Then trivially $(v,w) \in [\![C',D']\!]$ and, by rule $R1$, $(C,D) \overset{g}{\Leftarrow\!\!\!|} (C',D')$.

- $\mathcal{O}(\mathcal{C}(D,g),C) = \text{no}$. By contradiction we prove that g is proper. Suppose g is not proper. Then, since $i = j$ and $v \models \neg e_{ij}(\prec_{ij} \Longrightarrow \prec)e$, $v \models \neg(0 \prec e)$. By Lemma 6.3.4(6), $v \models e\neg \prec 0$. But $(v,w) \models g$ implies $v \models 0 \prec e$. Hence, by Lemma 6.3.4(1), $v \models 0 < 0$, a contradiction. Let $C' = C$ and $D' = D[g]$. Then, by rule $R2$, $(C,D) \overset{g}{\Leftarrow\!\!\!|} (C',D')$. Since $(v,w) \in [\![C,D]\!]$ and $(v,w) \models g$, it follows that $(v,w) \in [\![C',D']\!]$.

- $\mathcal{O}(\mathcal{C}(D,g),C) = \text{split}$ and $v \models \mathcal{C}(D,g)$. Let $C' = C \cup \{\mathcal{C}(D,g)\}$ and $D' = D$. Then, by rule $R3$, $(C,D) \overset{g}{\Leftarrow\!\!\shortmid} (C',D')$. Moreover, by the assumptions, $(v,w) \in [\![C',D']\!]$.

- $\mathcal{O}(\mathcal{C}(D,g),C) = \text{split}$ and $v \models \neg\mathcal{C}(D,g)$. By contradiction we prove that g is proper. Suppose g is not proper. Then, since $v \models \neg\mathcal{C}(D,g)$, $v \models \neg(0 \prec e)$. By Lemma 6.3.4(6), $v \models e\neg \prec 0$. But $(v,w) \models g$ implies $v \models 0 \prec e$. Hence, by Lemma 6.3.4(1), $v \models 0 < 0$, a contradiction. Let $C' = C \cup \{\neg\mathcal{C}(D,g)\}$ and $D' = D[g]$. Then, by rule $R4$, $(C,D) \overset{g}{\Leftarrow\!\!\shortmid} (C',D')$. By the assumptions $(v,w) \in [\![C',D']\!]$.

For the induction step, suppose that g is of the form $g' \wedge g''$. Then $(v,w) \models g'$. By induction hypothesis, there exist C'', D'' such that $(C,D) \overset{g'}{\Leftarrow\!\!\shortmid} (C'',D'')$ and $(v,w) \in [\![C'',D'']\!]$. Since $(v,w) \models g''$, we can use the induction hypothesis once more to infer that there exist C', D' such that $(C'',D'') \overset{g''}{\Leftarrow\!\!\shortmid} (C',D')$ and $(v,w) \in [\![C',D']\!]$. Moreover, by rule $R5$, $(C,D) \overset{g}{\Leftarrow\!\!\shortmid} (C',D')$.

"$\supseteq$" Assume $(C,D) \overset{g}{\Leftarrow\!\!\shortmid} (C',D')$ and $(v,w) \in [\![C',D']\!]$. By induction on size of the derivation of $(C,D) \overset{g}{\Leftarrow\!\!\shortmid} (C',D')$, we establish $(v,w) \in [\![C,D]\!]$ and $(v,w) \models g$. There are five cases, depending on the last rule r used in the derivation of $(C,D) \overset{g}{\Leftarrow\!\!\shortmid} (C',D')$.

1. $r = R1$. Then $C = C'$, $D = D'$ and $C \models \mathcal{C}(D,g)$. Let g be of the form $x_i - x_j \prec e$. Hence $(v,w) \in [\![C,D]\!]$ and $v \models \mathcal{C}(D,g)$. By the first statement $(v,w) \models x_i - x_j \prec^D_{ij} e^D_{ij}$, and by the second statement $v \models e^D_{ij}\ (\prec^D_{ij} \Longrightarrow \prec)\ e$. Combination of these two observations, using parts (2) and (4) of Lemma 6.3.4 yields $(v,w) \models g$.

2. $r = R2$. Then $C = C'$, $D' = D[g]$ and $C \models \neg\mathcal{C}(D,g)$. Hence $(v,w) \models g$ and $v \models \neg\mathcal{C}(D,g)$. Let g be of the form $x_i - x_j \prec e$. By Lemma 6.3.4(6), $v \models e\ \neg(\prec^D_{ij} \Longrightarrow \prec)\ e^D_{ij}$. Using parts (2) and (4) of Lemma 6.3.4, combination of these two observations yields $(v,w) \models x_i - x_j \prec^D_{ij} e^D_{ij}$. Since trivially (v,w) is a model for all the other guards in D, $(v,w) \in [\![C,D]\!]$.

3. $r = R3$. Then $C' = C \cup \{\mathcal{C}(D,g)\}$ and $D' = D$. Let g be of the form $x_i - x_j \prec e$. We have $(v,w) \in [\![C,D]\!]$. This implies $(v,w) \models x_i - x_j \prec^D_{ij} e^D_{ij}$. We also have $v \models e^D_{ij}\ (\prec^D_{ij} \Longrightarrow \prec)\ e$. Combination of these two observations, using parts (2) and (4) of Lemma 6.3.4 yields $(v,w) \models g$.

4. $r = R4$. Then $C' = C \cup \{\neg\mathcal{C}(D,g)\}$ and $D' = D[g]$. We have $v \models \neg\mathcal{C}(D,g)$ and $(v,w) \models g$. Let g be of the form $x_i - x_j \prec e$. By Lemma 6.3.4(6), $v \models e\ \neg(\prec^D_{ij} \Longrightarrow \prec)\ e^D_{ij}$. Using parts (2) and (4) of Lemma 6.3.4 yields $(v,w) \models x_i - x_j \prec^D_{ij} e^D_{ij}$. Since trivially (v,w) is a model for all other guards in D, $(v,w) \in [\![C,D]\!]$.

5. $r = R5$. Then g is of the form $g' \wedge g''$ and there are C'', D'' such that $(C,D) \overset{g'}{\Leftarrow\!\!\shortmid} (C'',D'')$ and $(C'',D'') \overset{g''}{\Leftarrow\!\!\shortmid} (C',D')$. By induction hypothesis, $(v,w) \in [\![C'',D'']\!]$ and $(v,w) \models g''$. Again by induction hypothesis, $(v,w) \in [\![C,D]\!]$ and $(v,w) \models g'$. It follows that $(v,w) \models g$.

□

FLOYD-WARSHALL (C_0, D_0)
$(C, D) := (C_0, D_0)$
for $k = 0$ **to** m
  do for $i = 0$ **to** m
    do for $j = 0$ **to** m
      $(C, D) :=$ **choose** (C', D') **such that**
        $(C, D) \overset{x_i - x_j \prec_{ik} \wedge \prec_{kj} e_{ik} + e_{kj}}{\Leftarrow\!\!\dashv} (C', D')$
return (C, D)

Figure 6.4: The Floyd-Warshall algorithm

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 [CLR90]. In the parametric case, we also apply this approach except that now we run FW *symbolically*, see Figure 6.4. The algorithm repeatedly compares the difference between two clocks to the difference obtained by taking an intermediate clock into account (cf. the inequality in Definition 6.3.3). The symbolic FW algorithm contains a nondeterministic assignment, in which (C, D) nondeterministically gets a value from a set. This set may be empty, in which case the algorithm terminates unsuccessfully. We are interested in the (possibly empty, finite) set of all possible constrained PDBMs that may result when running the algorithm.

For the purpose of proving things we find it convenient to describe the computation steps of the symbolic FW algorithm in SOS style. In the SOS description, 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 $[0, m]$.

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

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

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

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). It is easy to see that the set of all outcomes is finite and can be effectively computed. If the semantics of (C, D) is empty, then the set of outcomes is also empty. We write $(C, D) \overset{g}{\Leftarrow\!\!\dashv}_c (C', D')$ iff $(C, D) \overset{g}{\Leftarrow\!\!\dashv} (C'', D'') \rightarrow_c (C', D')$, 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 6.3.8 $[\![C, D]\!] = \bigcup \{ [\![C', D']\!] \mid (C, D) \rightarrow_c (C', D') \}$.

PROOF: By an inductive argument, using Lemma 6.3.7 and the fact that, for any valuation (v, w) in the semantics of (C, D),

$$\begin{array}{lll} (v,w) & \models & x_i - x_k \prec_{ik} e_{ik} \text{ and} \\ (v,w) & \models & x_k - x_j \prec_{kj} e_{kj}, \text{ and therefore by Lemma 6.3.4(5)} \\ (v,w) & \models & x_i - x_j \prec_{ik} \wedge \prec_{kj} e_{ik} + e_{kj}. \end{array}$$

□

Lemma 6.3.9 *Each outcome of the symbolic Floyd-Warshall algorithm is a constrained PDBM in canonical form.*

PROOF: As in [CLR90]. □

Remark 6.3.10 Non-parametric DBMs can be canonicalized in $\mathcal{O}(n^3)$, where n is the number of clocks. In the parametric case, however, each operation of comparing the bound $D(x, x')$ to the weight of another path from x to x' may give rise to two new PDBMs if this comparison leads to a split. Then the two PDBMs must both be canonicalized to obtain all possible PDBMs with tightest bounds. Still, that part of these two PDBMs which was already canonical, does not need to be investigated again. So in the worst case, the cost of the algorithm becomes $\mathcal{O}(2^{n^3})$. In practice, it turns out that this is hardly ever the case.

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 [Yov98]. If D is a PDBM and r is a singleton reset set $\{x_i := b\}$, then $D[r]$ is the PDBM obtained by (1) replacing each bound D^{ij}, for $j \neq i$, by $(e_{0j} + b, \prec_{0j})$; (2) replacing each bound D^{ji}, for $j \neq i$, by $(e_{j0} - b, \prec_{j0})$. We generalize this definition to arbitrary reset sets by

$$D[x_{i_1} := b_1, \ldots, x_{i_h} := b_h] \quad = \quad D[x_{i_1} := b_1] \ldots [x_{i_h} := b_h].$$

It easily follows from the definitions that resets preserve canonicity. Note also that the reset operation is well-defined on constrained PDBMs: if (C, D) is a constrained PDBM then $(C, D[r])$ is a constrained PDBM as well: since clocks can only be reset to natural numbers, lower bounds on clocks remain nonnegative.

Lemma 6.3.11 *If (C, D) is canonical then $(C, D[r])$ is canonical as well.*

The following lemma characterizes the reset operation semantically.

Lemma 6.3.12 *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 $\exists w' \in [\![D]\!]_v : w = w'[r]$.*

PROOF: We only prove the lemma for singleton resets. Using Lemma 6.3.11, the generalization to arbitrary resets is straightforward. Let $r = \{x_i := b\}$ and $D' = D[r]$.
"$\Leftarrow$" Suppose $w' \in [\![D]\!]_v$ and $w = w'[r]$. In order to prove $w \in [\![D']\!]_v$, we must show that $(v, w) \models D'_{kj}$, for all k and j. There are four cases:

1. $k \neq i \neq j$. Then $D'_{kj} = D_{kj}$. Since $(v, w') \models D_{kj}$ and w and w' agree on all clocks occurring in D_{kj}, $(v, w) \models D'_{kj}$.
2. $k = i = j$. Then $D'_{kj} = D_{kj}$. Since $(v, w') \models D_{ii}, 0 \prec_{ii} e_{ii}[v]$. Hence $(v, w) \models D'_{kj}$.
3. $k \neq i = j$. Then $D'_{kj} = x_k - x_j \prec_{k0} e_{k0} - b$. Using that $(v, w') \models D_{k0}$, we derive $w(x_k) - w(x_j) = w'(x_k) - b \prec_{k0} e_{k0}[v] - b$. Hence $(v, w) \models D'_{kj}$.
4. $k = i \neq j$. Then $D'_{kj} = x_k - x_j \prec_{0j} e_{0j} + b$. Using that $(v, w') \models D_{0j}$, we derive $w(x_k) - w(x_j) = b - w'(x_j) \prec_{0j} e_{0j}[v] + b$. Hence $(v, w) \models D'_{kj}$.

"⇒" Suppose $w \in [\![D']\!]_v$. We have to prove that there exists a clock valuation $w' \in [\![D]\!]_v$ such that $w = w'[r]$. Clearly we need to choose w' in such a way that, for all $j \neq i$, $w'(x_j) = w(x_j)$. This means that, for any choice of $w'(x_i)$, for all $j \neq i \neq k$, $v, w' \models D_{jk}$. Using the same argument as in the proof of Lemma 6.3.5, we can find a value for $w'(x_i)$ such that also the remaining simple guards of D are satisfied. □

Time Successors

Finally, we need to transform PDBMs for the passage of time, notation $D\uparrow$. As in the DBMs case [Dil98], this is done by setting the upper bounds $x_i - x_0$ to $(\infty, <)$, for each $i \neq 0$, and leaving all other bounds unchanged. We have the following lemma.

Lemma 6.3.13 *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 $\exists d \geq 0\ \exists w' \in [\![D]\!]_v : w' + d = w$.*

PROOF: "⇐" Suppose $d \geq 0$, $w' \in [\![D]\!]_v$ and $w' + d = w$. We claim that $w \in [\![D\uparrow]\!]_v$. For this we must show that for each guard f of $D\uparrow$, $(v, w) \models f$. Let f be of the form $x_i - x_j \prec e$. We distinguish between three cases:

- $i \neq 0 \wedge j = 0$. In this case, by definition of $D\uparrow$, f is of the form $x_i - x_0 < \infty$, and so $(v, w) \models f$ trivially holds.
- $i = 0$. In this case f is also a constraint of D. Since $w' \in [\![D]\!]_v$ we have $(v, w') \models f$, and thus $-w'(x_j) \prec e[v]$. But since $d \geq 0$, this means that $-w(x_j) = -w'(x_i) - d \prec e[v]$ and therefore $(v, w) \models f$.
- $i \neq 0 \wedge j \neq 0$. In this case f is again a constraint of D. Since $w' \in [\![D]\!]_v$ we have $(v, w') \models f$, and therefore $w'(x_i) - w'(x_j) \prec e[v]$. But this means that $w'(x_i) - w'(x_j) = (w(x_i) - d) - (w(x_j) - d) \prec e[v]$ and therefore $(v, w) \models f$.

"⇒" Suppose $w \in [\![D\uparrow]\!]_v$. If $m = 0$ (i.e., there are no clocks) then $D\uparrow = D$ and the rhs of the implication trivially holds (take $w' = w$ and $d = 0$). So assume $m > 0$. For all indices i, j with $i \neq 0$ and $j \neq 0$, $(v, w) \models D_{ij}$. Hence $w(x_i) - w(x_j) \prec_{ij} e_{ij}[v]$. Thus, for any real number t, $w(x_i) - t - (w(x_j) - t) \prec_{ij} e_{ij}[v]$. But this means $(v, w - t) \models D_{ij}$. It remains to be shown that there exists a value d such that in valuation $(v, w - d)$ also the remaining guards D_{0i} and D_{i0} hold. Let

$$\begin{aligned} t_0 &= \max(0, w(x_1) - e_{10}[v], \ldots, w(x_n) - e_{n0}[v]) \\ t_1 &= \min(w(x_1) + e_{01}[v], \ldots, w(x_n) + e_{0n}[v]) \\ d &= (t_0 + t_1)/2 \\ w' &= w - d \end{aligned}$$

Intuitively, t_0 gives the least amount of time one has to go backwards in time from w to meet all upper bounds of D (modulo strictness), whereas t_1 gives the largest amount of time one can go backwards in time from w without violating any of the lower bounds of D (again modulo strictness). Value d sits right in the middle of these two. We claim that d and w' satisfy the required properties. For any i, since $(v, w) \models D_{0i}$, trivially

$$0 \quad \prec_{0i} \quad w(x_i) + e_{0i}[v] \tag{6.6}$$

Using that D is canonical we have, for any i, j,

$$e_{ji}[v] \ (\prec_{ji} \Longrightarrow \prec_{j0} \wedge \prec_{0i}) \ e_{j0}[v] + e_{0i}[v]$$

and, since $v, w \models D_{ji}$,

$$w(x_j) - w(x_i) \prec_{ji} e_{ji}[v].$$

Using these two observations we infer

$$w(x_j) - e_{j0}[v] \ (\prec_{ji} \Longrightarrow \prec_{j0} \wedge \prec_{0i}) \ w(x_j) - e_{ji}[v] + e_{0i}[v] \prec_{ji} w(x_i) + e_{0i}[v].$$

Hence

$$w(x_j) - e_{j0}[v] \prec_{j0} \wedge \prec_{0i} w(x_i) + e_{0i}[v] \tag{6.7}$$

By inequalities (6.6) and (6.7), each subterm of $\max$ in the definition of t_0 is dominated by each subterm of $\min$ in the definition of t_1. This implies $0 \leq t_0 \leq t_1$.

Now either $t_0 < t_1$ or $t_0 = t_1$. In the first case it easy to prove that in valuation (v, w) the guards D_{0i} and D_{i0} hold, for any i:

$$w'(x_i) = w(x_i) - d < w(x_i) - t_0 \leq w(x_i) - (w(x_i) - e_{i0}[v]) = e_{i0}[v]$$

and thus $w'(x_i) < e_{i0}[v]$ and $v, w' \models D_{i0}$. Also

$$-w'(x_i) = -w(x_i) + d < -w(x_i) + t_1 \leq -w(x_i) + (w(x_i) + e_{0i}[v]) = e_{0i}[v]$$

and so $-w'(x_i) < e_{0i}[v]$ and $v, w' \models D_{0i}$.

In the second case, fix an i. If $w(x_i) - e_{i0}[v] < t_0$ then

$$w'(x_i) = w(x_i) - d = w(x_i) - t_0 < w(x_i) - (w(x_i) - e_{i0}[v]) = e_{i0}[v]$$

and thus $w'(x_i) < e_{i0}[v]$ and $v, w' \models D_{i0}$. Otherwise, if $w(x_i) - e_{i0}[v] = t_0$ observe that by $t_0 = t_1$, inequality (6.7) and the fact that, $t_1 = w(x_j) + e_{0j}[v]$, for some j, $\prec_{i0} = \leq$. Since

$$w'(x_i) = w(x_i) - d \leq w(x_i) - t_0 \leq w(x_i) - (w(x_i) - e_{i0}[v]) \leq e_{i0}[v]$$

and thus $w'(x_i) \leq e_{i0}[v]$ this implies $v, w' \models D_{i0}$.

The proof that $v, w' \models D_{0i}$, for all i, in the case where $t_0 = t_1$ proceeds similarly. □

6.3.3 Symbolic semantics

Having defined the four operations on PDBMs, we are now in a position to describe the semantics of a parametric timed automaton symbolically.

Definition 6.3.14 (Symbolic semantics) Let $\mathcal{A} = (Q, q_0, \to, I)$ be a PTA. The *symbolic semantics* of $\mathcal{A}$ 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 such that $[\![C, D]\!] \subseteq [\![I(q)]\!]$; the set of initial states is

$$\{(q_0, C, D) \mid (C_0, \mathsf{E}\uparrow) \overset{I(q_0)}{\Leftarrow}_c (C, D)\},$$

where $C_0 = \{p \geq 0 \mid p \in P\}$, E is the PDBM with $\mathsf{E}^{ij} = (0, \leq)$, for all i, j; the transitions are defined by the following rule:

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

Observe that if (q, C, D) is a state in the symbolic semantics and $(v, w) \in [\![C, D]\!]$, then (q, w) is a state of the concrete semantics $[\![\mathcal{A}]\!]_v$. It is also easy to see that the symbolic semantics of a PTA is a finitely branching LTS. It may have infinitely many reachable states though.

In order to establish that each run in the symbolic semantics can be simulated by a run in the concrete semantics, we require two lemmas.

Lemma 6.3.15 *Suppose that (q, C, D) is an initial state of the symbolic semantics of $\mathcal{A}$ with $(v, w) \in [\![C, D]\!]$. Then the concrete semantics $[\![\mathcal{A}]\!]_v$ has an initial state (q_0, w_0) from which state (q, w) can be reached.*

PROOF: Using the fact that $(v, w) \in [\![C, D]\!]$, the definition of initial states, Lemma 6.3.8 and Lemma 6.3.7, we know that $q = q_0$, $(v, w) \models I(q_0)$ and $(v, w) \in [\![C_0, \mathsf{E}\uparrow]\!]$. By Lemma 6.3.13, we get that there exists a $d \geq 0$ and $w_0 \in [\![\mathsf{E}]\!]_v$ such that $w_0 + d = w$. Since $(v, w) \models I(q_0)$ and invariants in a PTA only give upper bounds on clocks, also $(v, w_0) \models I(q_0)$. It follows that (q_0, w_0) is a state of the concrete semantics $[\![\mathcal{A}]\!]_v$ and $(q_0, w_0) \xrightarrow{d} (q, w)$. Since $w_0 \in [\![\mathsf{E}]\!]_v$, w_0 is of the form $\lambda x.0$. Hence (q_0, w_0) is an initial state of the concrete semantics. □

Lemma 6.3.16 *Suppose that $(q', C', D') \to (q, C, D)$ is a transition in the symbolic semantics of $\mathcal{A}$ and $(v, w) \in [\![C, D]\!]$. Then there exists a pair $(v, w') \in [\![C, D]\!]$ such that in the concrete semantics $[\![\mathcal{A}]\!]_v$ there is a path from (q', w') to (q, w).*

PROOF: By the definition of transitions in the symbolic semantics and Lemmas 6.3.8 and 6.3.7, we know that there is a transition $q' \xrightarrow{a,g,r} q$ in $\mathcal{A}$, and there are C'', D'' such that $(v, w) \models I(q)$, $(v, w) \in [\![C'', D''[r]\uparrow]\!]$ and $(C', D') \overset{g}{\Leftarrow}_c (C'', D'')$. By Lemma 6.3.13, we get that there exists a $d \geq 0$ and $w'' \in [\![D''[r]]\!]_v$ such that $w'' + d = w$. Since $(v, w) \models I(q)$ and invariants in a PTA only give upper bounds on clocks, also $(v, w'') \models I(q)$. It follows that (q, w'') is a state of the concrete semantics $[\![\mathcal{A}]\!]_v$ and $(q, w'') \xrightarrow{d} (q, w)$. Using Lemma 6.3.12 we get that there exists a $w' \in [\![D'']\!]_v$ such that $w'' = w'[r]$. Using Lemma 6.3.8 and Lemma 6.3.7 again, it follows that $(v, w') \models g$ and $(v, w') \in [\![C', D']\!]$. Since (q', C', D') is a state of the symbolic semantics, $(v, w') \models I(q')$. Hence, (q', w') is a state of the concrete semantics and $(q', w') \xrightarrow{a} (q, w'')$ is a transition in the concrete semantics. Combination of this transition with the transition $(q, w'') \xrightarrow{d} (q, w)$ gives the required path in the concrete semantics. □

Proposition 6.3.17 *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).*

PROOF: By induction on the number of transitions in the run.

As basis we consider a run with 0 transitions, i.e., a run that consists of an initial state of the symbolic semantics. So this means that (q, C, D) is an initial state. The induction basis now directly follows using Lemma 6.3.15.

For the induction step, assume that we have a run in the symbolic semantics, ending with a transition $(q', C', D') \to (q, C, D)$. By $(v, w) \in [\![C, D]\!]$ and Lemma 6.3.16, there exists a pair $(v, w') \in [\![C, D]\!]$ such that in the concrete semantics $[\![\mathcal{A}]\!]_v$ there is a path from (q', w') to (q, w). By induction hypothesis, there is a path in the concrete semantics leading up to state (q', w'). Extension of this path with the path from (q', w') to (q, w) gives the required path in the concrete semantics. □

Conversely, 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 6.3.18 *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 $(v, w) \in [\![C, D]\!]$.*

PROOF: In any execution in the concrete semantics, we can always insert zero delay transitions at any point. Also, two consecutive delay transitions $(q, w) \xrightarrow{d} (q, w + d)$ and $(q, w + d) \xrightarrow{d'} (q, w + d + d')$ can always be combined into a single delay transition $(q, w) \xrightarrow{d+d'} (q, w + d + d')$. Therefore, without loss of generality, we only consider concrete executions that start with a delay transition, and in which there is a strict alternation of action transitions and delay transitions. The proof is by induction on the number of action transitions.

As basis we consider a run $(q_0, w_0) \xrightarrow{d} (q_0, w_0 + d)$, where $w_0 = \lambda x.0$, consisting of a single time-passage transition. By definition of the concrete semantics, $(v, w_0 + d) \models I(q_0)$. Using Lemma 6.3.13, we have that $(v, w_0 + d) \in [\![C_0, \mathsf{E}\uparrow]\!]$ since $(v, w_0) \in [\![C_0, \mathsf{E}]\!]$. From $(v, w_0 + d) \in [\![C_0, \mathsf{E}\uparrow]\!]$ and $(v, w_0 + d) \models I(q_0)$, using Lemma 6.3.7 and Lemma 6.3.8 we get that there exists C, D such that $(C_0, \mathsf{E}\uparrow) \overset{I(q_0)}{\Leftarrow} (C, D)$ and $(v, w_0 + d) \in [\![C, D]\!]$. By definition, (C, D) is an initial state of the symbolic semantics. This completes the proof of the induction basis.

For the induction step, assume that the run in the concrete semantics of $[\![\mathcal{A}]\!]_v$ ends with transitions $(q'', w'') \xrightarrow{a} (q', w') \xrightarrow{d} (q, w)$. By induction hypothesis, there exists a run in the symbolic semantics ending with a state (q'', C'', D'') such that $(v, w'') \in [\![C'', D'']\!]$.

By definition of the concrete semantics, there is a transition $q'' \xrightarrow{g,a,r} q'$ in $\mathcal{A}$ such that $(v, w'') \models g$ and $w' = w''[r]$. Moreover, we have $q' = q$, $w = w' + d$ and $(v, w) \models I(q)$. Using Lemma 6.3.7 and Lemma 6.3.8 gives that there exists C', D' such that $(C'', D'') \overset{g}{\Leftarrow}_c (C', D')$ and $(v, w'') \in [\![C', D']\!]$. By Lemma 6.3.12, we have $w' \in [\![D'[r]]\!]_v$. Moreover, by Lemma 6.3.13, $w \in [\![D'[r]\uparrow]\!]_v$. Using $(v, w) \models I(q)$, Lemma 6.3.7 and Lemma 6.3.8, we infer that there exists C, D such that $(v, w) \in [\![C, D]\!]$ and $(C', D'[r]\uparrow) \overset{I(q)}{\Leftarrow}_c (C, D)$.

Finally, using the definition of the symbolic semantics, we infer the existence of a transition $(q'', C'', D'') \rightarrow (q, C, D)$.

□

Example 6.3.19 Figure 6.5 shows the symbolic state-space of the automaton in Fig. 6.1 represented by constrained PDBMs. In the initial state the invariant $x \leq p$ limits the value of x, and since both clocks have the same value also the value of y. When taking the transition from $S0$ to $S1$ we have to compare the parameters p and q. This leads to a split where in the one case no state is reachable since the region is empty, and in the other (when $q \leq p$) $S1$ can be reached. From then on no more splits occur and only one new state is reachable.

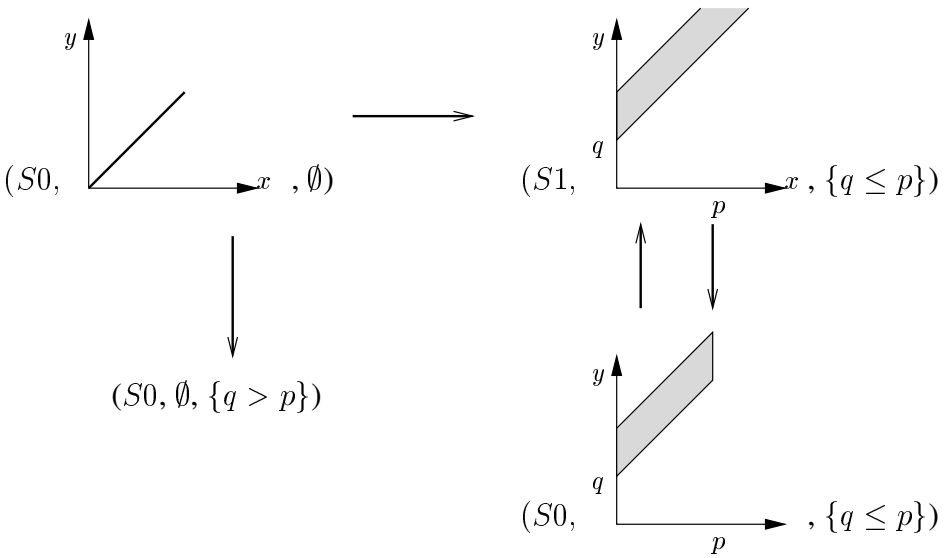

Figure 6.5: The symbolic state space of the PTA in Fig. 6.1.

6.3.4 Evaluating state formulas

We now define the predicate $\overset{\phi}{\Leftarrow\!\!\shortmid}$ which relates a symbolic state and a state formula ϕ (as defined in Definition 6.2.7) to a collection of symbolic states that satisfy ϕ.

In order to check whether a state formula holds, we break it down into its atomic subformulas, 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.

Definition 6.3.20 State formula ϕ is in *normal form* if all $\neg$ signs in ϕ appear only in subformulae of the form $\neg q$.

Since each simple guard with a $\neg$ sign in front can be rewritten to equivalent simple guard without, for each state formula there is an equivalent one in normal form.

In the following, let f be a simple guard, and ϕ be in normal form.

$$(Q_1)\frac{}{(q,C,D) \overset{q}{\Leftarrow} (q,C,D)} \qquad (Q_2)\frac{q \neq q'}{(q,C,D) \overset{\neg q'}{\Leftarrow} (q,C,D)}$$

$$(Q_3)\frac{(C,D) \overset{f}{\Leftarrow}_c (C',D')}{(q,C,D) \overset{f}{\Leftarrow} (q,C',D')}$$

$$(Q_4)\frac{(q,C,D) \overset{\phi_1}{\Leftarrow} (q,C',D') \ , \ (q,C',D') \overset{\phi_2}{\Leftarrow} (q,C'',D'')}{(q,C,D) \overset{\phi_1 \wedge \phi_2}{\Leftarrow} (q,C'',D'')}$$

$$(Q_5)\frac{(q,C,D) \overset{\phi_1}{\Leftarrow} (q,C',D')}{(q,C,D) \overset{\phi_1 \vee \phi_2}{\Leftarrow} (q,C',D')} \qquad (Q_6)\frac{(q,C,D) \overset{\phi_2}{\Leftarrow} (q,C',D')}{(q,C,D) \overset{\phi_1 \vee \phi_2}{\Leftarrow} (q,C',D')}$$

The following lemma gives the soundness and completeness of relation $\overset{\phi}{\Leftarrow}$.

Lemma 6.3.21 *Let ϕ be a state formula in normal form, q a location and (C,D) a constrained PDBMs. Let $[\![q,\phi]\!]$ denote the set $\{(v,w) \mid (q,w) \models_v \phi\}$. Then*

$$[\![C,D]\!] \cap [\![q,\phi]\!] \quad = \quad \bigcup \{ [\![C',D']\!] \mid (q,C,D) \overset{\phi}{\Leftarrow} (q,C',D') \}.$$

PROOF: "$\subseteq$": Assume that $(v,w) \in [\![C,D]\!]$ and $(q,w) \models_v \phi$. We prove that there are C', D' such that $(v,w) \in [\![C',D']\!]$ and $(q,C,D) \overset{\phi}{\Leftarrow} (q,C',D')$. We proceed by induction on the structure of ϕ.

- Base cases.
  - Suppose $\phi = q'$. As $(q,w) \models_v q'$, clearly, $q = q'$. Since, by rule Q_1, $(q,C,D) \overset{q}{\Leftarrow} (q,C,D)$, we can take $C = C'$ and $D = D'$ and the result follows.
  - Suppose $\phi = \neg q'$. Similar to the previous case, apply rule Q_2.
  - Suppose $\phi = f$ with f a simple guard. Then $(v,w) \in [\![C,D]\!]$ and $(v,w) \models f$. By Lemma 6.3.7 there exist C'', D'' such that $(C,D) \overset{f}{\Leftarrow} (C'',D'')$ and $(v,w) \in [\![C'',D'']\!]$. Lemma 6.3.8 yields the existence of C', D' with $(C'',D'') \rightarrow_c (C',D')$ and $(v,w) \in [\![C',D']\!]$. Now, application of rule Q_3 yields $(q,C,D) \overset{f}{\Leftarrow} (q,C',D')$.
- Induction step.
  - Suppose $\phi = \phi_1 \wedge \phi_2$. Then $(q,w) \models_v \phi_1$ and $(q,w) \models_v \phi_2$. By applying the induction hypothesis on ϕ_1, we derive that there exist C'', D'' such that $(q,C,D) \overset{\phi_1}{\Leftarrow} (q,C'',D'')$ and $(v,w) \in [\![C'',D'']\!]$. Applying the induction hypothesis on ϕ_2 yields the existence of C', D' such that $(q,C'',D'') \overset{\phi_2}{\Leftarrow} (q,C',D')$ and $(v,w) \in [\![C',D']\!]$. Then by application of rule Q_4 also $(q,C,D) \overset{\phi_1 \wedge \phi_2}{\Leftarrow} (q,C',D')$.

– Suppose $\phi = \phi_1 \vee \phi_2$. Then $(q,w) \models_v \phi_1$ or $(q,w) \models_v \phi_2$. Suppose that $(q,w) \models_v \phi_1$. The induction hypothesis yields the existence of C', D' such that $(q,C,D) \overset{\phi_1}{\Leftarrow} (q,C',D')$ and $(v,w) \in [\![C',D']\!]$. Then, by application of rule Q_5 $(q,C,D) \overset{\phi_1 \vee \phi_2}{\Leftarrow} (q,C',D')$. The case $(q,w) \models \phi_2$ is similar (using rule Q_6).

"$\supseteq$": Assume $(q,C,D) \overset{\phi}{\Leftarrow} (q,C',D')$ and $(v,w) \in [\![C',D']\!]$. By induction on the structure of the derivation of $\overset{\phi}{\Leftarrow}$, we establish that $(v,w) \in [\![C,D]\!]$ and $(q,w) \models_v \phi$.

- Base cases. The derivation consists of a single step r.
  - $r = Q_1$. Then $\phi = q$, $C = C'$, $D = D'$. Trivially $(v,w) \in [\![C,D]\!]$ and $(q,w) \models_v q$.
  - $r = Q_2$. Similar to the previous case.
  - $r = Q_3$. Suppose $\phi = f$ with f a simple guard. Then $(C,D) \overset{f}{\Leftarrow}_c (C',D')$. This means that there are C'', D'' with $(C,D) \overset{f}{\Leftarrow} (C'',D'')$ and $(C'',D'') \rightarrow_c (C',D')$. By Lemma 6.3.8 we have $(v,w) \in [\![C'',D'']\!]$. Then we have by Lemma 6.3.7 that $(v,w) \models f$ and $(v,w) \in [\![C,D]\!]$.

- Induction step. Consider the last rule r in the derivation of $(q,C,D) \overset{\phi}{\Leftarrow} (q,C',D')$.
  - $r = Q_4$. Then $\phi = \phi_1 \wedge \phi_2$ and $(q,C,D) \overset{\phi_1}{\Leftarrow} (q,C'',D'')$ and $(q,C'',D'') \overset{\phi_2}{\Leftarrow} (q,C',D')$ for some C'', D''. Applying the induction hypothesis to the second statement yields that $(q,w) \models_v \phi_2$ and $(v,w) \in [\![C'',D'']\!]$. Then applying the induction hypothesis to the first statement yields $(q,w) \models_v \phi_1$ and $(v,w) \in [\![C,D]\!]$. Then also $(q,w) \models_v \phi_1 \wedge \phi_2$.
  - $r = Q_5$. Then $\phi = \phi_1 \vee \phi_2$. Then $(q,C,D) \overset{\phi_1}{\Leftarrow} (q,C',D')$. By induction hypothesis we have $(q,w) \models_v \phi_1$ and $(v,w) \in [\![C,D]\!]$.
  - $r = Q_6$. Similarly to the previous case.

□

6.3.5 Algorithm

We are now in a position to present our model checking algorithm for parametric timed automata. The algorithm displayed in Fig. 6.6 describes how our tool explores the symbolic state space and searches for constraints on the parameters for which a reachability property $\exists\Diamond\phi$ holds in a PTA $\mathcal{A}$.

In the algorithm, we use inclusion between symbolic states defined by

$$(q,C,D) \subseteq (q',C',D') \quad \triangleq \quad q = q' \wedge [\![C,D]\!] \subseteq [\![C,D']\!].$$

Note that whenever a triple (q,C,D) ends up in one of the lists maintained by the algorithm, (C,D) is a constrained PDBM in canonical form. This fact, in combination with Lemma 6.3.6, gives decidability of the inclusion operation. Our search algorithm explores the symbolic semantics in an "intelligent" manner, and stops whenever it reaches a state

REACHABLE ($\mathcal{A}$, ϕ)

RESULT := $\emptyset$, PASSED := $\emptyset$, WAITING := $\{(q_0, C, D) \mid (C_0, \mathsf{E}\uparrow) \overset{I(q_0)}{\Leftarrow\!\!\!\mid}_c (C, D)\}$
while WAITING $\neq \emptyset$ **do**
  select (q, C, D) from WAITING
  RESULT := RESULT $\cup \{(q', C', D') \mid (q, C, D) \overset{\phi}{\Leftarrow\!\!\!\mid} (q', C', D')\}$
  FALSE := $\{(q', C', D') \mid (q, C, D) \overset{\neg\phi}{\Leftarrow\!\!\!\mid} (q', C', D')\}$
  for each (q', C', D') in FALSE **do**
    if for all (q'', C'', D'') in PASSED: $(q', C', D') \not\subseteq (q'', C'', D'')$ **then**
      add (q', C', D') to PASSED
      for each (q'', C'', D'') such that $(q', C', D') \rightarrow (q'', C'', D'')$ **do**
        WAITING := WAITING $\cup \{(q'', C'', D'')\}$
return RESULT

Figure 6.6: The parametric model checking algorithm

whose semantics is contained in the semantics of a state that has been encountered before. Despite this, our algorithm need not terminate.

If it terminates, the result returned by the algorithm is a set of satisfiable symbolic states, all of which satisfy ϕ, for any valuation of the parameters and clocks in the state.

Theorem 6.3.22 *Suppose (q, C, D) is in the result set returned by* REACHABLE *($\mathcal{A}$, ϕ). Then (C, D) is satisfiable. Moreover, for all $(v, w) \in [\![C, D]\!]$, (q, w) is a reachable state of $[\![\mathcal{A}]\!]_v$ and $(q, w) \models_v \phi$.*

PROOF: It is easy to see that all the symbolic states returned by the algorithm are satisfiable: the only operation that may modify the constraint set is adding a guard, but this will never lead to unsatisfiable constraint sets. Since all constrained PDBMs returned by the algorithm are in canonical form, they are all satisfiable by Lemma 6.3.5.

Suppose that $(v, w) \in [\![C, D]\!]$. By a straightforward inductive argument, using Lemmas 6.3.15, 6.3.16 and 6.3.21, it follows that (q, w) is a reachable state of $[\![\mathcal{A}]\!]_v$ and $(q, w) \models_v \phi$. □

For invariance properties $\forall\Box\phi$, our tool runs 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 6.2.2, 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.

A difference between the above algorithm and the standard timed model checking algorithm is that we continue the exploration until either no more new states are found or all paths end in a state satisfying the property. This is because we want to find all the possible constraints on the parameters for which the property holds. Also, the operations on non-parametric DBMs only change the DBM they are applied to, whereas in our case, we may end up with a set of new PDBMs and not just one.

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. 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. In fact, for unparameterized timed automata this trick *guarantees* termination [AD94, Alu98].

6.4 Lower Bound / Upper Bound Automata

This section introduces the class of *lower bound/upper bound (L/U) automata* and describes several (rather intuitive) observations that simplify the parametric model checking problem for PTAs in this class. Our results use the possibility to eliminate parameters in certain cases. This is a relevant issue, because the complexity of parametric model checking grows very fast in the number of parameters. Moreover, our observations yield some decidability results for L/U automata, where the corresponding problems are undecidable for general PTAs. The applicability of the results is illustrated by the verification of Fischer's algorithm.

6.4.1 Lower bound/Upper bound Automata

Informally, each parameter in an L/U automaton $\mathcal{A}$ occurs either as a lower bound in the invariants and guards of $\mathcal{A}$ or as an upper bound, but never as both. For instance, p is an upper bound parameter in $x - y < 2p$. Lower bound parameters are for instance q and q' in $y - x > q + 2q'$ ($\equiv x - y < -q - 2q'$) and in $x - y < 2p - q - 2q'$. A PTA containing both the guards $x - y \leq p - q$ and $z < q - p$ is not an L/U automaton.

Definition 6.4.1 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 $\mathcal{A}$ is a parameter that only occurs positively in $\mathcal{A}$. We call $\mathcal{A}$ a *lower bound/upper bound (L/U) automaton* if every parameter occurring in $\mathcal{A}$ is either a lower bound parameter or an upper bound parameter.

Example 6.4.2 The PTA in Fig. 6.7 is an L/U automaton, where *min* is a lower bound parameter and *max* is an upper bound parameter. The model of Fischer's algorithm in Fig. 6.3 is also an L/U automaton. Here *min_rw* and *min_delay* are the lower bound parameters and *max_rw* and *max_delay* are the upper bound parameters.

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 $L \cup U = P$. Furthermore, we consider, apart from parameter valuations, also *extended parameter valuations*. Intuitively, an extended parameter valuation is a parameter valuation with values in $\mathbb{R}^{>0} \cup \{\infty\}$, rather than in $\mathbb{R}^{>0}$. Extended parameter valuations are useful in certain cases to solve the verification problem (over non-extended valuations) stated in Section 6.2.3. Working with extended parameter valuations may cause the evaluation of an expression to be undefined. For example, the expression $e[v]$ is not defined for $e = p - q$ and $v(p) = v(q) = \infty$. We therefore require that an extended parameter valuation does not assign the value ∞ to both a lower bound parameter and an upper bound parameter. Then we can easily extend notions $e[v]$, $(v, w) \models e$ and $\mathcal{A}[v]$ (defined in Section 6.2) to extended valuations. Here, we use the conventions that $0 \cdot \infty = 0$, that $x - y \prec \infty$ evaluates to true and $x - y \prec -\infty$ to false. In particular, we have $[\![\mathcal{A}]\!]_v = [\![\mathcal{A}[v]]\!]$ for extended valuations v and L/U automata $\mathcal{A}$. Moreover, we extend the orders $\sim$ to $\mathbb{R} \cup \{\infty\}$ in the usual way and we extend them to

extended parameter valuations via point wise extension (i.e. $v \sim v'$ iff $v(p) \sim v'(p)$ for all $p \in P$). We denote an extended valuation of an L/U automaton by a pair (λ, μ), which equals the function λ on the lower bound parameters and μ on the upper bound parameters. 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 LTS 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 LTS whose reachable states are a subset of those of $\mathcal{A}$. The result crucially depends on the fact that state formulae (by definition) do not contain parameters. The usefulness of this result (and of several other results in this section) lies in the fact that the satisfaction of a property for infinitely many extended parameter valuations (λ', μ') is reduced to its satisfaction for a single valuation (λ, μ).

Proposition 6.4.3 *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. $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \models \forall\Box\phi \iff \forall \lambda \leq \lambda', \mu' \leq \mu : [\![\mathcal{A}]\!]_{(\lambda',\mu')} \models \forall\Box\phi.$

PROOF: (sketch) The "$\Longleftarrow$" parts of both statements are trivial. The crucial observation for both "$\Longrightarrow$" parts is the following. For all linear expressions e in $\mathcal{A}$ and all extended parameter valuations (λ, μ), (λ', μ') with $\lambda' \leq \lambda$ and $\mu \leq \mu'$, we have that $e[\lambda, \mu] \leq e[\lambda', \mu']$. Therefore, if $((\lambda, \mu), w) \models x - y \prec e$, then $((\lambda', \mu'), w) \models x - y \prec e$.
□

The following example illustrates how Proposition 6.4.3 can be used to eliminate parameters in L/U automata.

Example 6.4.4 Reconsider the L/U automaton in Fig. 6.7. 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 6.4.3(1) (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. The next example shows that $[\![\mathcal{A}']\!]_v \models \exists\Diamond S_2$ for all valuations v, which essentially means that $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \models \exists\Diamond S_2$ for all λ, μ such that $\mu(max) = \lambda(min) < \infty$. From this fact, Proposition 6.4.3(1) concludes that $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \models \exists\Diamond S_2$ for all λ, μ with $\lambda(min) \leq \mu(max)$ and $\lambda(min) < \infty$.

The question whether there exists a (non-extended) parameter valuation such that a given location q is reachable, is known as the *emptiness problem* for PTAs. In [AHV93], it is shown that the emptiness problem is undecidable for PTAs with three clocks or more. The following proposition implies that we can solve the emptiness problem for an L/U automaton $\mathcal{A}$ by only considering $\mathcal{A}[(0, \infty)]$, which is a non-parametric timed automaton. Since reachability is decidable for timed automata ([AD94]), 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

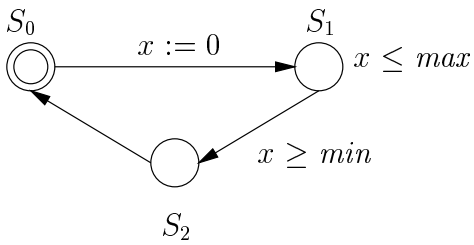

Figure 6.7: Reducing parametric to non-parametric model checking

universality problem for invariance properties, asking whether an invariance property holds for all parameter valuations.

Proposition 6.4.5 *Let $\mathcal{A}$ be an L/U automaton with location q. Then $\mathcal{A}[(0,\infty)] \models \exists\Diamond q$ if and only if there exists a (non-extended) parameter valuation (λ,μ) such that $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \models \exists\Diamond q$.*

PROOF: The "only if" part is an immediate consequence of Proposition 6.4.3(1) and the fact that $[\![\mathcal{A}[(0,\infty)]]\!] = [\![\mathcal{A}]\!]_{(0,\infty)}$. For the "if" part, assume that α is a run of $[\![\mathcal{A}[(0,\infty)]]\!]$ that reaches the location q. Let T' be the smallest constant occurring in $\mathcal{A}$ and let T be the maximum clock value occurring in α. (More precisely, if $\alpha = s_0 a_1 s_1 a_2 \ldots a_N s_N$ and $s_i = (q_i, w_i)$, then $T = \max_{i\leq N, x\in X} w_i(x)$; T' compensates for negative constants t_0 in expressions e of $\mathcal{A}$.) Now, take $\lambda(l_j) = 0$ and $\mu(u_j) = T + |T'| + 1$. Let $i \leq N$ and $g = x - y \prec e$ be the invariant associated with a state s_i occurring in α or the guard associated with the i^{th} transition taken by α. One easily shows that, since $w_i(x) - w_i(y) \prec e[0,\infty]$, also $w_i(x) - w_i(y) \prec e[\lambda,\mu]$, that is $((\lambda,\mu), w_i) \models g$. Hence, α is a run of $[\![\mathcal{A}]\!]_{(\lambda,\mu)}$, so $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \models \exists\Diamond q$.
□

Corollary 6.4.6 *The emptiness problem is decidable for L/U automata.*

Definition 6.4.7 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 $t_i \in \mathbb{Z}$.

The following proposition is basically the observation in [AD94], that multiplication of each constant in a timed automaton and in a system property with the same positive factor preserves satisfaction.

Proposition 6.4.8 *Let $\mathcal{A}$ be fully parametric PTA. Then for all parameter valuations v and all system properties ψ*

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

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.

PROOF: It is easy to see that for all $t \in \mathbb{R}^{>0}$, $\alpha = s_0 a_1 s_1 a_2 \dots a_N s_N$ with $s_i = (q_i, w_i)$ is a run of $[\![\mathcal{A}]\!]_v$ if and only if $s'_0 a_1 s'_1 \dots a_N s'_N$ is a run of $[\![\mathcal{A}]\!]_{t \cdot v}$, where $s'_i = (q_i, t \cdot w_i)$ and $t \cdot w_i$ denotes $x \mapsto t \cdot w_i(x)$. □

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 $\mathcal{A}[1] \models \psi$. The need for a separate treatment of the value 0 is illustrated by the (fully parametric) automaton with a single transition equipped with the guard $x < p$. The target location of the transition is reachable for any value of p, except for $p = 0$.

Corollary 6.4.9 *For a fully parametric PTA $\mathcal{A}$ with one parameter, a constraint set C and a property ψ without constants (except 0), it is decidable whether $\forall v \in [\![C]\!] : [\![\mathcal{A}]\!]_v \models \psi$.*

Example 6.4.10 The PTA $\mathcal{A}'$ mentioned in Example 6.4.4 is a fully parametric timed 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 6.4.8 implies that $\mathcal{A}'[v] \models \exists\Diamond S_2$ for all v. As shown in Example 6.4.4, 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 same methods as above to verify that $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \not\models \exists\Diamond S_2$ if $\lambda(min) > \mu(max)$. However, we can not take $min = max$ in this case, since the bound in the constraint is a strict one. The following definition and results allows us to move the strictness of a constraint into the PTA.

Definition 6.4.11 Let $P' \subseteq P$ be a set of parameters. Define $\mathcal{A}^{<}_{P'}$ as the PTA obtained from $\mathcal{A}$ 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 from P'. Similarly, define $\mathcal{A}^{\leq}_{P'}$ as the PTA obtained from $\mathcal{A}$ by replacing every inequality $x - y < e$ by a non–strict inequality $x - y \leq e$, provided that e contains at least one parameter from P'. For $\prec = <, \leq$, write $\mathcal{A}^{\prec}$ for $\mathcal{A}^{\prec}_{P}$. Moreover, define $v \prec_{P'} v'$ by $v(p) \prec v'(p)$ if $p \in P'$ and $v(p) = v'(p)$ otherwise.

Proposition 6.4.12 *Let $\mathcal{A}$ be an L/U automaton. Then*

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

PROOF:

1 Let e be a linear expression occurring in $\mathcal{A}$. Then we can write $e = t_0 + e_1 + e_2$, where $t_0 \in \mathbb{Z}, e_1$ is an expression over the upper bound parameters and e_2 an expression over the lower bound parameters. Then we have

$$\begin{aligned} \mu \leq \mu' &\implies e_1[\mu] \leq e_1[\mu'], \\ \lambda' \leq \lambda &\implies e_2[\lambda'] \leq e_2[\lambda], \\ \lambda' \leq \lambda, \mu \leq \mu' &\implies e[(\lambda, \mu)] \leq e[(\lambda', \mu')]. \end{aligned}$$

If there is at least one parameter occurring respectively in e_1 or e_2 then respectively

$$\begin{aligned} \mu < \mu' &\implies e_1[\mu] < e_1[\mu'] \\ \lambda' < \lambda &\implies e_2[\lambda] < e_2[\lambda']. \end{aligned}$$

Thus if there is at least one parameter occurring in e, then

$$\lambda' < \lambda, \mu < \mu' \implies e[(\lambda, \mu)] < e[(\lambda', \mu')].$$

Now, let (λ, μ) be an extended valuation. Let $g \equiv x - y \prec e$ be a simple guard occurring in $\mathcal{A}^{\leq}$ and let $g' \equiv x - y \prec' e$ be the corresponding guard in $\mathcal{A}$. Assume that $(w, (\lambda, \mu)) \models g$, i.e. $w(x) - w(y) \prec e[(\lambda, \mu)]$. We show that $(w, (\lambda, \mu)) \models g'$. We distinguish two cases.

case 1: There exists a parameter occurring in e. Then $w(x) - w(y) \prec e[(\lambda, \mu)] < e[(\lambda', \mu')]$. Then certainly $(w, (\lambda, \mu)) \models g' \equiv x - y \prec' e$.

case 2: The expression e does not contain any parameter. Then $g' \equiv g$ and hence $(w, (\lambda, \mu)) \models g'$.

Now it easily follows that every run of $[\![\mathcal{A}^{\leq}]\!]_{(\lambda,\mu)}$ is also a run of $[\![\mathcal{A}]\!]_{(\lambda',\mu')}$. Thus, if a state satisfying ψ is reachable in $[\![\mathcal{A}^{\leq}]\!]_{(\lambda,\mu)}$ then it is also reachable in $[\![\mathcal{A}]\!]_{(\lambda',\mu')}$.

2, $\implies$: This follows from statement (1) of this proposition: assume that $[\![\mathcal{A}^{<}]\!]_{(\lambda,\mu)} \models \forall\Box\phi$ and let λ', μ' be such that $\lambda < \lambda'$, $\mu' < \mu$. Since $[\![\mathcal{A}^{<}]\!]_{(\lambda,\mu)} \not\models \exists\Diamond\neg\phi$, we have

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

Then contraposition of statement (1) together with $(\mathcal{A}^{<})^{\leq} = \mathcal{A}^{\leq}$ yields $[\![\mathcal{A}^{\leq}]\!]_{(\lambda',\mu')} \not\models \exists\Diamond\neg\phi$. As $\mathcal{A}$ imposes stronger bounds than $\mathcal{A}^{\leq}$, also $[\![\mathcal{A}]\!]_{(\lambda',\mu')} \not\models \exists\Diamond\neg\phi$, which means $[\![\mathcal{A}]\!]_{(\lambda',\mu')} \models \forall\Box\phi$.

2, $\impliedby$: Let (λ, μ) be an extended valuation and assume that $[\![\mathcal{A}]\!]_{(\lambda',\mu')} \models \forall\Box\phi$ for all $\lambda' > \lambda$, $\mu' < \mu$. Assume that α is a run of $[\![\mathcal{A}^{<}]\!]_{(\lambda,\mu)}$. We construct $\lambda' > \lambda$ and $\mu' < \mu$ such that α is also a run of $[\![\mathcal{A}]\!]_{(\lambda',\mu')}$. (Then we are done: since $[\![\mathcal{A}]\!]_{(\lambda',\mu')} \models \forall\Box\phi$, the last state of α satisfies ϕ. Hence every reachable state of $[\![\mathcal{A}]\!]_{(\lambda,\mu)}$ satisfies ϕ, i.e. $[\![\mathcal{A}]\!]_{(\lambda,\mu)} \models \forall\Box\phi$.)

We use the following notation. We write $v = (\lambda, \mu)$ and $v' = (\lambda', \mu')$. For a run α, we write $\alpha = s_0 a_1 s_1 a_2 \dots a_N s_N$ with $s_i = (q_i, w_i)$, $I(q_i) = \wedge_j^J I_{ij}$, $I_{ij} = x_{ij} \prec_{ij} E_{ij}$. As α is a run, either $a_{i+1} \in \mathbb{R}$ or there exists a transition $q_i \xrightarrow{g_{i-1}, a_{i+1}, r_{i+1}} q_{i+1}$ for each i, $1 \leq i \leq N$. We write the guard on this transition by $g_i = \wedge_j^J g_{ij}$, $g_{ij} = x_{ij} - y_{ij} \prec_{ij} e_{ij}$. Finally, if g is a guard or invariant in $\mathcal{A}$, then we denote the corresponding guard or invariant in $\mathcal{A}^{<}$ by $g^{<}$, i.e. the guard that is obtained as in Definition 6.4.11.

If neither the guards g_{ij} nor the invariants I_{ij} contains a parameter, then we can take v' arbitrarily and we have that α is a run of $[\![\mathcal{A}]\!]_{v'}$. Therefore, assume that at least one of the guards g_{ij} or invariants I_{ij} contains a parameter. Then, by definition of $\mathcal{A}^{<}$, this guard or invariant contains a strict bound. In this case, we construct $\lambda' > \lambda$ and $\mu' < \mu$ such that $w_i(x - y) < e[(\lambda', \mu')] < e[(\lambda, \mu)]$ if $g = x - y < e$ is an invariant I_{ij} or guard g_{ij} as above. Informally, we use the minimum "distance" $e[(\lambda, \mu)] - w_i(x - y)$ occurring in α to slightly increase the lower bounds and slightly decrease the upper bounds yielding $\lambda < \lambda'$ and $\mu < \mu'$.

Let

$$\begin{aligned} T_0 &= \min_{i\leq N, j\leq J'} \{E_{ij}[v] - w_i(x_{ij}) \quad | \quad \prec_{ij} = <\}, \\ T_1 &= \min_{i\leq N, j\leq <J} \{e_{ij}[v] - w_i(x_{ij} - y_{ij}) \quad | \quad \prec_{ij} = <\}, \\ 0 &< T < \min\{T_0, T_1\}, \end{aligned}$$

with the convention that $\min \emptyset = \infty$. At least one of the inequalities $\prec_i\ j$ is strict, since at least one of the guards contains a parameter. Hence $T_0 < \infty$ or $T_1 < \infty$. Since $(v, w_i) \models I_{ij} \wedge g_{ij}$, we have we have that $T_0 > 0$ and $T_1 > 0$. Hence $\infty > \min\{T_0, T_1\} > 0$ and the requested T exists. The crucial property is that if $g_{ij} \equiv x_{ij} - y_{ij} < e_{ij}$ or $g_{ij} \equiv x_{ij} - y_{ij} < E_{ij}$ we have respectively

$$\begin{aligned} T &< e_{ij}[v] - w_i(x_{ij} - y_{ij}) \\ T &< E_{ij}[v] - w_i(x_{ij} - y_{ij}). \end{aligned}$$

Now, let T' be the sum of the constants appearing in the guards and invariants that appear in the run α i.e.

$$T' = \sum_{i\leq N, j\leq J'} \mathrm{sum_of_const(E_{ij})} + \sum_{\mathrm{i\leq n, j\leq J}} \mathrm{sum_of_const(e_{ij})},$$

where $\mathrm{sum_of_const}(\mathrm{t_0 + t_1 \cdot p_1 + \cdots + t_n \cdot p_n}) = |\mathrm{t_1}| + \cdots + |\mathrm{t_n}|$. Since at least one of the guards or invariants contains a parameter, we have $T' > 0$.

Now, take $v' = (\lambda + \frac{T}{T'}, \mu - \frac{T}{T'})$ and consider $g_{ij} \equiv x_{ij} - y_{ij} \prec_{ij} e_{ij}$. We claim that $(v', w_i) \models g_{ij}$.

case 1: The expression g_{ij} does not contain any parameter. Then $g_{ij} = g_{ij}^{<}$ and $e_{ij}[v] = e_{ij}[v']$. Since $(w_i, v) \models g_{ij}$, also $(w_i, (v')) \models g_{ij}^{<}$.

case 2: There exists a parameter occurring in e. We can write $e = t_0 + t_1 \cdot u_1 + \cdots + t_M \cdot u_M - t'_1 \cdot l_1 - \cdots - t'_K \cdot l_K$, with $t_i \geq 0$, $t'_i \geq 0$ for $i > 0$. Then

$$\begin{aligned} e_{ij}[v'] &= t_0 + \sum_{k=1}^{M} t_k \cdot (\mu'_k - \frac{T}{T'}) - \sum_{k=1}^{K} t_k \cdot (\lambda'_k + \frac{T}{T'}) \\ &= (t_0 + \sum_{k=1}^{M} t_k \cdot \mu'_k - \sum_{k=1}^{K} t_k \cdot \lambda'_k) - \frac{T}{T'} \cdot (\sum_{k=1}^{M} t_k + \sum_{k=1}^{K} t'_k) \\ &\geq e_{ij}[v] - T \\ &> e_{ij}[v] - (e_{ij}[v] - w_i(x_{ij} - y_{ij})) \\ &= w_i(x_{ij} - y_{ij}). \end{aligned}$$

Therefore $(w_i, v') \models x_{ij} - y_{ij} < g_{ij}^{<}$ and then also $(w_i, v') \models x_{ij} - y_{ij} \prec_{ij} g_{ij}^{<}$.

Combining the cases (1) and (2) yields that for all i, j, $(w_i, v') \models x_{ij} - y_{ij} \prec_{ij} g_{ij}^{<}$. Similarly, one proves that $(w_i, v') \models x_{ij} - y_{ij} \prec_{ij} I_{ij}$. Therefore, α is a run of $[\![\mathcal{A}]\!]_{(\lambda', \mu')}$.

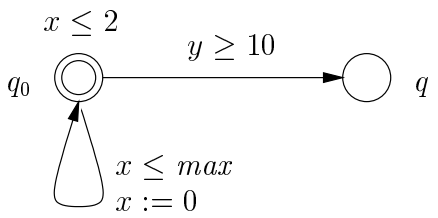

Figure 6.8: The converse of Proposition 6.4.12(1) does not hold.

□

The previous result concerns the automaton that is obtained when all the strict inequalities in the automaton are changed into nonstrict ones (or the other way around). Sometimes, we want to "toggle" only some of the inequalities. Then the following result can be applied.

Corollary 6.4.13 *Let $\mathcal{A}$ be an L/U automaton and $P' \subseteq P$.*

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

PROOF: Let (λ, μ) be an extended valuation. Let $\mathcal{A}_0$ be the automaton obtained from $\mathcal{A}$ by substituting p by $(\lambda, \mu)(p)$ for every $p \notin P'$. Then $[\![\mathcal{A}^{\leq}_{P'}]\!]_{(\lambda,\mu)} = [\![\mathcal{A}^{<}_{0}]\!]_{(\lambda,\mu)}$ and $[\![\mathcal{A}^{\leq}_{P'}]\!]_{(\lambda,\mu)} = [\![\mathcal{A}^{\leq}_{0}]\!]_{(\lambda,\mu)}$. Now the result follows by applying Proposition 6.4.12 to $\mathcal{A}_0$. □

The following example shows that the converse of Proposition 6.4.12(1) does not hold.

Example 6.4.14 Consider the automaton $\mathcal{A}$ in Fig. 6.8. Recall that the clocks x and y are initially 0. Then $\mathcal{A} = \mathcal{A}^{\leq}$ and the location q is reachable if $max > 0$ but not if $max = 0$. This is so because if $max = 0$, then clock y is never augmented. Thus $\forall\lambda' < 0, 0 < \mu' : [\![\mathcal{A}]\!]_{(\lambda',\mu')} \models \exists\Diamond\phi$, but not $[\![\mathcal{A}^{\leq}]\!]_{(0,0)} \models \exists\Diamond\phi$.

We believe that the class of L/U automata can be very useful in practice. Several examples known from the literature fall into this class, or can be modeled slightly differently to achieve this. We mention the root contention protocol [IEE96], Fischer's mutual exclusion algorithm [Lam87], the (toy) rail road crossing example from [AHV93], the bounded retransmission protocol (when considering fixed values for the integer variables) and the biphase mark protocol (with minor adaptations). Moreover, the time constrained automata models of [MMT91, Lyn96] can be encoded straightforwardly into L/U automata.

We expect that quite a few other distributed algorithms and protocols can be modeled with L/U automata, since it is natural that the duration of an event (such as the communication delay in a channel, the computation time needed to produce a result, the time required to open the gate in a rail road crossing) lies between a lower bound and an upper bound. These bounds are often parameters of the system.

The next section and Section 6.5 show that the techniques discussed in this section to eliminate parameters in L/U models reduce the verification effort significantly and possibly lead to a completely non-parametric model. Finally, we remark that similar techniques can

be applied to lower bound and upper bound parameters, when they are present in a general PTA, which may thus also have parameters which are neither lower bound nor upper bound parameters and to which the techniques cannot be applied.

6.4.2 Verification of Fischer's mutual exclusion protocol

In this section, we apply the results from the previous section to verify the Fischer protocol described in Section 6.2.4. We establish the sufficiency of the protocol constraints by non-parametric model checking and the necessity of the constraints by eliminating three of the four parameters.

We also tried to use the prototype to verify the protocol model without any substitutions or changing of bounds, but this did not terminate within 20 hours. Since we observed that the constraint lists of the states explored kept on growing, we suspected that this experiment would not terminate at all. (Recall that parametric verification is undecidable.) Verification of the reduced models took only 2 seconds.

Now, consider the Fischer protocol model from Section 6.2.4 again. In this section, we analyze a system $\mathcal{A}$ consisting of two parallel processes P_1 and P_2. It is clear that $\mathcal{A}$ is a fully parametric L/U automaton: min_rw and min_delay are lower bound parameters and max_rw and max_delay upper bound parameters.

The mutual exclusion property is expressed by the formula $\Phi_{ME} \equiv \forall\Box\neg(P_1.cs \wedge P_2.cs)$. Recall that, when assuming the basic constraints $B_{ME} \equiv 0 \leq min_rw < max_rw \ \wedge\ 0 \leq min_delay < max_delay$, mutual exclusion is guaranteed for the parameter values satisfying $C_{ME} \equiv max_rw \leq min_delay$. Thus, we will prove that $v \models B_{ME} \implies (\llbracket\mathcal{A}\rrbracket_v \models \Phi_{ME} \iff v \models C_{ME})$, for all valuations v.

Sufficiency of the Constraints

We show that the constraints assure mutual exclusion, that is

$$\text{if } v \models C_{ME} \wedge B_{ME}, \text{ then } \mathcal{A}[v] \models \Phi_{ME}.$$

We perform the substitution

$$min_rw \mapsto 0, max_delay \mapsto \infty, min_delay \mapsto max_rw$$

to obtain a fully parametric automaton $\mathcal{A}'$ with one parameter, max_rw. We have established by non-parametric model checking that $\mathcal{A}'[0] \models \Phi_{ME}$ and $\mathcal{A}'[1] \models \Phi_{ME}$. Now Proposition 6.4.8 yields that $\llbracket\mathcal{A}'\rrbracket_v \models \Phi_{ME}$ for all valuations v (where only the value of max_rw matters). This means that $\llbracket\mathcal{A}\rrbracket_v \models \Phi_{ME}$ if $v(min_rw) = 0$, $v(max_rw) = v(min_delay)$ and $v(max_delay) = \infty$. Then Proposition 6.4.3(2) yields that the invariance property Φ_{ME} also holds if we increase the lower bound parameters min_rw and min_delay and if we decrease the upper bound parameter max_rw. More precisely, Proposition 6.4.3(2) implies that $\llbracket\mathcal{A}\rrbracket_v \models \Phi_{ME}$ for all v with $0 \leq v(min_rw), v(max_rw) \leq v(min_delay)$ and $v(max_delay) \leq \infty$. Then, in particular, $\llbracket\mathcal{A}\rrbracket_v \models \Phi_{ME}$ if $v \models C_{ME} \wedge B_{ME}$.

Necessity of the Constraints:

We show that

$$v \models B_{ME} \wedge \neg C_{ME} \implies \mathcal{A}[v] \models \neg\Phi_{ME},$$

i.e. that if $v \models min_rw < max_rw \wedge min_delay < max_delay \wedge min_delay < max_rw$, then $\mathcal{A}[v] \models \neg\Phi_{ME} \equiv \exists\Diamond(P_1.cs \wedge P_2.cs)$. We construct the automaton $\mathcal{A}^{\leq}$ and proceed in two steps.

Step 1 Let v_0 be the valuation $v_0(min_delay) = v_0(max_delay) = 0$ and $v_0(min_rw) = v_0(max_delay) = 1$. By non-parametric model checking we have established that

$$\mathcal{A}^{\leq}[0] \models \neg\Phi_{ME} \tag{6.8}$$

$$\mathcal{A}^{\leq}[v_0] \models \neg\Phi_{ME}. \tag{6.9}$$

We show that it follows that for all v

$$v \models 0 = min_delay = max_delay \leq min_rw = max_rw \implies \mathcal{A}^{\leq}[v] \models \neg\Phi_{ME}. \tag{6.10}$$

Assume $v \models 0 = min_delay = max_delay \leq min_rw = max_rw$. Consider $t = v(min_rw)$. If $v(min_rw) = 0$, then (6.8) shows that $[\![\mathcal{A}^{\leq}]\!]_v \models \neg\Phi_{ME}$. Therefore, assume $v(min_rw) > 0$ and consider $\frac{v}{t} \equiv \lambda x.\frac{v(x)}{t}$. It is not difficult to see that

$$\frac{v}{t} \models 0 = min_delay = max_delay \leq min_rw = max_rw = 1.$$

Therefore, (6.9) yields $[\![\mathcal{A}^{\leq}]\!]_{\frac{v}{t}} \models \neg\Phi_{ME}$. Since $\mathcal{A}^{\leq}$ is a fully parametric PTA, Proposition 6.4.8 yields that $[\![\mathcal{A}^{\leq}]\!]_v \models \neg\Phi_{ME}$.

Step 2 Let $\mathcal{A}'$ be the automaton that is constructed from $\mathcal{A}^{\leq}$ by performing the following substitution $min_delay \mapsto 1$, $max_delay \mapsto 1$, $min_rw \mapsto max_rw$. By parametric model checking we have established

$$v \models 1 \leq max_rw \implies [\![\mathcal{A}']\!]_v \models \neg\Phi_{ME}. \tag{6.11}$$

This means that if

$$v \models min_delay = max_delay = 1 \leq min_rw = max_rw \implies [\![\mathcal{A}^{\leq}]\!]_v \models \neg\Phi_{ME}.$$

By an argument similar to the one we used to prove (6.10), we can use Proposition 6.4.8 to show that

$$v \models min_delay = max_delay \leq min_rw = max_rw \implies [\![\mathcal{A}^{\leq}]\!]_v \models \neg\Phi_{ME},$$

where now the case $v(min_delay) = 0$ is covered by Equation (6.10) in Step 1. Then Proposition 6.4.3(1) yields that the reachability property $\neg\Phi_{ME}$ also holds if the values for the lower bounds are decreased and the values for the upper bounds are increased. Note that we may increase max_delay as much as we want; $v(max_delay)$ may be larger than $v(min_rw)$. Thus we have

$$\begin{aligned} v \models min_rw \leq max_rw \wedge min_delay \leq max_delay \wedge min_delay \leq max_rw \\ \implies [\![\mathcal{A}^{\leq}]\!]_v \models \neg\Phi_{ME} \end{aligned}$$

and then Proposition 6.4.12 yields that

$$\begin{aligned} v \models min_rw < max_rw \wedge min_delay < max_delay \wedge min_delay < max_rw \\ \implies [\![\mathcal{A}]\!]_v \models \neg\Phi_{ME}. \end{aligned}$$

model from	*initial constraints*	*property*	*Uppaal*	*time*	*memory*
[DKRT97a]	yes	safety1	param	1.3 m	34 Mb
[DKRT97a]	no	safety2	param	11 m	180 Mb
[DKRT97a]	yes	safety2	param	3.5 m	64 Mb

Figure 6.9: Experimental results for the bounded retransmission protocol

We have checked the result formulated in Equation (6.11) with our prototype implementation. The experiment was performed on a SPARC Ultra in 2 seconds CPU time and 7.7 Mb of memory.

The substitutions and techniques used in this verification to eliminate parameters are ad hoc. We believe, however, that more general strategies can be applied. Especially in this case, where the constraints are L/U–like (i.e. can be written in the form $e \prec 0$ such that every p occurring negatively in e is a lower bound parameter and every p occurring positively in e is an upper bound parameter), it should be possible to come up with smarter strategies for parameter elimination.

6.5 Experiments

6.5.1 A Prototype Extension of Uppaal

Based on the theory described in Section 6.3, we have built a prototype extension of Uppaal. In this section, we report on the results of experimenting with this tool.

Our prototype allows the user to give some initial constraints on the parameters. This is particularly useful when explorations cannot be finished due to lack of memory or time resources, or because a non-converging series of constraint sets is being generated. Often, partial results can be derived by observing the constraint sets that are generated during the exploration. Based on partial results, the actual solution constraints can be established in many cases. These partial results can then be checked by using an initial set of constraints.

6.5.2 The Bounded Retransmission Protocol

Description This protocol was designed by Philips for communication between remote controls and audio/video/TV equipment. It is a slight alteration of the well-known alternating bit protocol, to which timing requirements and a bound on the retry mechanism have been added. In [DKRT97a] constraints for the correctness of the protocol are derived by hand, and some instances are checked using Uppaal. Based on the models in [DKRT97a], an automatic parametric analysis is performed in [AAB00], however, no further results are given.

Parametric approach For our analysis, we use the timed automata models from the paper [DKRT97a]. These models typically consist of 7 communicating processes, varying from 2 locations with 4 transitions to 6 locations with 54 transitions, and has 5 clocks and 9 non-clock variables in total. In [DKRT97a] 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 contains a parameter which our prototype version of Uppaal is not able to handle yet.

One of the constraints derived in [DKRT97a] is that $\mathrm{TR} \geq 2 \cdot \mathrm{MAX} \cdot \mathrm{T}_1 + 3 \cdot \mathrm{TD}$, where TR is the timeout of the receiver, T_1 is the timeout of the sender, MAX is the number of resends made by the sender, and TD is the delay of the channel. This constraint is needed to ensure that the receiver does not time out prematurely before the sender has decided to abort transmission. The sender has a parameter SYNC which decides for how long the sender waits until it expects that the receiver has realized a send error and reacted to it. In our parametric analysis we used TR and SYNC as parameters and instantiated the others to fixed values. Using our prototype we did derive the expected constraint $\mathrm{TR} \geq 2 \cdot \mathrm{MAX} \cdot \mathrm{T}_1 + 3 \cdot \mathrm{TD}$. However, we also derived the additional constraint $\mathrm{TR} - 2 \leq \mathrm{SYNC}$ which was not stated in [DKRT97a] for this property. The necessity of this constraint was verified by trying models with different fixed values for the parameters. The full set of constraints derived in [DKRT97a] includes a constraint $\mathrm{TR} \geq \mathrm{SYNC}$ which is based on the property we cannot check. Therefore the error we have encountered is only present in an intermediate result, the complete set of constraints derived is correct. The authors of [DKRT97a] have acknowledged the error and provided an adjusted model of the protocol, for which the additional constraint is not necessary.

The last constraint derived in [DKRT97a] arises from checking that the sender and receiver are not sending messages too fast for the channel to handle. In this model we treat T_1 as a parameter and derive the constraint $\mathrm{T}_1 > 2 \cdot \mathrm{TD}$ which is the same as is derived in [DKRT97a].

6.5.3 Other Experiments

We have experimented with parametric versions of several models from the standard Uppaal distribution, namely Fischer's mutual exclusion protocol, a train gate controller, and a car gear box controller.

In the case of Fischer's protocol (which is the version of the standard Uppaal distribution, and not the one discussed in the rest of this chapter), we parameterized a model with two processes, by turning the bound on the period the processes wait, before entering the critical section, into a parameter. We were able to generate the constraints that ensure the mutual exclusion within 2 seconds of CPU time on a 266 MHz Pentium MMX. Using these constraints as initial constraints and checking that now indeed the mutual exclusion is guaranteed, is done even faster. Fischer's protocol with two processes was also checked in [AAB00], which took about 3 minutes.

6.5.4 Discussion

Our prototype handles parametric versions of bench-mark timed automata rather well. In some cases, the prototype will not generate a converging series of constraints, but in all cases we were able to get successful termination when applying (conjectures of) solution constraints as initial constraints in the exploration. The amount of time and memory used is then in many cases quite reasonable.

From our results it is not easy to draw clear-cut conclusions about the type of parametric model, for which our prototype can successfully generate constraints. It seems obvious from the case studies that the more complicated the model, the larger the effort in memory and time consumption. So it is worthwhile to have small, simple models. However, the danger of non-termination is most present in models which have a lot of behavioral freedom. The most promising direction, therefore, will be to experiment with conjectured solution constraints,

and to combine this with the techniques for L/U automata.

6.6 Conclusions

This chapter reports on a parametric extension to the model checker Uppaal. This tool is capable of generating parameter constraints that are necessary and sufficient for a reachability or invariant property to hold for a linear parametric timed automaton. The semantics of the algorithms underlying the tool is given in clean SOS–style rules. Although the work [AHV93] shows that parameter synthesis is undecidable in general, our prototype implementation terminates on many practical verification questions and the run time of the tool is acceptable. Significant reductions are obtained by parameter elimination in L/U automata.

There are several relevant and interesting topics for future research. First of all, serious improvements in the applicability of the tool can be obtained by improving the user interface. Currently, the tool generates many parameter equations whose disjunction is the desired constraint. Since the number of equations are can be quite large, it would be more convenient if the tool could simplify these set of equations. This could for instance be done with reduction techniques for BDDs.

Another relevant issue for parameter analysis is the theoretical investigation of the class of L/U automata. It would for instance be interesting to get more insight which types of problems are decidable for L/U automata and which are not. Furthermore, it would be interesting to investigate the use of L/U automata for synthesizing the constraints, rather than for analyzing given constraints as we did in this chapter. On the practical side, the reduction techniques for L/U automata could be implemented.