Integer Parameter Synthesis for Parametric Timed Automata

Arnab Sur

Advisor:
Prof. B. Srivathsan, CMI

A thesis submitted in partial fulfilment
of the requirements for the degree
Master of Science
in
Computer Science

Chennai Mathematical Institute
June 2022

Abstract

Parametric Timed Automata (PTA) were introduced by Alur et al. as a formalism to model real-time systems with certain timing parameters. The Parameter Synthesis Problem for Reachability asks for all the values of the parameters for which there is an accepting run in the corresponding timed automaton (TA). However, Alur et al. show that it is, in general, undecidable whether there exists a valuation for the parameter of a PTA for which there is an accepting run in the corresponding timed automaton. We, therefore, focus on a restricted version of the problem where the parameter values belong to a bounded set of integers. Deciding if there is such a parameter valuation is obviously decidable, one just iterates over all parameter valuations from the domain (which is finite) and checks reachability in the corresponding TA.

However that approach to solve the parameter synthesis problem is unsatisfactory since for many parameter valuations the corresponding TAs may exhibit similar behviours so we want to merge computations and compute symbolically the set of desired parameter valuations. We use an extension of zones into the parametric case introduced by Hune et al. and incorporate the LU abstraction for timed automata, which is finite and coarsest given LU bounds, in parametric zones. We provide a terminating algorithm which is both sound and correct. We further discuss a related problem of finding the set of zone graph equivalent parameter valuations given a reference parameter valuation. We discuss how a solution to that could be used for the parameter synthesis problem.

Acknowledgement

I would first like to thank my Advisor Prof. B. Srivathsan. He has had immense patience with me, and has provided valuable guidance and suggestions regarding both my thesis and my career, and my gratitude to him knows no bounds.

Next I would like to thank my classmates Sayantan and Vaishnavi, amongst others, for not only making my time at CMI memorable but also for all their time and moral support that they provided that helped me finish my thesis.

I would like to extend my gratitude to the teachers, mentors and professors I have encountered over the years who have inculcated in me an interest in mathematics and computer science, especially professors at CMI who opened my eyes to the vast field of automata theory and verification.

Finally I thank my parents, my first teachers, who went to great lengths to provide me with a comfortable education and have supported me along my academic journey. They showed me that dedication and perseverance pays off, and I hope to live up to the standards they have set for me, someday.

Contents

Chapter 1

Introduction

Real-time systems are everywhere today, be it pacemakers, satellite systems, air traffic control systems, or autonomous driving systems. Needless to say, such systems must be predictable, precise, and robust. They need to respond quickly and reliably to events. It is thus crucial to have a formal framework that verifies that such a system meets required specifications. Timed automata [AD94] provide such a formalism to model and verify real-time systems.

A timed automaton is a finite state automaton equipped with clocks, which are real valued variables. Each transition specifies a linear constraint, called a *guard*, on the values of the clocks that must be satisfied so that the transition is taken. For instance, $x > 3 \ \wedge \ y < 5$ is a guard which says the value of clock x must be greater than 3 and the value of clock y must be less than 8. Additionally a transition specifies a set of clocks that are to be *reset*, i.e., the values of those clocks are set to 0 once the transition is taken. Resets allows measuring time intervals beteen events.

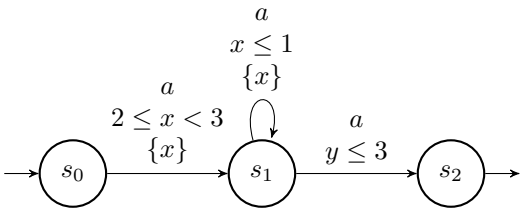

Figure 1.1: Example of a TA

One of the fundamental questions in verification is *reachability*, where one asks if some state of the system could be reached in any execution. It is useful to check if the system could end up in an undesirable state. The reachability problem of timed automata has been shown to be decidable in [AD94]. In the decades since, timed automata have been extensively studied and we have state of the art algorithms that decide reachability. These algorithms have been implemented in many tools such as Uppaal [Beh+06b], Kronos [Boz+98], TChecker [HP], etc.

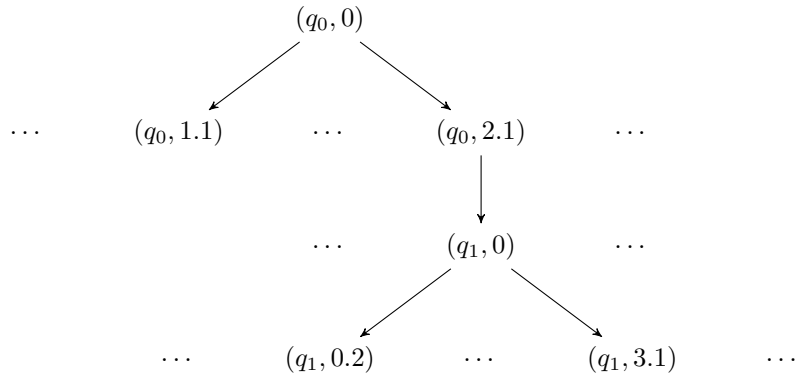

Figure 1.2: Semantics of a TA

Since the space of configurations of a TA is uncountably infinite, to decide reachability, the common approach is to group together valuations using *zones*. Zones are convex sets described by finite conjunctions of difference bounds and can be efficiently represented by Difference Bound Matrices (DBMs) [Dil89] which can be efficienty manipulated and hence algorithms using them can be efficiently implemented. A reachability algorithm starts from the initial state and computes the set of valuations (as zones) reachable at each step. The graph created this way is called the *zone graph*. The algorithm halts returning success when an accepting state is reached. An accepting state of the TA is reachable if and only if there is a node with an accepting state in the zone graph [DT98].

The zone graph, however, may be infinite and algorithm may not terminate. However, some zones may be *simulated*, where the behaviour of the TA from those zones can be "mimcked" from other zones. We can therefore stop the exploration there. One such *simulation relation* is the LU simulation, which yields a finite zone graph and gives one of the best approximations.

Timed automata, however, require concrete timing constraints such as "the value of a clock must be between 1 and 2 time units". In reality, systems are designed with respect to certain parameters of the environment. The system designer would want to determine for what values of said parameters would an undesirable state be reachable, preferably as symbolic constraints over the parameter values. Alur et al. in [AHV93] introduce Parametric Timed Automata (PTA) where the guards on the transitions compare clock values against the parameters, e.g., $x < p \wedge y > q$. Alur et al. in the same paper where they introduce PTA, however, show that the *parameter emptiness* problem, i.e., given a parametric timed automaton, does there exist values of the parameters for which some specific state in the corresponding timed automaton is reachable, is undecidable for PTA containing three clocks. They also show that the problem single clock PTA is decidable and decidability for PTA containing two clocks is open.

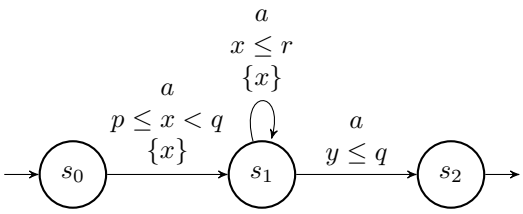

Figure 1.3: Example of a PTA

Many subclasses of PTA have since been studied but decidability results are scarce. [Hun+02] introduce L/U-PTAs where any parameter in a upper bound guard (i.e., guards of the form $x < p$) does not appear in a lower bound guard (i.e., guards of the form $x > p$) and vice-versa. However an exact solution for the synthesis problem, i.e., computing the set of all parameter values for which some state in the corresponding TA is reachable, is intractable (its result cannot be represented using a finite union of polyhedra) [JLR15]. It is known that exact synthesis can be achieved for the following subclasses:

1. U-PTAs (and L-PTAs) where parameters must always appear in an upper (resp. lower) bound guard over integer valued parameters [BL09].
2. All PTAs with bounded and integer-valued parameters (which reduces to reachability in TAs) [JLR15].
3. Reset-update-to-parameters-PTAs ("R-U2P-PTAs"), in which all clocks must be updated (possibly to a parameter) whenever a clock is compared to a parameter in a guard [RLA21].

In this thesis we focus on the PTAs where parameter values are bounded and integer-valued. Practically this isn't much restrictive since in classical timed automata, guards use integer constants (or rational constants which are dealt by scaling up the model to get integers) and in practice there usually is a (possibly huge) bound on the delays modelled by the parameters. In this case the parameter emptiness problem is trivially decidable. One just iterates over all parameter valuations and checks reachability in the corresponding timed automaton. However, this is an unsatisfactory solution for the parameter synthesis problem for two reasons. One, for many parameter valuations the corresponding TA may behave similarly and we would be making a lot of wasteful computations. Two, we want a succinct, symbolic representation of the set of all such parameter valuations, as a constraint over the parameters.

To decide reachability in PTA, a *parametric zone graph* is constructed which abstracts over zone graphs. It can be seen as a succinct representation of multiple zone graphs, one corresponding to each parameter valuation. It is a transition system over *parametric zones*, which represent multiple zones (one for each parameter valuation). Thus all parameter valuations that are in nodes that have an accepting state of the PTA are such that in their corresponding TAs, there is an accepting run. A parametric zone can be represented by constrained PDBMs [Hun+02] which are extensions of the DBMs to the parametric case. However, parametric zone graphs can still be infinite.

1.1 Related Work

Hune et al. in [Hun+02] provide semi-algorithms for the parameter synthesis problem for general PTA and introduce constrained PDBMs as a representation of parametric zones. They implement their algorithms in a prototype extension of Uppaal [Beh+06b].

Jovanovic et al. in [JLR15] provide a terminating algorithm for bounded integer parameters. They use a representation of parametric zones that are polyhedra over the space of both clock and parameter valuations and project the zones with accepting states to the space of parameter valuations. They implement their algorithm in their tool Romeo [Lim+09] which uses *Timed Petri Nets* as a model.

André et al. in [ALR15] and [AA22] introduce, for PTA with bounded parameters, a parametric extrapolation based on the maximum bounds on the parameters and provide algorithms that terminate and can output constraints arbitrarily close to the complete result. They implement their algorithms in their tool Imitator [And21].

André et al. in [AF10] provide a solution to the parameter synthesis problem for bounded integer parameters using algorithms in [And+08] which starts from a reference parameter valuation and derives constraints on parameters such that the behaviour of the PTA for those parameters remains time-abstract equivalent. However, their algorithms are not guaranteed to terminate. They implement their algorithms in their tool Imitator [And21].

1.2 Contribution of this Thesis

In this thesis, we provide a terminating algorithm for the bounded integer parameter synthesis problem that incorporates LU simulations (which gives one of the best known approximations for TA) into parameteric zone graphs, giving us a finite parameteric zone graph.

We then discuss zone graph equivalence, where we partition parameters based on the shape of their corresponding zone graphs. We use this characterization to solve the bounded integer parameter synthesis problem by picking up parameter valuations and finding zone graph equivalent parameters until the domain of parameter valuations is exhausted, similar to [AF10], but our algorithm terminates.

Organization of this Thesis

The first chapter introduces timed automata and an approach to solve the reachability problem for timed automata. The second chapter introduces parametric timed automata and the bounded integer parameter synthesis problem for parametric timed automata. The third chapter introduces parametric zones and

operations on parametric zones. The fourth chapter describes a way to incorporate the LU abstraction into the parametric setting thereby giving a fast and terminating algorithm for the bounded integer parameter synthesis problem. The fifth chapter introduces zone graph equivalence and an alternate algorithm for the bounded integer parameter synthesis problem. We conclude with the sizth chapter.

Chapter 2

Timed Automata

In this chapter we introduce Timed Automata [AD94] formally and describe one of the best known approaches to solve the reachability problem. We introduce zones which are used to group valuations together and obtain a zone graph computed by starting with the initial set of valuations and collecting the valuations reachable at each step [DT98]. Zones can be efficiently represented and manipulated as difference bound matrices (DBMs) [Dil89; BY03]. Zone graphs maybe infinite, so we then introduce simulations and abstractions from simulations for zones that provide a finite simulation graph. We finish by describing the standard algorithm for reachability.

2.1 Timed Automata

Let X be a finite set of *clocks*. We define a *guard* as a finite conjunction of *simple guards* which are expressions of the form $x \sim c$ where $x \in X$, $c \in \mathbb{Z}$ and $\sim \in \{<, \leq, =, >, \geq\}$.

A *clock valuation* w is a function $w : X \to \mathbb{R}_{\geq 0}$ which assigns to each clock a non-negative real value. We denote the set of all clock valuations by $\mathbb{R}^X_{\geq 0}$, and by $\mathbf{0}$ that valuation that associates 0 to every clock in X.

For δ in $\mathbb{R}_{\geq 0}$, let $w + \delta$ be the clock valuation that assigns to each clock x the value $w(x) + \delta$.

For $R \subseteq X$ let $[R]w$ be the valuation that assigns 0 to each clock $x \in R$ and $w(x)$ otherwise.

A clock valuation w satisfies a guard g, denoted by $w \models g$, if g evaluates to true when every clock x in g is replaced by $w(x)$. We define the semantics of g, $[\![g]\!] = \{w \mid w \models g\}$.

Definition 2.1.1 (Timed Automaton (TA) [AD94])**.** A *TA* is a tuple $\mathcal{A} = (\Sigma, Q, q_0, X, T, Acc)$ where

- Σ is a finite set of *actions*,
- Q is a finite set of states
- $q_0 \in Q$ is the *initial state*,
- X is a finite set of clocks,

- $Acc \subseteq Q$ is a set of accepting states, and
- T is a finite set of transitions (q, g, R, q') where $q, q' \in Q$, g is a guard, and R is the set of clocks that are *reset* on the transition.

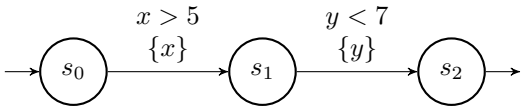

Figure 2.1: Example of a TA with 2 clocks

Definition 2.1.2 (Semantics of a TA)**.** Given a TA $\mathcal{A}$ the *semantics of* $\mathcal{A}$ *(*$S_{\mathcal{A}}$*)* is a transition system $(S, s_0, \rightarrow)$ where

- $S = Q \times \mathbb{R}^X_{\geq 0}$ is the set of states,
- $s_0 = (q_0, \mathbf{0})$ is the initial state, and
- $\rightarrow$ consists of the following kinds of transitions:

  Delay: $(q, w) \rightarrow^{\delta} (q, w + \delta)$ for some $\delta \in \mathbb{R}_{\geq 0}$.

  Action: $(q, w) \rightarrow^{t} (q', w')$ for some transition $t = (q, g, R, q') \in T$ such that $w \models g$ and $w' = [R]w$.

A *run* of $\mathcal{A}$ is a finite sequence of transitions in $S_{\mathcal{A}}$ starting from the initial state s_0. A run is *accepting* if the sequence ends in a state (q, w) where $q \in Acc$.

A *trace* (or a time abstract run) of $\mathcal{A}$ is a sequence of alternating states of $\mathcal{A}$ and discrete transitions in $\mathcal{A}$

$$q_1 \rightarrow^{t_1} q_2 \rightarrow^{t_2} \dots q_m$$

such that there is a run in $S_{\mathcal{A}}$ of the form

$$(q_1, w_1) \rightarrow^{\delta_1} \rightarrow^{t_1} (q_2, w_2) \rightarrow^{\delta_2} \rightarrow^{t_2} \dots (q_m, w_m)$$

Example 2.1.1. The following is an accepting run of the TA in Figure 2.1.

$$(s_0, \langle 0, 0\rangle) \rightarrow^{\delta_1} (s_0, \langle 5.1, 5.1\rangle) \rightarrow^{t_1} (s_1, \langle 0, 5.1\rangle) \rightarrow^{\delta_2} (s_1, \langle 1, 6.1\rangle) \rightarrow^{t_2} (s_2, \langle 1, 1\rangle)$$

The corresponding trace of the run is

$$s_0 \rightarrow^{t_1} s_1 \rightarrow^{t_2} s_2$$

where t_1 represents the transition from s_0 to s_1 and t_2 represents the transition from s_1 to s_2.

Definition 2.1.3 (Reachability Problem)**.** The reachability problem for timed automata is to decide whether a given TA has an accepting run.

2.2 Zones

Since $S_{\mathcal{A}}$ has uncountably infinite transitions, it cannot be used to solve the reachability problem. One solution is to find an approximation of this system by grouping together all valuations reaching a state of the TA via a particular path. Thus we work with configurations of the form (q, W) where q is a state of the TA and W is a set of clock valuations. We define a transition relation $\Rightarrow$ over symbolic states (q, W).

Definition 2.2.1 (Symbolic transition $\Rightarrow$)**.** Let $\mathcal{A}$ be a TA. For every transition t of $\mathcal{A}$ and every set of valuations W, we have a transition $\Rightarrow^t$ defined as

$$(q, W) \Rightarrow^t (q', W') \text{ where } W' = \{w' \mid \exists w \in W,\ \exists \delta \in \mathbb{R}_{\geq 0}.\ (q, w) \rightarrow^t \rightarrow^\delta (q', w')\}$$

The transition relation $\Rightarrow$ is the union of all $\Rightarrow^t$.

It has additionally been noticed that the sets W obtained in the nodes (q, W) can be described by some simple constraints involving only the difference between clocks [BY03]. This has motivated the definition of *zones*, which are sets of valuations defined by difference constraints.

Definition 2.2.2 (Zones [BY03])**.** A zone is a set of valuations defined by a conjunction of two kinds of clock constraints: for $x, y \in X$, $x \sim c$ and $x - y \sim c$ where $\sim \in \{<, \leq, =, >, \geq\}$ and $c \in \mathbb{Z}$.

It can be shown that starting from a node (q, W) with W being a zone, the transition $(q, W) \Rightarrow (q', W')$ leads to a node in which W' is again a zone. Observe that the initial set of valuations $W_0 = \{0 + \delta \mid \delta \in \mathbb{R}_{\geq 0}\}$ is indeed a zone. It is given by the constraints

$$\bigwedge_{x,y \in X} (x \geq 0 \wedge x - y = 0)$$

2.2.1 Difference Bound Matrices (DBMs)

Zones are efficiently represented by Difference Bound Matrices (DBMs).

Definition 2.2.3 (DBM)**.** Let X be a set of clocks and x_0 be a special variable added to X which represents the constant 0. A *DBM* D is a matrix $(D_{xy})_{x,y \in X}$ in which each entry $D_{xy} = (\prec_{xy}, c_{xy})$ represents the constraint $x - y \prec_{xy} c_{xy}$ where $c_{xy} \in \mathbb{Z} \cup \{\infty\}$ and $\prec_{xy} \in \{<, \leq\}$.

Every zone can be represented by a *canonical* DBM in which reducing the the weight $(\prec_{xy}, c_{xy})$ for any $x, y \in X$ would change the solution set. Given a DBM with n rows and columns, the canonical DBM representing the same solution set can be calculated in time $\mathcal{O}(n^3)$ using Floyd-Warshall's algorithm for shortest paths.

In what follows we use the notation Z for zones and use it to represent the sets of valuations in the nodes of the symbolic transition system.

Definition 2.2.4 (Zone Graph)**.** Given a timed automaton $\mathcal{A} = (Q, q_0, X, T, Acc)$, the *zone graph* $ZG(\mathcal{A})$ of $\mathcal{A}$ is a transition system whose nodes are of the form (q, Z) with $q \in Q$ and Z a zone. The initial node is (q_0, Z_0) where $Z_0 = \{0 + \delta \mid \delta \in \mathbb{R}_{\geq 0}\}$ is the set of valuations obtained by elapsing time from 0. The transitions are given by the relation $\Rightarrow$ of Definition 2.2.1.

2.2.2 Successor Computation

The successor computation $(q, Z) \Rightarrow^t (q', Z')$ for a transition $t = (q, g, R, q')$ proceeds as $(q, Z) \xrightarrow{g} (q, Z \wedge g) \xrightarrow{R} (q, [R](Z \wedge g)) \xrightarrow{elapse} (q, Z')$. All these operations can be computed efficiently using DBMs. At each step, the canonical DBM representing the zone obtained is computed. The algorithms for computing these operations with DBMs can be found in [BY03].

We will now define a symbolic semantics of timed automata which is a transition system with nodes consisting of zones. This is called the *zone graph* of the automaton.

2.3 Simulation

The zone graph could be infinite. So we find finite approximations by further grouping valuations. We define an *abstraction operator* to denote it.

Definition 2.3.1 (Abstraction Operator [Beh+03])**.** Let Z be a zone. An *abstraction operator* is a function $\mathfrak{a} : \mathcal{P}(\mathbb{R}_{\geq 0}) \to \mathcal{P}(\mathbb{R}_{\geq 0})$ such that $Z \subseteq \mathfrak{a}(Z)$ and $\mathfrak{a}(\mathfrak{a}(Z)) = Z$.

If $\mathfrak{a}$ has a finite range then this abstraction is *finite*.

One way to obtain abstractions is to group together valuations that are not distinguishable by an automaton, i.e. consider a bisimulation relation. If we are after reachability properties it has been noted in [Beh+06a] that one can even consider (time abstract) simulation relation [TaŞ+96]. For the rest of the section, assume a given automaton $\mathcal{A}$.

Definition 2.3.2 (Time-abstract simulation)**.** A *(state based) time-abstract simulation* between two states of transition system $\mathcal{S}_\mathcal{A}$ is a relation $(q, w) \preceq_{t.a.} (q', w')$ such that:

1. $q = q'$
2. if $(q, w) \to^\delta (q, w + \delta) \to^t (q_1, w_1)$, then there exists a $\delta' \in \mathbb{R}_{\geq 0}$ such that $(q, w') \to^{\delta'} (q, w' + \delta') \to^t (q_1, w_1')$ satisfying $(q_1, w_1) \preceq_{t.a.} (q_1, w_1')$ for the same transition t.

For two valuations w, w', we say that $w \preceq_{t.a.} w'$ if for every state q of the automaton, we have $(q, w) \preceq_{t.a.} (q', w')$. An abstraction $\mathfrak{a}_{\preceq_{t.a.}}$ based on a simulation $\preceq_{t.a.}$ can be defined as follows:

Definition 2.3.3 (Abstraction based on simulation)**.** Given a zone Z, we define $\mathfrak{a}_{t.a.}(Z) = \{w \mid \exists w' \in Z.\ w \preceq_{t.a.} w'\}$.

Computing the coarsest simulation relation is EXPTIME-hard [LS00]. We can get simulation relations that are computationally easier if we consider only a part of the structure of the automaton. The common way is to look at constants appearing in the guards of the automaton and consider them as parameters for abstraction.

2.3.1 LU Simulation

Definition 2.3.4 (LU-bounds)**.** The L bound for an automaton $\mathcal{A}$ is the function assigning to every clock x a maximal constant that appears in a lower bound guard for x in $\mathcal{A}$, that is, maximum over guards of the form $x > c$ or $x \geq c$. Similarly U is the function assigning to every clock x a maximal constant appearing in an upper bound guard for x in $\mathcal{A}$, that is, maximum over guards of the form $x < c$ or $x \leq c$.

We write U_x or L_x to mean $U(x)$ or $L(x)$ where $x \in X$.

The paper introducing LU-bounds [Beh+06a] also introduced the abstraction operator $\mathfrak{a}_{\preccurlyeq_{LU}}$.

We first recall the definition of an LU-preorder.

Definition 2.3.5 (LU-preorder [Beh+06a])**.** Let $L, U : X \to \mathbb{N} \cup -\infty$ be two bound functions. For a pair of valuations we set $w \preccurlyeq_{LU} w'$ if for every clock x:

- if $w'(x) < w(x)$ then $w'(x) > L_x$, and
- if $w'(x) > w(x)$ then $w(x) > U_x$.

It has been shown in [Beh+06a] that $\preccurlyeq_{LU}$ is a simulation relation. The $\mathfrak{a}_{\preccurlyeq_{LU}}$ abstraction is based on this LU-preorder $\preccurlyeq_{LU}$.

Definition 2.3.6 ($\mathfrak{a}_{\preccurlyeq_{LU}}$-abstraction [Beh+06a])**.** Given L and U bound functions, for a zone Z we define:

$$\mathfrak{a}_{\preccurlyeq_{LU}}(Z) = \{w \mid \exists w' \in Z.\ w \preccurlyeq_{LU} w'\}$$

2.3.2 LU Zone Graph

The $\mathfrak{a}_{\preccurlyeq_{LU}}$ abstraction defines an abstract semantics. We call this abstract semantics the LU zone graph of the automaton.

Definition 2.3.7 (LU Zone Graph)**.** Let $\mathcal{A}$ be an automaton. The *LU Zone Graph* $ZG^{LU}(\mathcal{A})$ is the transition system (Σ, σ_0, τ) where

- $\Sigma = \{(q, Z) \mid q \in Q \text{ and } Z \text{ is a zone}\}$
- $\sigma_0 = (q_0, Z_0)$ where $Z_0 = \bigwedge_{x,y \in X} (x \geq 0 \wedge x - y = 0)$ is the initial state
- τ is the set of transitions and is of two types.

  Simulation $(q, Z) \rightsquigarrow_{\mathfrak{a}_{\preccurlyeq_{LU}}} (q, Z')$ when $Z \subseteq \mathfrak{a}_{\preccurlyeq_{LU}}(Z')$.

  Successor $(q, Z) \Rightarrow_{\mathfrak{a}_{\preccurlyeq_{LU}}} (q', Z')$ when $(q, Z) \Rightarrow (q', Z')$ is a symbolic transition (Definition 2.2.1) and there is no $(q, Z') \in \Sigma$ such that $Z \subseteq \mathfrak{a}_{\preccurlyeq_{LU}}(Z')$.

The LU zone graph $ZG^{LU}(\mathcal{A})$ satisfies the following two properties for reachability.

Soundness: If there is a path from (q_0, Z_0) to a state (q, Z) in $ZG^{LU}(\mathcal{A})$ then there is $w \in Z$ such that there is a path from $(q_0, 0)$ to a state (q, w) in $\mathcal{S}_\mathcal{A}$.

Completeness: If there is a path from $(q_0, 0)$ to (q, w) in $\mathcal{S}_\mathcal{A}$, then there is Z such that there is a path from (q_0, Z_0) to a state (q, Z) in $ZG^{LU}(\mathcal{A})$.

The $\mathfrak{a}_{\preccurlyeq_{LU}}$ is the coarsest abstraction that is sound and complete for reachability [HSW16] given only LU-bounds.

2.4 Standard Algorithm for Reachability

A forward exploration algorithm for solving the reachability problem constructs the reachability tree starting from the initial node (q_0, Z_0) with $Z_0 = \{0+\delta \mid \delta \in \mathbb{R}_{\geq 0}\}$. The successor with respect to $\Rightarrow$ is computed. By definition $\Rightarrow$ computes a time-elapsed zone. Therefore, all nodes that are explored by the algorithm have time-elapsed zones.

Before continuing exploration from a node (q, Z), the algorithm first checks if q is accepting. If not, the algorithm checks if for some visited node (q, Z'), we have $Z \subseteq \mathfrak{a}_{\preccurlyeq_{LU}}(Z')$. If this is the case, (q, Z) need not be explored. Otherwise, the successors of (q, Z) are computed as stated above. This way we ensure termination of the algorithm since $\mathfrak{a}_{\preccurlyeq_{LU}}$ is a finite abstraction [Beh+06a].

We specify the algorithm below that also builds the LU zone graph for $\mathcal{A}$.

Algorithm 1 Standard Reachability Algorithm for Timed Automata

```
1:  procedure REACH(A)
2:      Waiting ← {(q0, Z0)}
3:      Passed ← ∅
4:      ZG ← ∅
5:      while Waiting ≠ ∅ do
6:          Remove (q, Z) from Waiting
7:          Passed ← Passed ∪ {(q, Z)}
8:          if q ∈ F then
9:              return ⊤
10:         if Z ⊆ a_{≼LU}(Z') for some (q, Z') ∈ Passed then
11:             ZG ← ZG ∪ {(q, Z) ⇝ (q, Z')}
12:         else
13:             for (q, Z) ⇒^t (q', Z') where t = (q, g, R, q') ∈ T do
14:                 Waiting ← Waiting ∪ {(q', Z')}
15:                 ZG ← ZG ∪ {(q, Z) ⇒ (q', Z')}
16:     return ⊥
```

2.4.1 Efficient inclusion test $Z \not\subseteq \mathfrak{a}_{\preccurlyeq LU}(Z')$

Since during the course of the algorithm a lot of inclusion tests ($\mathfrak{a}_{\preccurlyeq_{LU}}$) are performed, an efficient algorithm for the same is provided by [HSW16].

Theorem 2.4.1. Let Z, Z' be non-empty zones. Then, $Z \not\subseteq \mathfrak{a}_{\preccurlyeq LU}(Z')$ iff there exist two variables x, y such that:

$$Z_{0x} \geq (\leq -U_x) \text{ and } Z'_{yx} < Z_{yx} \text{ and } Z'_{yx} + (<, -L_y) < Z_{0x}$$

Remark 2.4.1. The indices of the matrix representations of the zones differ from the [HSW16] paper (in particular, the indices are flipped) since they use *distance graphs* representation of zones, while in this thesis we use the DBM representation.

Chapter 3

Parametric Timed Automata

In this chapter we introduce Parametric Timed Automata [AHV93], their syntax and semantics. We then define the bounded integer parameter synthesis problem.

3.1 Parametric Timed Automata

Let P be a finite set of *parameters* and X be a finite set of *clocks*. We define a *parametric guard* as a finite conjunction of *simple guards* which are expressions of the form $x \sim c$ where $x \in X$, $c \in \mathbb{Z} \cup P$, and $\sim \in \{<, \leq, =, >, \geq\}$.

A *parameter valuation* v is a function $v : P \to \mathbb{R}_{\geq 0}$ which assigns to each parameter in P a non-negative real. Similarly, a *clock valuation* w is a function $w : X \to \mathbb{R}_{\geq 0}$ which assigns to each clock a non-negative real value.

A pair of parameter and clock valuations (v, w) satisfies a parametric guard g, denoted by $(v, w) \models g$, if g evaluates to true when every parameter p and clock x in g is replaced by $v(p)$ and $w(x)$, respectively. We define the semantics of g, $[\![g]\!] = \{(v, w) \mid (v, w) \models g\}$.

Example 3.1.1. Let $P = \{p, q\}$ be the set of parameters and $X = \{x, y\}$ be the set of clocks.

Let w_0 be the clock valuation where $w_0(x) = 0;\ w_0(y) = 0$ then $w_1 = w + 1$ is the clock valuation where $w_1(x) = 1;\ w_1(y) = 1$.

Let $R = \{x\}$ be the set of clocks to be reset, and w_2 be a clock valuation where $w_2(x) = 1;\ w_2(y) = 1$ then $w_3 = [R]w_2$ is the clock valuation where $w_3(x) = 0;\ w_3(y) = 1$.

Let v be the parameter valuation where $v(p) = 2$ and $v(q) = 4$ and w_4 the clock valuation where $w_4(x) = 1; w_4(y) = 2$, g_1 be the parametric guard $x < p$, and g_2 be the parametric guard $y > q$, it can be seen that $(v, w_4) \in [\![g_1]\!]$ and $(v, w_4) \notin [\![g_2]\!]$.

Definition 3.1.1 (Parametric Timed Automaton (PTA))**.** A *PTA* is a tuple $\mathcal{A} = (\Sigma, Q, q_0, X, P, T, Acc)$ where

- Σ is a finite set of *actions*,

- Q is a finite set of states
- $q_0 \in Q$ is the *initial state*,
- X is a finite set of clocks,
- P is a finite set of parameters,
- $Acc \subseteq Q$ is a set of accepting states, and
- T is a finite set of transitions (q, g, R, q') where $q, q' \in Q$, g is a parametric guard, and R is the set of clocks that are *reset* on the transition.

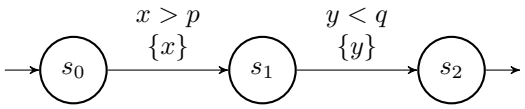

Figure 3.1: Example of a PTA with 2 clocks and 2 parameters

Given a parameter valuation v, we denote by $\mathcal{A}[v]$ the TA obtained by substituting each parameter p appearing in $\mathcal{A}$ by $v(p)$.

Definition 3.1.2 (Semantics of a PTA)**.** Given a PTA $\mathcal{A}$ and a parameter valuation v, the *semantics of* $\mathcal{A}$ *under* v *(*$S^v_{\mathcal{A}}$*)* is the semantics of the TA $\mathcal{A}[v]$, i.e. $S_{\mathcal{A}[v]}$.

3.2 Bounded Integer Parameter Synthesis Problem

Alur et al. in [AHV93] show that the parametric emptiness problem, i.e. whether there exists a valuation of the parameters such that the automaton has an accepting run, is undecidable. However, since in classical timed automata all clocks are usually compared against integer constants, it is useful to study PTA that use parameters from a bounded set of integers. Therefore, in this thesis we focus on the bounded integer parameter synthesis problem:

Definition 3.2.1 (Bounded Integer Parameter Synthesis Problem for Reachabaility)**.** Given a PTA $\mathcal{A}$ and $M \in \mathbb{N}$, compute all parameter valuations $v \in [M]^P$ such that $v(\mathcal{A})$ has an accepting run in $\mathcal{A}[v]$.

Example 3.2.1. In the example PTA in Figure 3.1, for all parameter valuations v satisfying $0 \leq p < q$ there is an accepting run in $\mathcal{A}[v]$.

A naïve algorithm that solves the bounded integer parameter synthesis problem runs the standard algorithm for reachability (Algorithm 1) on the TA $\mathcal{A}[v]$ for all $v \in [M]^P$. However our goal is to perform this search better and merge some computations. Hence we study parametric zones.

Chapter 4

Parametric Zones

In this chapter we introduce parametric zones and their representation as constrained PDBMs [Hun+02]. We describe the operations of guard intersection, canonicalization, clock reset, and time elapse on constrained PDBMs required to compute successors for the parametric zone graph.

4.1 Preliminaries

Given a finite set of parameters, $P = \{p_1, p_2, \ldots, p_n\}$, and $t_0, t_1, \ldots t_n \in \mathbb{Z}$, a *linear expression* is of the form $t_0 + p_1 t_1 + p_2 t_2 \cdots + p_n t_n$. Let E be the set of all linear expressions. Given $e \in E$ and a parameter valuation v, $e[v]$ is the expression obtained by replacing each parameter p in e by $v(p)$.

A *simple constraint* is an inequality of the form $e \prec e'$ where $e, e' \in E$ and $\prec \in \{<, \leq\}$. Given c, a simple constraint and v, a parameter valuation, $c[v]$ is the inequality obtaining by replacing e and e' with $e[v]$ and $e'[v]$ respectively. We say v satisfies c or $v \models c$ if $c[v]$ evaluates to true. We define $[\![c]\!] = \{v \mid v \models c\}$ as the semantics of the simple constraint c.

We extend the above to *constraints* which are boolean expressions of simple constraints. If C is a constraint, then $v \models C$, or $v \in [\![C]\!]$ if $C[v]$ evaluates to true. A simple constraint c covers a constraint C, denoted $C \models c$, if $[\![C]\!] \subseteq [\![c]\!]$. A constraint c *splits* a constraint C if $\exists v \in [\![C]\!].\, v \in [\![c]\!]$ and $\exists v \in [\![C]\!].\, v \notin [\![c]\!]$.

Example 4.1.1. Let $P = \{p, q\}$ be the set of parameters, $X = \{x, y\}$ be the set of clocks, and v be a parameter valuation where $v(p) = 2; v(q) = 4$.

Examples of linear expressions are $p + q + 1$, $2q$, and 5. Example of a simple constraint is $c = p + q + 1 < 2q$, and $c[v] = 7 < 8$ which is true and hence $v \in [\![c]\!]$.

Let $C = p < 5 \wedge q > 3$ be a constraint. $v \in [\![C]\!]$ since it satisfies $p < 5$ and $q > 3$.

The constraint $c_0 = q > 2$ covers C since every $v \in [\![C]\!]$ also satisifes c_0.

The constraint $c_1 = p > 3$ splits C since valuations $v \in [\![C]\!]$ such that $3 < v(p) < 5$ satisfy c_1 while valuations $v \in [\![C]\!]$ such that $v(p) < 3$ do not satisfy c_1.

The constraint $c_2 = p > 5$ is such that $C \models \neg c_2$ since no valuation $v \in [\![C]\!]$ can satisfy c_2.

4.2 Parametric Zones

Definition 4.2.1 (Parametric Zone)**.** A *parametric zone* is a pair (C, Z) where C is a constraint and Z is a conjunction of difference constraints of the form $x \sim e$ or $x - y \sim e$ where $x, y \in X$, $e \in E$, and $\sim \in \{<, \leq, =, >, \geq\}$.

Given a parameter valuation v and a clock valuation w, we say $(v, w) \models (C, Z)$ when $v \in [\![C]\!]$ and for each difference constraint $g = x - y \sim e \in Z$, $(v, w) \models g$.

Given a parameter valuation v, $Z[v]$ is the zone (Definition 2.2.2) obtained by replacing each difference constraint $x - y \sim e \in Z$ with $x - y \sim e[v]$.

Definition 4.2.2 (Canonical Parametric Zone)**.** A *canonical* parametric zone (C, Z) is such that for all $\sigma \in [\![C]\!]$, $Z[\sigma]$ is canonical.

4.2.1 Constrained Parametric Difference Bound Matrices

We represent parametric zones using constrained parametric difference bound matrices (PDBMs), as introduced in [Hun+02].

Definition 4.2.3 (Constrained PDBM)**.** A *PDBM* is a $(|X|+1 \times |X|+1)$ matrix whose each entry D_{ij} is a pair $(\prec_{ij}, e_{ij})$ where $\prec_{ij} \in \{<, \leq\}$ and $e_{ij} \in E \cup \{\infty\}$ which represents a difference constraint $x_i - x_j \prec_{ij} e_{ij}$ where $x_i, x_j \in X \cup \{x_0\}$. The clock x_0 is a reference clock whose value is 0 at all times, i.e. for all clock valuations w, $w(x_0) = 0$.

A *constrained PDBM* is a pair (C, D) where C is a constraint and D is a PDBM. We say a parameter and clock valuation $(v, w) \models (C, D)$ iff $v \models C$ and $w \in [\![D[v]]\!]$ where $D[v]$ is the DBM obtained by replacing each $e_{ij} \in D$ with $e_{ij}[v]$.

The semantics of a constrained PDBM is defined as $[\![(C, D)]\!] = \{(v, w) \mid (v, w) \models (C, D)\}$.

We will use each PDBM entry D_{ij} as a pair $(\prec_{ij}, e_{ij})$ and as the difference constraint $x_i - x_j \prec_{ij} e_{ij}$ interchangeably.

4.2.2 Arithmetic over Constrained PDBM Entries

We will now define an arithmetic over PDBM entries, $(\prec, e)$ where $e \in E$ and $\prec \in \{\leq, <\}$.

Addition $(\prec_1, e_1) + (\prec_2, e_2) = (\prec, e_1 + e_2)$ where $\prec = \leq$ iff $\prec_1 = \prec_2 = \leq$.

Equality Given a constraint C, $(\prec_1, e_1) = (\prec_2, e_2)$ if for all parameter valuations $v \in [\![C]\!]$, $e_1[v] = e_2[v]$ and $\prec_1 = \prec_2$.

Order When intersecting with guards and when canonicalizing constrained PDBMs we would need to determine whether one difference constraint contains another. So we say a difference constraint like $g_1 = x_i - x_j \prec_1 e_1$ is tighter than $g_2 = x_i - x_j \prec_2 e_2$ when for all $(v, w) \models g_1$, $(v, w) \models g_2$. However, since e_1 and e_2 are linear expressions, different parameter valuations could imply either one is tighter than the other. For example, $x_i - x_j < p$ is tighter than $x_i - x_j \leq q$ iff $p \leq q$. Hence given two PDBM entries $w_1 = (\prec_1, e_1)$ and $w_2 = (\prec_2, e_2)$, we formulate a tightness constraint of the form $e_1 \prec e_2$. Parameter valuations satisfying the constraint imply that w_1 is tighter than w_2, and valuations not satisfying it imply otherwise. We denote and define it as follows.

$$w_1 \leq w_2 = \begin{cases} e_1 < e_2 & \text{if } \prec_1 = \leq \text{ and } \prec_2 = < \\ e_1 \leq e_2 & \text{otherwise} \end{cases}$$

We would similarly require a constraint that implies a difference constraint *strictly* contains another. We define it as follows.

$$w_1 < w_2 = \begin{cases} e_1 \leq e_2 & \text{if } \prec_1 = < \text{ and } \prec_2 = \leq \\ e_1 < e_2 & \text{otherwise} \end{cases}$$

We can now conveniently look at PDBMs as a complete weighted directed graph with the vertex set as the set of clocks $X \cup \{x_0\}$ and the edges having weights as given by the PDBM entries, i.e., the edge $x_i \to x_j$ has weight D_{ij}. The arithmetic defined above allows us to define path weights as the sum of the weights of the edges that make up the path.

Given a constraint C we can determine an order on the weights. So we can say a path with weight $w_1 = (\prec_1, e_1)$ is shorter than one with weight $w_2 = (\prec_2, e_2)$ if $C \models w_1 \leq w_2$, i.e., $(\prec_1, e_1[v]) \leq (\prec_2, e_2[v])$ for all $v \in [\![C]\!]$. Indeed it may so happen that C splits $w_1 \leq w_2$ in which case we will want to study both cases separately. As we will see soon, guard intersection and canonicalization may split a parametric zone into several parametric zones.

4.3 Operations on Parametric Zones

We define the following operations on zones on constrained PDBMs: intersection with parametric guards, canonicalization, resetting clocks, and time elapse.

Guard Intersection When intersecting a simple guard $g = x_i - x_j \prec e$ (where exactly one of x_i and x_j is 0) to a constrained PDBM (C, D), we check if the constraint $D_{ij} = (x_i - x_j \prec_{ij} e_{ij})$ is tighter than g by checking whether the tightness constraint, $t = D_{ij} \leq g$, is covered by C. We therefore get 3 cases.

1. If $C \models t$, t holds under C which means D_{ij} is indeed tighter than g, and we leave D unchanged.
2. If $C \models \neg t$, t does not hold under C which implies g is tighter, and we set $D_{ij} = g$, denoted D_g.
3. If the tightness constraint splits C, i.e. C cannot determine if D_{ij} is tighter than g, we get two constrained PDBMs, one for the case in which it is tighter, and one for which it is not.

More precisely we define simple guard intersection as follows.

$$(C, D) \cap g = \begin{cases} \{(C \wedge t, D), (C \wedge \neg t, D_g)\} & \text{if } t \text{ splits } C \\ \{(C, D)\} & \text{if } C \models t \\ \{(C, D_g)\} & \text{otherwise} \end{cases}$$

Given (C, D), we define intersection with a guard $g = \bigwedge_{i=0}^{k} g_i$ (where each g_i is a simple guard), $(C, D) \cap g = A_k$ where A_k is computed as follows.

$$A_0 = (C, D) \cap g_0$$
$$A_i = \bigcup_{(C', D') \in A_{i-1}} (C', D') \cap g_i$$

Remark 4.3.1. If the number of conjuncts in g is k then the number of constrained PDBMs in $(C, D) \cap g$ is $\mathcal{O}(2^k)$.

Lemma 4.3.1 ([Hun+02])**.** $[\![(C, D)]\!] \cap [\![g]\!] = \bigcup \{[\![(C', D')]\!] \mid [\![(C', D')]\!] \in ((C, D) \cap g)\}$.

Remark 4.3.2. Note that in case of a split, for each parameter valuation $v \in [\![C]\!]$, v either satisfies t or $\neg t$, not both. Hence a split partitions $[\![(C, D)]\!] \cap [\![g]\!]$ into two sets each of which is the semantics of one of the resulting parametric zones.

Canonicalization The idea of a canonical constrained PDBM (C, D) carries over from the case of DBMs, i.e. (C, D) is *canonical* iff for all $k, i, j \in [0 \ldots |X|], C \models (D_{ij} \leq D_{ik} + D_{kj})$. Algorithm 2 describes canonicalization. We denote the operation as $\boldsymbol{\mathcal{C}}(C, D)$. The algorithm carries out the Floyd-Warshall all-pairs shortest path algorithm on the PDBM which, as discussed earlier, can be seen as a complete weighted directed graph. So for each $k \in [0 \ldots |X|]$ we compute the shortest paths from x_i to x_j via clocks in $\{x_0, \ldots x_k\}$. At the same time we check if there are negative cycles in the constrained PDBMs and remove them if there are since it implies that the corresponding parametric zone is empty.

Algorithm 2 Canonicalize a constrained PDBM

```
1:  procedure 𝒞(C, D)
2:      A = [(C, D)]                                  ▷ Represented as a list
3:      for k ← 0 to |X| − 1 do
4:          for i ← 0 to |X| − 1 do
5:              for j ← 0 to |X| − 1 do
6:                  for (C', D') ∈ A do
7:                      S = (C', D') ∩ D'_ik + D'_kj
8:                      for (C'', D'') ∈ S do
9:                          if C'' ⊨ (D''_ii ≤ (≤, 0)) for some i then
10:                             Remove (C', D') from S
11:                     Replace (C', D') with S in A
12:     return A
```

Lemma 4.3.2. Let $A = \boldsymbol{\mathcal{C}}(C, D)$.

1. For all $(C', D') \in A$, (C', D') is canonical.
2. $[\![(C, D)]\!] = \bigcup\{[\![(C', D')]\!] \mid (C', D') \in A\}$ and for all $(C_1, D_1), (C_2, D_2) \in A$, $[\![C_1]\!] \cap [\![C_2]\!] = \emptyset$.

Proof. Given a PDBM D, let $x_i \rightarrow^*_{\leq k} x_j$ denote a path from x_i to x_j involving clocks $\{x_l\}_{0 \leq l \leq k}$, and $x_i \rightarrow x_j$ denote an edge in D when D is seen as a graph. We prove the following invariants.

1. After iteration k of the **for** loop on line 3, for each $(C', D') \in A$ and for each pair of clocks $x_i, x_j \in X$, D'_{ij} holds the weight of the shortest $x_i \rightarrow^*_{\leq k} x_j$ path.
2. $[\![(C, D)]\!] = \bigcup\{[\![(C', D')]\!] \mid (C', D') \in A\}$ and for all $(C_1, D_1), (C_2, D_2) \in A$, $[\![C_1]\!] \cap [\![C_2]\!] = \emptyset$.

Before the iterations begin, the invariants trivially hold true.

1. For each (C', D'), at the beginning of iteration k since each D'_{ij} (i.e. $x_i \rightarrow x_j$) holds the weight of the shortest $x_i \rightarrow^*_{<k} x_j$ path already, and in particular $x_i \rightarrow x_k$ and $x_k \rightarrow x_j$ contain the shortest $x_i \rightarrow^*_{<k} x_k$ and $x_k \rightarrow^*_{<k} x_j$ paths respectively, we need only check if paths containing clock x_k give us a shorter path, i.e. if the path $x_i \rightarrow x_k \rightarrow^*_{<k} x_k \rightarrow x_j$ in (C', D') is shorter. If there are no negative cycles, the shortest $x_k \rightarrow^*_{<k} x_k$ path always has a weight $(\leq, 0)$, and so we check if $x_i \rightarrow x_k \rightarrow x_j$ (having weight $D'_{ik} + D'_{kj}$) is shorter than the path $x_i \rightarrow x_j$ (having weight D'_{ij}) in (C', D'). It is easy to see from the definition of guard intersection that line 7 does just that.

   Since the Floyd-Warshall algorithm assigns to D'_{ii} the shortest path from x_i to x_i, $C' \models (D'_{ii} \leq (\leq, 0))$ iff the constrained PDBM (C', D') has a negative cycle and there exists no $(v, w) \in [\![(C', D')]\!]$. At lines 8-10 the algorithm detects negative cycles in the resulting PDBMs and removes them if they do contain negative cycles. Note that if (C, D) itself had a negative cycle we would return an empty set.

   Together, the above two observations show that the first invariant is maintained.

2. For a constrained PDBM (C', D') and $(v, w) \in [\![(C', D')]\!]$, we know that for all k, i, j, $(v, w) \models D'_{ik}$ and $(v, w) \models D'_{kj}$ and therefore $(v, w) \models D'_{ik} + D'_{kj}$. This implies $[\![(C', D')]\!] = [\![(C', D')]\!] \cap [\![D'_{ik} + D'_{kj}]\!]$. It follows from Lemma 4.3.1 and Remark 4.3.2 that the second invariant is maintained at each iteration as well.

Thus at line 12 after the loops terminate, for all $(C', D') \in A$, for all k, i, j, $C' \models (D'_{ij} \leq D'_{ik} + D'_{kj})$, and $[\![(C, D)]\!] = \bigcup\{[\![(C', D')]\!] \mid (C', D') \in A\}$ and for all $(C_1, D_1), (C_2, D_2) \in A$, $[\![(C_1, D_1)]\!] \cap [\![(C_2, D_2)]\!] = \emptyset$.

□

Remark 4.3.3. Canonicalization yields $\mathcal{O}(2^{n^3})$ constrained PDBMs where $n = |X|$.

We extend the operation to a set of constrained PDBMs A as $\mathcal{C}(A) = \{\mathcal{C}(C, D) \mid (C, D) \in A\}$.

Resetting Clocks The reset operation for PDBMs is the same as for DBMs. For each clock $x_i \in R$, the set of clocks to be reset, we do for each $j \neq i, D_{ij} = D_{0j}$ and $D_{ji} = D_{j0}$. We denote the resulting constrained PDBM, $[R](C, D)$. We extend the notation to reset clocks in R in a set of constrained PDBMs, A. $[R]A = \{[R](C, D) \mid (C, D) \in A\}$.

Lemma 4.3.3 ([Hun+02])**.** If (C, D) is canonical then $[R](C, D)$ is canonical as well.

Lemma 4.3.4 ([Hun+02])**.** Let (C, D) be a canonical constrained PDBM. For all $v \in [\![C]\!]$ and clock valuations w, $w \models [R]D[v]$ iff $\exists w' \models D[v]$. $w = [R]w'$.

Time Elapse The time elapse operation for PDBMs is the same as for DBMs as well. For each clock $x_i \in X$, $i \neq 0$, we set $D_{i0} = (<, \infty)$. We denote the resulting constrained PDBM, $(C, \overrightarrow{D})$. We extend the notation to time elapse a set of constrained PDBMs A, $\overrightarrow{A} = \{(C, \overrightarrow{D}) \mid (C, D) \in A\}$.

Lemma 4.3.5 ([Hun+02])**.** Let (C, D) be a canonical constrained PDBM. For all $v \in [\![C]\!]$ and clock valuations w, $w \models \overrightarrow{D[v]}$ iff $\exists \delta \in \mathbb{R}_{\geq 0}\ \exists w' \models D[v]$. $w' + \delta$.

Lemma 4.3.6. Given a constrained PDBM (C, D) and a transition $t = (q, g, R, q')$ in a PTA $\mathcal{A}$, let $A = \overrightarrow{[R]\mathcal{C}((C, D) \cap g)}$.

1. For each $(C', D') \in A$ and for each $v \in [\![C']\!]$, $(q, D[v]) \Rightarrow (q', D'[v])$ is a symbolic transition (Definition 2.2.1) in the zone graph (Definition 2.2.4) of $\mathcal{A}[v]$.

2. For each $v \in [\![C]\!]$, if $(q, D[v]) \Rightarrow (q', Z)$ is a symbolic transition in the zone graph of $\mathcal{A}[v]$, then $\exists (C', D') \in A$. $v \in [\![C']\!]$ and $Z = D'[v]$.

Proof. The proof follows from the arguments in the proofs of Propositions 1 and 2 in [Hun+02]. □

Chapter 5

Parameter Synthesis for Reachability

In this chapter we present an algorithm that returns a constraint, the semantics of which is the set of all parameter values for which some final state of the PTA is reachable. We provide ways to compute LU Bounds for PTA. Then we symboically compute parametric zones that are not simulated. Finally, we define the parametric zone graph with simulation of a PTA given LU bounds.

5.1 LU Bounds

Without an ordering on the parameters we can't compute the LU bounds for each clock. Hence we guess LU bounds for each clock and build the necessary constraint for the parametric zone graph computation. First we define what a valid ordering on the parameters and constants in the PTA $\mathcal{A}$ is.

Definition 5.1.1 (Valid Ordering)**.** Let N be the set of integer constants appearing in the guards of the PTA $\mathcal{A}$. A valid ordering of the set $P \cup N$ is a sequence of its elements $\langle l_1, l_2, \ldots, l_n \rangle$ which defines a total order $l_i \leq l_{i+1}$ for each $i \in [n-1]$ and its projection on N agrees with the usual ordering on integers.

We now describe an algorithm which returns a set of LU functions and the corresponding constraints. The algorithm, for each valid ordering of $P \cup N$, infers the LU bounds for each clock and adds the constraints that imply they are larger than all the parameters and constants that appear in guards involving that clock. Note that not all valid orderings may result in unique LU bounds. We use valid orderings since it makes the description of the algorithm more succinct.

Recall that LU bounds are functions $L : X \to P \cup N$ and $U : X \to P \cup N$. We use $L_x \leftarrow l$ (similarly $U_x \leftarrow l$) to set the value of $L(x)$ (similarly $U(x)$) to some $l \in P \cup N$ for some $x \in X$. We denote by LU the pair of functions so constructed.

Algorithm 3 LU Bounds of $\mathcal{A}$

procedure $\mathsf{LU}(\mathcal{A})$
  $LUs \leftarrow \emptyset$
  for each valid ordering $ord = \langle l_1, l_2, \ldots, l_n \rangle$ of $P \cup N$ **do**
    $C \leftarrow \top$
    for $x \in X$ **do**
      $\mathcal{B}_U = \{l_i \mid x < l_i \text{ or } x \leq l_i \text{ appears in a guard}\}$
      $U_x \leftarrow \max_i \mathcal{B}_U$ ▷ According to *ord*
      $C \leftarrow C \wedge \bigwedge\{U_x \geq l \mid l \in \mathcal{B}_U\}$

      $\mathcal{B}_L = \{l_i \mid x > l_i \text{ or } x \geq l_i \text{ appears in a guard}\}$
      $L_x \leftarrow \max_i \mathcal{B}_L$ ▷ According to *ord*
      $C \leftarrow C \wedge \bigwedge\{L_x \geq l \mid l \in \mathcal{B}_L\}$
    $LUs \leftarrow LUs \cup \{(LU, C)\}$
  return LUs

We denote by $LU[v]$ the LU bounds obtained by substituting each U_x and L_x by $v(U_x)$ and $v(L_x)$ for all $x \in X$.

Lemma 5.1.1. Given $(LU, C) \in LUs$, for all $v \in [\![C]\!]$, $LU[v]$ is a valid LU bounds of the TA $\mathcal{A}[v]$.

Proof. For each clock x, $U_x = l$ (or $L_x = l$) iff $l = max_i\ \mathcal{B}_U$ (or $l = max_i\ \mathcal{B}_L$), i.e., l is the largest parameter or constant appearing in an upper (or lower) bound guard for x in $\mathcal{A}$ according to some valid ordering *ord*. Consequently we conjunct with C the constraints $l \geq l'$ for all $l' \in \mathcal{B}_U$ (or all $l' \in \mathcal{B}_L$). Thus a paramater valuation $v \models C$ iff $v(l) \geq v(l')$ for all $v(l') \in \mathcal{B}_U$ (or all $v(l') \in \mathcal{B}_L$) for each $x \in X$ implying $LU[v]$ is a valid LU bounds of $\mathcal{A}[v]$ (from Definition 2.3.4). □

We say a paramater valuation satisfies (LU, C) if $v \models C$.

Example 5.1.1. Assume the following guards in $\mathcal{A}$

$$x < p,\ x < q,$$
$$y < q,\ y < 5,$$
$$z < 4,\ z < p.$$

Table 5.1 lists the upper bound function (since there are no lower bound guards in this example) for each valid ordering.

5.2 Simulation

To ensure faster convergence and termination we want to quit exploring parametric zones which are simulated by other parametric zones. For that we use inclusion test in Theorem 2.4.1 for TA zones and try to extend it to the parametric case. Given a parametric zone (C, D), there may be different parameter valuations v and v' such that there exists parametric zones (C', D') and (C'', D'') where $v \in [\![C']\!]$ and $v' \in [\![C'']\!]$ and $D[v] \subseteq \mathfrak{a}_{\preccurlyeq_{LU[v]}}(D'[v])$ and

ord	U_x	U_y	U_z
$\langle p, q, 4, 5\rangle$	q	5	4
$\langle p, 4, q, 5\rangle$	q	5	4
$\langle p, 4, 5, q\rangle$	q	q	4
$\langle q, p, 4, 5\rangle$	p	5	4
$\langle q, 4, p, 5\rangle$	p	5	p
$\langle q, 4, 5, p\rangle$	p	5	p
$\langle 4, p, q, 5\rangle$	q	5	p
$\langle 4, p, 5, q\rangle$	q	q	p
$\langle 4, q, p, 5\rangle$	p	5	p
$\langle 4, q, 5, p\rangle$	p	5	p
$\langle 4, 5, p, q\rangle$	q	q	p
$\langle 4, 5, q, p\rangle$	p	q	p

Table 5.1: Upper bound function for each valid ordering

$D[v'] \not\subseteq \mathfrak{a}_{\preccurlyeq_{LU[v']}}(D''[v'])$. This implies we need to split the constraint into constraints for which the corresponding zone is simulated by another zone, and constraints for which the corresponding zone is not simulated. The algorithm below returns the constrained PDBM representing the set of zones not simulated.

Algorithm 4 Non Simulated Zones

1: **procedure** $\mathsf{NS}_{LU}((C, D), (C', D'))$
2: $\quad \tilde{C} \leftarrow \emptyset$
3: $\quad$ **for** $x, y \in X$ **do**
4: $\quad\quad \tilde{C} \leftarrow \tilde{C} \vee (((\leq -U_x) \leq D_{0x}) \wedge (D'_{yx} < D_{yx}) \wedge (D'_{yx} + (<, -L_y) < D_{0x}))$
5: $\quad \tilde{C} \leftarrow \tilde{C} \wedge C \wedge C'$
6: $\quad$ **return** $(\tilde{C} \vee (C \wedge \neg C'), D)$

Lemma 5.2.1. Given two constrained PDBMs, (C, D) and (C', D'), for all $v \in [\![C \wedge C']\!], v \in [\![\hat{C}]\!]$ iff $D[v] \not\subseteq \mathfrak{a}_{\preccurlyeq_{LU[v]}}(D'[v])$ where $(\hat{C}, D) = \mathsf{NS}_{LU}((C, D), (C', D'))$.

Proof. Let $v \in [\![C \wedge C']\!]$.

1. $v \in [\![\hat{C}]\!]$ implies for some $x, y \in X, (\leq v(-U_x)) \leq D_{0x}[v]$ and $D'_{yx}[v] < D_{yx}[v]$ and $(D'_{yx}[v] + (<, -v(L_y)) < D_{0x}[v])$. It follows from Theorem 2.4.1 that $D[v] \not\subseteq \mathfrak{a}_{\preccurlyeq_{LU[v]}}(D'[v])$.

2. $D[v] \not\subseteq \mathfrak{a}_{\preccurlyeq_{LU[v]}}(D'[v])$. By Theorem 2.4.1, for some $x, y \in X, (\leq v(-U_x)) \leq D_{0x}[v]$ and $D'_{yx}[v] < D_{yx}[v]$ and $(D'_{yx}[v] + (<, -v(L_y)) < D_{0x}[v])$. This implies $v \models ((\leq -U_x) \leq D_{0x})$ and $v \models (D'_{yx} < D_{yx})$ and $v \models (D'_{yx} + (< , -L_y) < D_{0x})$. It follows that $v \in [\![\hat{C}]\!]$.

□

It is easy to see the following lemma holds.

Lemma 5.2.2. Given a constrained PDBM (C, D) and a set of constrained PDBMs A, let $\hat{C} = \bigwedge\{C' \mid \exists (C'', D'') \in A, (C', D') = \mathsf{NS}_{LU}((C, D), (C'', D''))\}$. We claim $v \in [\![\hat{C}]\!]$ iff $D[v] \not\subseteq \mathfrak{a}_{\preccurlyeq_{LU[v]}}(D'[v])$ for all $(C', D') \in A$ such that $v \models C \wedge C'$.

5.3 Parametric Zone Graph

Now we introduce the notion of a parametric zone graph with simulation given parametric LU bounds.

Definition 5.3.1 (Parametric Zone Graph with Simulation ($PZG^{LU}(\mathcal{A})$))**.** The Parametric Zone Graph with Simulation of a PTA $\mathcal{A}$ given parametric LU bounds is a transition system (Σ, σ_0, τ) where

- $\Sigma = \{(q, (C, D)) \mid q \in Q$ and (C, D) is a constrained PDBM$\}$ is the set of states
- $\sigma_0 = (q_0, (C_0, D_0))$ where C_0 is the constraint representing the necessary and sufficient conditions on parameter valuations that specify the LU bounds and the bounds on the parameters, and D_0 is the PDBM corresponding to the zone $\bigwedge\limits_{x \in X} x \geq 0 \wedge \bigwedge\limits_{x,y \in X} x = y$ is the initial state
- τ is the set of transitions and is of two types. Given $(q, g, R, q') \in T$ and a state $(q, (C, D))$, we have the following transitions in τ.

  Simulation $(q, (C, D)) \rightsquigarrow_{\tilde{C}} (q', (C', D'))$ and there is a state $(q, (C', D')) \in \Sigma$ such that $[\![\tilde{C}]\!] \subseteq [\![C \wedge C']\!]$ and for all $v \in [\![\tilde{C}]\!]$, $D[v] \subseteq \mathfrak{a}_{\preccurlyeq_{LU[v]}}(D'[v])$.

  Successor $((q, (C, D))) \rightarrow (q', (C', D'))$ such that for all $v \models C'$, $D'[v] = \overrightarrow{[R]\mathcal{C}(D[v] \cap g)}$ and for all states $(q, (C'', D'')) \in ZG_{\mathcal{A}}$, for all $v \models C'$, $D[v] \not\subseteq \mathfrak{a}_{\preccurlyeq_{LU[v]}}(D''[v])$, or $v \not\models C''$.

  Let $C_{succ} = \bigvee\{C' \mid ((q, (C, D))) \rightarrow (q', (C', D')) \in PZG^{LU}(\mathcal{A})\}$, $C_{sim} = \bigvee\{\tilde{C} \mid (q, (C, D)) \rightsquigarrow_{\tilde{C}} (q', (C', D')) \in PZG^{LU}(\mathcal{A})\}$, and $C_{dis} = C \backslash (C_{succ} \vee C_{sim})$. $[\![C_{succ}]\!], [\![C_{sim}]\!]$, and $[\![C_{dis}]\!]$ form a partition of $[\![C]\!]$.

Given a parameter valuation v satisfying a parametric LU bounds LU, we denote by $PZG^{LU}(\mathcal{A})[v]$ the transition system with nodes $(q, D[v])$ such that $(q, C, D) \in \Sigma$ and $v \models C$, and $(q, D_0[v])$ is the start state. The transition system consists of the following transitions

1. $(q, D[v]) \Rightarrow (q', D'[v])$ if $((q, (C, D))) \rightarrow (q', (C', D'))$ is a successor edge in τ and $v \models C'$
2. $(q, D[v]) \rightsquigarrow (q, D'[v])$ if $(q, (C, D)) \rightsquigarrow_{\tilde{C}} (q', (C', D'))$ is a simulation edge in τ and $v \models \tilde{C}$.

Lemma 5.3.1. Given a PTA $\mathcal{A}$ and LU bounds (LU, C), for all parameter valuations v such that $v \models C$, $PZG^{LU}(\mathcal{A})[v] = ZG^{LU}(\mathcal{A}[v])$.

Proof. It is easy to see that the initial states of $PZG^{LU}(\mathcal{A})[v]$ and $ZG^{LU}(\mathcal{A}[v])$ are the same. We need only show that outgoing transitions from each node are the same too. There are two cases.

1. $(q, D[v]) \Rightarrow (q', D'[v])$ is a successor edge in $ZG^{LU}(\mathcal{A}[v])$ iff $(q, D[v]) \Rightarrow (q', D'[v])$ is a successor edge in $ZG^{LU}(\mathcal{A}[v])$ by the same arguments used in the proof of 4.3.6.

2. $(q, D[v]) \rightsquigarrow (q, D'[v])$ is a simulation edge in $PZG^{LU}(\mathcal{A})[v]$ iff $(q, D[v]) \rightsquigarrow (q, D'[v])$ is a simulation edge in $ZG^{LU}(\mathcal{A}[v])$ by definition of simulation edges in Definitions 2.3.7 and 5.3.1.

It follows that both graphs are the same. □

Lemma 5.3.2. Given a parameter valuation v such that v satisfies two distinct (LU, C) and (LU', C'), $PZG^{LU}(\mathcal{A})[v] = PZG^{LU'}(\mathcal{A})[v]$

Proof. Since they are distinct, LU and LU' differ on some bound for some clock, say, U_x for some $x \in X$ without loss of generality. Let $U_x = l$ and $U'_x = l'$ for some $l, l' \in P \cup N$ and $l \neq l'$. v satsifies both (LU, C) and (LU', C') iff $v(l) = v(l')$. This implies $LU(\mathcal{A}[v]) = LU(\mathcal{A}[v])$ which in turn implies $ZG^{LU}\mathcal{A}[v] = ZG^{LU'}\mathcal{A}[v]$. From Lemma 5.3.1 it follows that $PZG^{LU}(\mathcal{A})[v] = PZG^{LU'}(\mathcal{A})[v]$. □

5.4 The Algorithm

We are now ready to describe the algorithm. Algorithm 5 builds a parametric zone graph with simulation for each selection of LU bounds. It collects and returns the constraints at each node of the parametric zone graphs containing an accepting state of the PTA. In what follows we assume a function $M : P \to \mathbb{N}$ that assigns an upper bound to each parameter $p \in P$ and that the lower bound of each parameter is 0.

Lemma 5.4.1. For each LU bounds (LU, C), let ZG be the parametric zone graph constructed by the Algorithm. $ZG = PZG^{LU}(\mathcal{A})$.

Proof. It can be seen from Lemma 5.1.1 and the definition of $PZG^{LU}(\mathcal{A})$ (Definition 5.3.1) that the initial state (q, C, D_0) added to *Waiting* at line 4 is the initial state of $PZG^{LU}(\mathcal{A})$. It now suffices to prove that for each $(q, (C, D))$ in *Waiting*, we add all simulation and successor edges from it in $PZG^{LU}(\mathcal{A})$ to LU.

It follows from Lemma 5.2.1 that the simulation edge added at line 14 is a simulation edge in $PZG^{LU}(\mathcal{A})$, and Lemma 5.2.2 shows that all simulation edges from $(q, (C, D))$ in $PZG^{LU}(\mathcal{A})$ have been added to ZG after the **for** loop at lines 12-15 has terminated.

Lemma 4.3.6 shows that all successor edges from $(q, (C, D))$ in $PZG^{LU}(\mathcal{A})$ are added to ZG in the **for** loop at lines 17-19.

Thus, the algorithm computes $PZG^{LU}(\mathcal{A})$ for each LU bounds (LU, C). □

We now argue that the following properties hold.

Soundness For all parameter and clock valuations (v, w) such that there is a path in some $ZG \in ZGs$ reaching a state (q, C, D) with $(v, w) \models (C, D)$, there is a run in $S^v_{\mathcal{A}}$ reaching a state (q, w).

Algorithm 5 Parameter Synthesis using Simulation

procedure SYNTHESIS($\mathcal{A}$, M)
  $ZGs \leftarrow \emptyset$
  for each $(LU, C) \in \mathsf{LU}(\mathcal{A})$ **do**
    $C \leftarrow C \bigwedge_{p \in P} 0 \leq p \leq M_p$ ▷ All parameters are within bounds
    $Waiting \leftarrow \{(q_0, (C, D_0))\}$
    $Passed \leftarrow \emptyset$
    $ZG \leftarrow \emptyset$ ▷ The Zone Graph defined as a set of edges
    while $Waiting \neq \emptyset$ **do**
      Remove $(q, (C, D))$ from $Waiting$
      $Passed \leftarrow Passed \cup \{(q, (C, D))\}$
      for $(q, g, R, q') \in T$ **do**
        for $(q, (C', D')) \in Passed$ **do**
          $(\tilde{C}, D) \leftarrow \mathsf{NS}_{LU}((C, D), (C', D'))$
          $ZG \leftarrow ZG \cup \{(q, (C, D)) \rightsquigarrow_{C \setminus \tilde{C}} (q, (C', D'))\}$
          $(\hat{C}, D) \leftarrow (\hat{C} \wedge \tilde{C}, D)$
        $A \leftarrow [R]\boldsymbol{\mathcal{C}}((\hat{C}, D) \cap g)$
        for $(C', D') \in A$ **do**
          $Waiting \leftarrow Waiting \cup \{(q', (C', D'))\}$
          $ZG \leftarrow ZG \cup \{(q, (C, D)) \rightarrow (q', (C', D'))\}$
    $ZGs \leftarrow ZGs \cup \{ZG\}$

  $K \leftarrow \bot$
  for $ZG \in ZGs$ **do**
    for each node $(q, (C, D)) \in ZG$ such that $q \in F$ **do**
      $K \leftarrow K \vee C$
  return K

Completeness For all parameter and clock valuations (v, w) such that there is a run in $S^v_{\mathcal{A}}$ reaching a state (q, w), there is a path in some $ZG \in ZGs$ reaching a state (q, C, D) with $(v, w) \models (C, D)$.

Proof. From Lemma 5.3.2 we know that for each parameter valuation v, $PZG^{LU}(\mathcal{A})[v]$ is the same for all (LU, C) satisfied by v, and from Lemmas 5.4.1, 5.3.1, and the soundness and completeness of LU zone graphs (Definition 2.3.7) each $ZG \in ZGs$ is sound and complete for all parameter valuations v satisfying the corresponding LU bounds. □

Lemma 5.4.2. The algorithm SYNTHESIS terminates.

Proof. We claim that $ZG_{\mathcal{A}}$ is finite for each selection of the LU bounds, and the termination of the algorithm follows.

We assume on the contrary that $ZG_{\mathcal{A}}$ is infinite for one such selection. We first note that since the parameters are from a bounded set of integers, there are only a finite number of constraints representing a non-empty set of parameter valuations. Thus there must be an infinite path in $ZG_{\mathcal{A}}$ such that there is a parameter valuation v such that for each node $(q, (C, D))$ in the path, $v \models C$. By Lemma 5.3.1 the simulation graph for $\mathcal{A}[v]$ must be infinite, which cannot be since $\mathfrak{a}_{\preccurlyeq_{LU}}$ is a finite abstraction. □

Chapter 6

Zone Graph Equivalence

In this chapter we discuss zone graph equivalence, and provide an algorithm to compute the set of parameter valuations whose corresponding zone graphs have the same shape as a given reference valuation. Intuitively, the corresponding TAs for those parameter valuations behave similarly. We then use that to solve the bounded integer parameter synthesis problem.

We first define zone graph equivalence.

Definition 6.0.1 (Zone Graph Equivalence)**.** Two parameter valuations, γ and γ' are zone graph equivalent if the zone graphs $PZG^{LU}(\mathcal{A})[\gamma]$ and $PZG^{LU'}(\mathcal{A})[\gamma']$ on omitting the parametric zones (C, D) from each node $(q, (C, D))$ yield the same graph. We denote it as $\gamma \equiv_{zg} \gamma'$, and the equivalence class of γ by $[\gamma]$.

6.1 Guided Constrained PDBMs

We introduce the idea of guided constrained PDBMs where we include a guide parameter valuation γ along with a constrained PDBM. The result of the operations on the constrained PDBMs is determined by this guide valuation, i.e. in case of splits we choose the one that is satisfied by the guide valuation.

Definition 6.1.1 (Guided Constrained PDBM)**.** A *Guided Constrained PDBM* is a triple (γ, C, D) where γ is a parameter valuation, C is a constraint, D is a PDBM, and $\gamma \in C$.

6.1.1 Operations on Guided Constrained PDBMs

We now define the operations of guard intersection, canonicalization, resetting clocks, and time elapse on guided constrained PDBMs.

Guard Intersection $(\gamma, C, D) \cap g = (\gamma, C', D')$ where $(C', D') \in (C, D) \cap g$ and $\gamma \in C'$. Note that for each simple guard there is exactly one such (C', D') because in the case of a split, γ satisfies only one of t and $\neg t$, and consequently either $\gamma \models C' \wedge t$ or $\gamma \models C' \wedge \neg t$, where t is the tightness constraint.

Canonicalization We use the same algorithm for canonicalization as in the constrained PDBM case however we get at most one canonical guided constrained PDBM since there are no splits in the guard intersection operator. Note that we don't check for emptiness of the parametric zone yet. The reasons will be clear later.

Algorithm 6 Canonicalize a Guided constrained PDBM

```
1: procedure C(γ, C, D)
2:     A ← (γ, C, D)
3:     for k ← 0 to |X| − 1 do
4:         for i ← 0 to |X| − 1 do
5:             for j ← 0 to |X| − 1 do
6:                 (γ, C′, D′) ← (γ, C, D) ∩ D_ik + D_kj
7:     return A
```

The algorithms for clock reset and time elapse operators remain the same.

It is easy to see from the definitions of the operations and Lemma 4.3.6 that the following lemma holds.

Lemma 6.1.1. Given (γ, C, D), let $(\gamma, C', D') = \overrightarrow{[R]\mathcal{C}((\gamma, C, D) \cap g)}$.

1. For each $v \in [\![C']\!]$, $(q, D[v]) \Rightarrow (q', D'[v])$ is a symbolic transition (Definition 2.2.1) in the zone graph (Definition 2.2.4) of $\mathcal{A}[v]$.
2. For each $v \in [\![C]\!]$, if $(q, D[v]) \Rightarrow (q', Z)$ is a symbolic transition in the zone graph of $\mathcal{A}[v]$, then $v \in [\![C']\!]$ and $Z = D'[v]$.

In particular (γ, C', D') is a valid guided constrained PDBM, i.e., $\gamma \models C'$.

6.2 γ-Complete Parametric Zone Graph

A γ-Complete Parametric Zone Graph is a Parametric Zone Graph (Definition 5.3.1) where the states are pairs of states of the PTA $\mathcal{A}$ and guided constrained PDBMs, i.e., $(q, (\gamma, C, D))$ and also contains disabled edges (since we also want constraints for which certain transitions in $\mathcal{A}$ are disabled).

Definition 6.2.1 (γ-Complete Parametric Zone Graph with Simulation $(PZG_\gamma^{LU}(\mathcal{A}))$**.** The γ-Complete Parametric Zone Graph with Simulation of a PTA $\mathcal{A}$ given LU bounds satisfied by γ is a transition system (Σ, σ_0, τ) where

- $\Sigma = \{(q, (\gamma, C, D)) \mid q \in Q$ and (γ, C, D) is a constrained PDBM$\}$ is the set of states
- $\sigma_0 = (q_0, (\gamma, C_0, D_0))$ where C_0 is the constraint representing the necessary and sufficient conditions on parameter valuations that specify the LU bounds and the bounds on the parameters, and D_0 is the PDBM corresponding to the zone $\bigwedge_{x \in X} x \geq 0 \wedge \bigwedge_{x,y \in X} x = y$ is the initial state
- τ is the set of transitions and is of two types. Given $(q, g, R, q') \in T$ and a state $(q, (\gamma, C, D))$, we have exactly one of the following kinds transitions in τ.

Simulation $(q,(\gamma,C,D)) \rightsquigarrow_{\tilde{C}} (q',(\gamma,C',D'))$ and there is a state $(q,(\gamma,C',D')) \in \Sigma$ such that $[\![\tilde{C}]\!] \subseteq [\![C \wedge C']\!]$ and for all $v \in [\![\tilde{C}]\!]$, $D[v] \subseteq \mathfrak{a}_{\preccurlyeq_{LU[v]}}(D'[v])$ and $\gamma \in [\![\tilde{C}]\!]$.

Successor $((q,(\gamma,C,D))) \rightarrow (q',(\gamma,C',D'))$ such that for all $v \models C'$, $D'[v] = \overrightarrow{[R]\mathcal{C}(D[v] \cap g)}$ and for all states $(q,(\gamma,C'',D'')) \in ZG_{\mathcal{A}}$, for all $v \models C'$, $D[v] \not\subseteq \mathfrak{a}_{\preccurlyeq_{LU[v]}}(D''[v])$, or $v \not\models C''$.

Disabled $((q,(\gamma,C,D))) \dashrightarrow (q',(\gamma,C',D'))$ such that for all $v \in [\![C']\!]$, $D[v]$ has no successor.

6.3 Algorithm to Compute $[\gamma]_{zg}$

Before we present the algorithm, we discuss a few preliminaries.

Disabled Edges There may be parameter valuations γ' for which $PZG^{LU'}(\mathcal{A})[\gamma]$ after removing the zones from each node is a subgraph of $PZG^{LU'}(\mathcal{A})[\gamma']$ after removing the zones from each node. To ensure equality, we will need to ensure that if some transition is disabled for γ, it should be disabled for all parameter valuations γ' that we return. Hence we collect those constraints around γ that lead to empty parametric zones too.

Definition 6.3.1 (Constraint of a PZG)**.** The constraint of a parametric zone graph ZG, $\textsc{Constraint}(ZG) = \bigwedge\{C \mid (q,(\gamma,C,D)) \in ZG\}$.

Algorithm 7 computes the γ-complete parametric zone graph with simulation of the PTA $\mathcal{A}$ and the contraint representing $[\gamma]_{zg}$ is $\textsc{Constraint}(ZG)$.

Lemma 6.3.1. Given a PTA $\mathcal{A}$ and a reference parameter valuation γ, the graph computed by Algorithm 7, $ZG = PZG^{LU}_{\gamma}(\mathcal{A})$.

Proof. It is easy to see that the initial state of ZG is the same as that of $PZG^{LU}_{\gamma}(\mathcal{A})$ and it is added to *Waiting*. So we need only show that for each $(q,(\gamma,C,D))$ in *Waiting* we add all outgoing edges of the node in $PZG^{LU}_{\gamma}(\mathcal{A})$ to ZG. There are 3 cases.

1. If $(q,(\gamma,C,D)) \rightsquigarrow_{C\setminus\tilde{C}} (q,(\gamma,C',D'))$ is a simulation edge in $PZG^{LU}_{\gamma}(\mathcal{A})$ the Lemma 5.2.1 shows that it is added to ZG at line 13.

2. If $(q,(\gamma,C,D)) \rightarrow (q',(\gamma,C',D'))$ is a successor edge in $PZG^{LU}_{\gamma}(\mathcal{A})$, then Lemma 6.1.1 and Lemma 5.2.2 shows that the edge is added to ZG and that for no $(q,(\gamma,C',D'))$, zones in (γ,C,D) is simulated by corresponding zones in (γ,C',D') at line 19.

3. If $(q,(\gamma,C,D)) \dashrightarrow (q',(\gamma,C',\emptyset))$ is a disabled edge in $PZG^{LU}_{\gamma}(\mathcal{A})$, then zones in (γ,C,D) neither have a successor nor are simulated so the above two arguments show they are neither successor nor simulation edges are added from it, and hence is added as a disabled edge to ZG in line 21.

□

It is now easy to that the following lemma holds.

Algorithm 7 Zone Graph Equivalence

1: **procedure** ZGEQ(γ, $\mathcal{A}$, M)
2: Let (LU, C) be LU bounds such that $\gamma \models C$
3: $C \leftarrow C \bigwedge_{p \in P} 0 \leq p \leq M_p$ ▷ All parameters are within bounds
4: $\mathit{Waiting} \leftarrow \{(q_0, (\gamma, C, D_0))\}$
5: $\mathit{Passed} \leftarrow \emptyset$
6: $ZG \leftarrow \emptyset$ ▷ The Zone Graph defined as a set of edges
7: **while** $\mathit{Waiting} \neq \emptyset$ **do**
8: Remove $(q, (\gamma, C, D))$ from $\mathit{Waiting}$
9: $\mathit{Passed} \leftarrow \mathit{Passed} \cup \{(q, (\gamma, C, D))\}$
10: **for** $(q, g, R, q') \in T$ **do**
11: **for** $(q, (\gamma, C', D')) \in \mathit{Passed}$ **do**
12: $(\tilde{C}, D) \leftarrow \mathsf{NS}_{LU}((C, D), (C', D'))$
13: $ZG \leftarrow ZG \cup \{(q, (\gamma, C, D)) \rightsquigarrow_{C \setminus \tilde{C}} (q, (\gamma, C', D'))\}$
14: $(\hat{C}, D) \leftarrow (\hat{C} \wedge \tilde{C}, D)$
15: **if** $\gamma \models \hat{C}$ **then**
16: $(\gamma, C', D') \leftarrow \overrightarrow{[R]\boldsymbol{\mathcal{C}}((\gamma, \hat{C}, D) \cap g)}$
17: **if** $[\![(C', D') \neq \emptyset]\!]$ **then**
18: $\mathit{Waiting} \leftarrow \mathit{Waiting} \cup \{(q', (\gamma, C', D'))\}$
19: $ZG \leftarrow ZG \cup \{(q, (\gamma, C, D)) \rightarrow (q', (\gamma, C', D'))\}$
20: **else**
21: $ZG \leftarrow ZG \cup \{(q, (\gamma, C, D)) \dashrightarrow (q', (\gamma, C', \emptyset))\}$
22: **return** ZG

Lemma 6.3.2. Given a PTA $\mathcal{A}$ and a reference parameter valuation γ, $[\gamma]_{zg} = \textsc{Constraint}(\text{ZGEQ}(\mathcal{A}, M))$.

The following lemma also holds. The proof is essentially the same as that for Lemma 5.4.2.

Lemma 6.3.3. The algorithm ZGEQ terminates.

6.4 Algorithm for Parameter Synthesis

In this section we provide an algorithm to compute integer parameter valuations using zone graph equivalence. Remarks 4.3.1 and 4.3.3 show that the parametric zone graph can potentially add an exponential (in the number of clocks) number of constrained PDBMs to the zone graph with each transition. However in the previous sections we see that the guided parametric zone graph adds atmost one parametric zone with each transition. This hints at a possibly faster algorithm for parameter synthesis.

The idea is to pick a parameter valuation from the domain of parameter valuations and find the zone graph equivalent parameter valuations. We know that either these valuations lead to at least one accepting run in their corresponding TAs, or they don't. We then continue with a parameter valuation not considered before. The process continues till all parameter valuations have been considered, and this way we partition the parameter valuation space into sets of "good" and not "good" valuations, where a "good" valuation gives a TA with an accepting run.

The caveat however is that we do not have good bounds on the number of parameter valuations that are picked for exploration.

Algorithm 8 Parameter Synthesis with $\equiv_{zg}$

```
1: procedure ZGEQSynth((A, M))
2:     Passed ← ⊥
3:     C ← ⊥
4:     while v ∈ M^P such that v ⊭ Passed do
5:         ZG ← ZGEQ(v, A, M)
6:         if there is an accepting PTA state in some node of ZG then
7:             C ← C ∨ Constraint(ZG)
8:         Passed ← Passed ∨ Constraint(ZG)
9:     return C
```

Chapter 7

Conclusion

In this thesis we explored two solutions to the bounded integer parameter synthesis problem for parametric timed automata, and incorporated the $\mathfrak{a}_{\preccurlyeq_{LU}}$ abstraction for timed automata, the coarsest abstraction given LU bounds, into parametric zone graphs to get a fast and terminating algorithm for the problem.

The first solution is to generate the entire parametric zone graph and collect all the parameter valuations from the nodes containing final states of the PTA. These parameter valuations have an accepting run in their corresponding TAs. The parametric zone graph gets exponentially large with each transition followed.

The second solution uses zone graph equivalence, where zone graph equivalent parameter valuations can be computed quickly. The idea is to partition the domain of parameter valuations using zone graph equivalence by computing the equivalence classes of parameter valuations picked one by one, into sets of "good" and not "good" valuations, where a "good" valuation has an accepting run in its corresponding TA.

7.1 Future Work

We plan to implement the algorithms presented here in the tool TCHECKER [HP] and run experiments to empirically determine the efficiency of our algorithms.

We further plan to incorporate other known developments for TA like partial-order reductions in the parametric setting.

Bibliography

[AA22] Johan Arcile and Étienne André. "Zone extrapolations in parametric timed automata". In: *NASA Formal Methods Symposium*. Springer. 2022, pp. 451–469.

[AD94] Rajeev Alur and David L Dill. "A theory of timed automata". In: *Theoretical computer science* 126.2 (1994), pp. 183–235.

[AF10] Étienne André and Laurent Fribourg. "Behavioral Cartography of Timed Automata". In: *Reachability Problems, 4th International Workshop, RP 2010, Brno, Czech Republic, August 28-29, 2010. Proceedings*. Ed. by Antonín Kucera and Igor Potapov. Vol. 6227. Lecture Notes in Computer Science. Springer, 2010, pp. 76–90. DOI: 10.1007/978-3-642-15349-5_5. URL: https://doi.org/10.1007/978-3-642-15349-5%5C_5.

[AHV93] Rajeev Alur, Thomas A Henzinger, and Moshe Y Vardi. "Parametric real-time reasoning". In: *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*. 1993, pp. 592–601.

[ALR15] Étienne André, Didier Lime, and Olivier H Roux. "Integer-complete synthesis for bounded parametric timed automata". In: *International Workshop on Reachability Problems*. Springer. 2015, pp. 7–19.

[And+08] Étienne André et al. "An Inverse Method for Parametric Timed Automata". In: *Electron. Notes Theor. Comput. Sci.* 223 (2008), pp. 29–46. DOI: 10.1016/j.entcs.2008.12.029. URL: https://doi.org/10.1016/j.entcs.2008.12.029.

[And21] Étienne André. "IMITATOR 3: Synthesis of timing parameters beyond decidability". In: *International Conference on Computer Aided Verification*. Springer. 2021, pp. 552–565.

[Beh+03] Gerd Behrmann et al. "Static guard analysis in timed automata verification". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2003, pp. 254–270.

[Beh+06a] Gerd Behrmann et al. "Lower and upper bounds in zone-based abstractions of timed automata". In: *International Journal on Software Tools for Technology Transfer* 8.3 (2006), pp. 204–215.

[Beh+06b] Gerd Behrmann et al. "Uppaal 4.0". In: (2006).

[BL09] Laura Bozzelli and Salvatore La Torre. "Decision problems for lower/upper bound parametric timed automata". In: *Formal Methods in System Design* 35.2 (2009), pp. 121–151.

[Boz+98] Marius Bozga et al. "Kronos: A model-checking tool for real-time systems". In: *International symposium on formal techniques in real-time and fault-tolerant systems*. Springer. 1998, pp. 298–302.

[BY03] Johan Bengtsson and Wang Yi. "Timed automata: Semantics, algorithms and tools". In: *Advanced Course on Petri Nets*. Springer. 2003, pp. 87–124.

[Dil89] David L Dill. "Timing assumptions and verification of finite-state concurrent systems". In: *International Conference on Computer Aided Verification*. Springer. 1989, pp. 197–212.

[DT98] Conrado Daws and Stavros Tripakis. "Model checking of real-time reachability properties using abstractions". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 1998, pp. 313–329.

[HP] Frédéric Herbreteau and Gerald Point. *TChecker*. Available at https://github.com/fredher/tchecker.

[HSW16] Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. "Better abstractions for timed automata". In: *Inf. Comput.* 251 (2016), pp. 67–90. DOI: 10.1016/j.ic.2016.07.004. URL: https://doi.org/10.1016/j.ic.2016.07.004.

[Hun+02] Thomas Hune et al. "Linear parametric model checking of timed automata". In: *The Journal of Logic and Algebraic Programming* 52 (2002), pp. 183–220.

[JLR15] Aleksandra Jovanovic, Didier Lime, and Olivier Henri Roux. "Integer parameter synthesis for real-time systems". In: *IEEE Transactions on Software Engineering* 41.5 (2015), pp. 445–461.

[Lim+09] Didier Lime et al. "Romeo: A parametric model-checker for Petri nets with stopwatches". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2009, pp. 54–57.

[LS00] François Laroussinie and Philippe Schnoebelen. "The state explosion problem from trace to bisimulation equivalence". In: *International Conference on Foundations of Software Science and Computation Structures*. Springer. 2000, pp. 192–207.

[RLA21] Mathias Ramparison, Didier Lime, and Étienne André. "Parametric updates in parametric timed automata". In: *Logical Methods in Computer Science* 17 (2021).

[TaŞ+96] Serdar TaŞiran et al. "Verifying abstractions of timed systems". In: *International Conference on Concurrency Theory*. Springer. 1996, pp. 546–562.