Concrete Delays for Symbolic Traces

Danny Bøgsted Poulsen & Jonas van Vliet
{*dannybp, jonasvv*}*@cs.aau.dk*
Department of Computer Science, Aalborg University

June, 2010

Aalborg University
Department of Computer Science

TITLE:
Concrete Delays for Symbolic Traces

PROJECT PERIOD:
DAT6:
1^{st} of February 2010 -
1^{st} of June 2010

PROJECT GROUP:
d620a

GROUP MEMBERS:

Danny Bøgsted Poulsen
Jonas van Vliet

SUPERVISOR:
Kim Guldstrand Larsen

CO-SUPERVISORS:
Alexandre David
Jiří Srba

NUMBER OF COPIES: 6

TOTAL PAGES: 87

RESOURCES:
`http://www.launchpad.net/ctu`

SYNOPSIS:

The goal of model checking is verifying that a system adheres to its specification. The model checking tool UPPAAL verifies Timed Automata and returns diagnostic traces to help users understand violations of the specifications. Due to the time abstractions used in the computations, the diagnostic traces returned are hard to comprehend for users, as they do not contain concrete delays.
In this thesis we describe several approaches to generating concrete delays for diagnostic traces. Two of these approaches are based on the diagnostic traces that violate safety properties and one approach is based on the diagnostic traces that violate liveness properties. We prove the correctness of all approaches and test the safety approaches against each other. The liveness approach is tested for viability.

Preface

The work at hand documents the work conducted during the DAT6 semester at Aalborg University and forms our master thesis. The work has been conducted within the Distributed and Embedded Systems group at the Department of Computer Science. Some of the material represents work conducted during the DAT5 semester [19]. The work has been improved and rewritten. The sections that are derived from previous work are 3.1, 3.2, 3.3 and 5.2.

We would like to thank Kim Guldstrand Larsen for always providing interesting suggestions and pointing us in the right direction. Furthermore, we would also like to thank Alexandre David for his help in improving our understanding of UPPAAL and for providing an experimental version of UPPAAL that allowed us to test the liveness approach. Finally we would like to thank Jiří Srba for his input on the problems we have faced and for his feedback on numerous thesis drafts.

Danny Bøgsted Poulsen

Jonas van Vliet

Contents

Chapter 1

Introduction

1.1 Reactive Systems

Since their introduction, computers have often been considered black boxes that given an input produce an output. Today, many computer systems are not supposed to terminate, and accept input throughout the entire computation. This has led to a different view on computer systems. A reactive system is a nonterminating computer system that accepts inputs at any time during computation. An input to a reactive system is called a stimulus. A reactive system alters its behaviour according to the stimulus given. Therefore the system does not always react similarly to the same stimulus. The way a reactive system reacts to stimuli and what stimuli they accept is referred to as the state of the reactive system.

A property of a system is a characterisation of its behaviour. A system specification can be translated into a set of properties that the system must satisfy. We split the properties into the following categories [5]:

Safety A safety property is a property that is violated by a finite execution.

Liveness A liveness property is a property that is violated by an infinite execution.

Consider a mutual exclusion protocol. In this case, a safety property is that no two processes are within their critical section at the same time. A liveness property is that any process that requests access to its critical section is eventually granted permission. The goal is to verify that certain properties hold for a given system.

1.2 Software Verification

Reactive systems are used in a wide range of embedded systems that are both business-, mission- and life-critical and verifying that these systems adhere to their specification is important. Testing is a verification process in which an implemented system is run on various inputs. The output is then validated for correctness and if the output is incorrect an error has occurred. When testing, a subset of the executions of the implemented system are examined, hence testing is only adequate for finding errors. A successful test proves the program can return correct output but does not refute the existence of errors.
Computer systems today are to a great extent component based. The components compute in

parallel and communicate according to protocols. A component can be verified once it is implemented, but the entire system must be implemented before the communication between the components can be verified. Finding an error in the communication protocol at this stage is expensive as fixing it may require rewriting large parts of the system.

Model checking is, contrary to testing, not working on an implemented system. Instead a formal model of the system is constructed and tested against the properties. To prove or disprove whether a modelled system satisfies a property, model checking tools explores states of the modelled system. This is both an advantage and disadvantage. Model checking can be used to disprove certain behaviour, but when the state space is very large (a problem often referred to as a state space explosion), checking all states is not feasable. As testing only considers a part of the state space, it is not affected by this problem.

Model checking tools can disprove the existence of certain behaviour by exploring all states of a modelled system. However, model checking can never fully substitute testing as it works on a model of the final system and not on the actual implementation. This creates a gap between the properties of the model and the properties of the implementation, as there is no guarantee that the implementation matches the model. Model checking can however help the industry verify their system specification before implementing the system and thereby minimise the risk of having to rewrite a part of the system.

1.3 Diagnostic Traces

Model checking tools are not only capable of answering whether a property is satisfied, but in some cases they even provide a diagnostic trace as proof. The trace can help developers to understand the behaviour of a system.

From a computational perspective, model checking tools need abstractions of the state space to verify properties. The diagnostic traces returned are subject to these abstractions, hence the information therein can be difficult to comprehend. Consider the Timed Automaton in Figure 1.1.

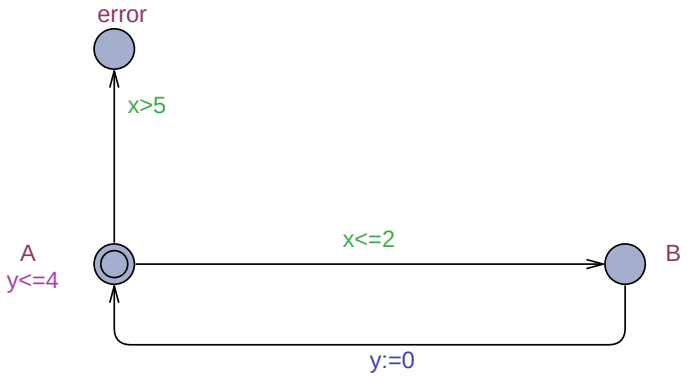


Figure 1.1: A Timed Automaton modelled in UPPAAL. The model contains an error location, and reaching this location is undesirable.

Reaching the error location is undesired. We therefore attempt verify in UPPAAL [18] that the Timed Automaton never enters the error location. Unfortunately, this is not the case as UPPAAL

can find a counter-example. To prove the system can reach the error location, UPPAAL provides a diagnostic trace as follows,

$$\begin{aligned}&\langle A, x \geq 0 \ \wedge \ x = y\rangle \overset{x\leq 2}{\rightsquigarrow} \langle B, 0 \leq x \leq 2 \wedge x = y\rangle \\ \overset{y:=0}{\rightsquigarrow}&\langle A, 0 \leq x \leq 2 \ \wedge \ y \leq 4 \ \wedge \ x - y \leq 2\rangle \overset{x>5}{\rightsquigarrow} \langle error, 5 < x \ \wedge \ x - y \leq 2\rangle.\end{aligned}$$

The generated diagnostic trace is symbolic in the sense that the values of the clocks are represented as constraints. This makes it hard to comprehend how the error state is reached. It is preferable to obtain a concrete diagnostic trace containing delays and clock values such as,

$$\begin{aligned}&\langle A, [x = 0, y = 0]\rangle \xrightarrow{1.5} \langle A, [x = 1.5, y = 1.5]\rangle \\ \xrightarrow{x\leq 2}&\langle B, [x = 1.5, y = 1.5]\rangle \xrightarrow{2} \langle B, [x = 3.5, y = 3.5] \\ \xrightarrow{y:=0}&\langle A, [x = 3.5, y = 0]\rangle \xrightarrow{2} \langle error, [x = 5.5, y = 2]\rangle \\ \xrightarrow{x>5}&\langle error, [x = 5.5, y = 2]\rangle.\end{aligned}$$

The focus of this thesis is creating concrete diagnostic traces from symbolic diagnostic traces.

1.4 Structure of Thesis

The remainder of this thesis is structured as follows. In Chapter 2 we introduce Timed Automata as a formal model for reactive systems. In Chapter 3 we introduce data structures and algorithms that are widely used throughout this thesis. In Chapter 4 the symbolic semantics of Timed Automata is introduced. The semantics introduce the abstraction which causes the problem we address. Chapter 5 presents two different algorithms for solving the problem for safety traces and Chapter 6 presents one algorithm to solve it for liveness traces. In Chapter 7 we present a prototype implementation of the ideas presented throughout this thesis. Finally in Chapter 8 we present our conclusion and possible future work.

Chapter 2

Preliminaries

In the previous chapter we discussed modelling formalisms and the role they play in software verification. In this chapter, we introduce the formalism of Timed Automata, which has established itself as a standard for modelling reactive systems. Since their introduction by Alur and Dill [3, 4], Timed Automata have been applied in verification tools such as UPPAAL [18] and KRONOS [22] and their ability to verify reactive systems has been proven in case studies [15]. In this chapter we first introduce Timed Automata by example.

2.1 Timed Automata

Consider the Timed Automaton in Figure 2.1 modeling a vending machine. The model contains a fixed set of clocks, where each clock is a counter that increases from 0 to infinity. The values of these clocks are not present in the model. Each circle in the model represents a location. The Timed Automaton in Figure 2.1 contains the locations ℓ_0, ℓ_1, ℓ_2 and ℓ_3. The state of a Timed Automaton is the location it is in and the value of each clock.

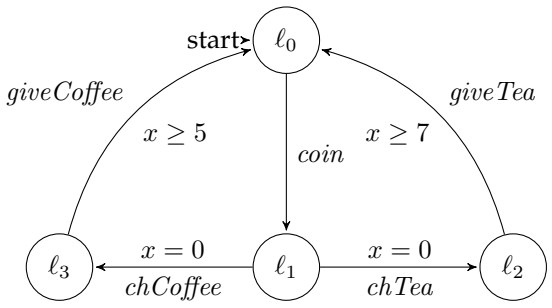


Figure 2.1: A Timed Automaton modeling a vending machine

The arrows between locations represent edges that allow the automaton to transit from one location to another. We call this "taking an edge". To take an edge, the current state of the automaton must satisfy the enabling condition of the edge, which is a boolean formula over the values of the clocks. Note that $x = 0$ is not a boolean formula. We call such an enabling condition a guard. In the model, a guard is depicted as the expression close to an edge. To allow

communication between an automaton and its environment, edges can perform synchronisations.

A synchronisation is a handshake over a channel and intuitively consists of a sending participant (denoted by the prefix !) and a receiving participant (denoted by the prefix ?). In the example in Figure 2.1, the edge from ℓ_1 to ℓ_2 performs a synchronisation, as it waits for the environment to press a button, hereby choosing coffee or tea. The edge from ℓ_2 to ℓ_0 performs a synchronisation, when it outputs tea. To manipulate the values of the clocks, edges can also apply resets to clocks. When a clock is reset, its value becomes 0. In the Timed Automaton, $x = 0$ denotes a reset of clock x.

The only way to change the state of the automaton besides taking an edge is delaying. When delaying, the value of every clock is increased by the same amount and the location remains the same.

Observe the Timed Automaton in Figure 2.1. From the start location it can accept a coin and will move to a state in which it awaits the user giving a coin. Once in a state where the location is ℓ_1, it waits for the user to choose either coffee or tea. Depending on the selection, it will enter a location from which it can produce the selected drink. Assume coffee is selected. The reset $x = 0$ on the edge from ℓ_1 to ℓ_3 implies that at least 5 time units must pass before the edge from ℓ_3 to ℓ_0 can be taken.

The vending machine modelled in Figure 2.1 does not guarantee ever producing the selected drink. It can accept a coin, let the user choose a drink and afterwards never produce the selected drink by delaying forever. To prevent this behaviour, invariants are added to the locations of the automaton. An invariant is a boolean formula that poses a restriction on the values that clocks are allowed to have in a location. Consider the automaton in Figure 2.2, which is a copy of the automaton in Figure 2.1 where invariants have been added. In ℓ_3, the value of clock x must always be less than 6. Since clock x cannot be reset from this location, it is only possible to delay 6 time units.

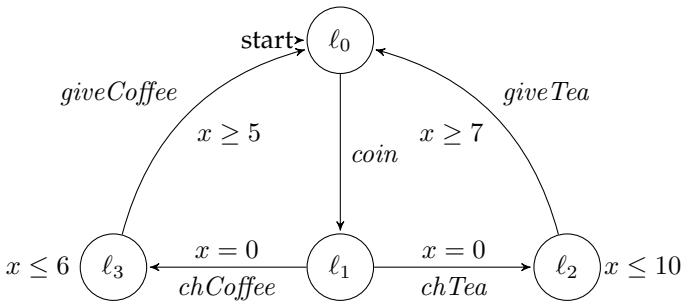


Figure 2.2: The Timed Automaton extended with invariants.

The means to express timing constraints is the notion of difference constraints. A difference constraint is a constraint on the difference between two clocks. If x and y are clocks then $x - y \lhd n$ where $\lhd \in \{\leq, <\}$ and $n \in \mathbb{Z}$ is a difference constraint. Difference constraints are only useful to express bounds on the difference between clocks. To express constraints on a single clock such as $x \leq 5$, we introduce a pseudo clock that always has the value zero. We denote this clock $\mathbf{0}$ and can then represent the beforementioned constraint as $x - \mathbf{0} \leq 5$. Finally, for a set of clocks $\mathcal{C}$ where $\{\mathbf{0}\} \subseteq \mathcal{C}$, we define $DC^{\mathbb{Z}}(\mathcal{C})$ to be all possible difference constraints over $\mathcal{C}$. We continue by defining Timed Automata formally. Note that synchronisations have been left out of this

definition deliberately, as we do not need them and they have no impact on our results.

Definition 1 (Timed Automata)

A *Timed Automaton* is a tuple $(\mathcal{L}, \ell_0, \mathcal{C}, E, \mathtt{I})$, where

- $\mathcal{L}$ is a finite set of locations in the automaton,
- $\ell_0 \in \mathcal{L}$ is the initial location,
- $\mathcal{C}$: is a finite set of clocks where $\mathbf{0} \in \mathcal{C}$.
- $E \subseteq L \times 2^{DC^{\mathbb{Z}}(\mathcal{C})} \times 2^{\mathcal{C}} \times L$ is a finite set of edges, and
- $\mathtt{I} : \mathcal{L} \to 2^{DC^{\mathbb{Z}}(\mathcal{C})}$ assigns invariants to locations. ♢

We often write $\ell \xrightarrow{g,r} \ell'$ instead of (ℓ, g, r, ℓ').

The semantics of a Timed Automaton is given as a timed transition system, which has the following syntax:

Definition 2 (Timed Transition System)
A timed transition system is a tuple $(S, s_0, Lab, \to)$, where

- S is a set of states
- $s_0 \in S$ is the initial state
- Lab is a finite set of labels, and
- $\to \subseteq S \times (\mathbb{R}_{\geq 0} \cup \Sigma) \times S$ ♢

To simplify the notation we often write,

- $s \xrightarrow{d} s$ when $(s, d, s) \in \to$ and $d \in \mathbb{R}_{\geq 0}$, and
- $s \xrightarrow{a} s'$ when $(s, a, s') \in \to$ and $a \in Lab$.

A timed transition system is based on states and transitions between these. Recall that a state in a Timed Automaton contains a location and the value of each clock. We represent the clock values as a function $v : \mathcal{C} \to \mathbb{R}_{\geq 0}$, where $v(\mathbf{0}) = 0$. We call such a function a valuation. By $[x = 4, y = 5]$ we denote a valuation v where $v(x) = 4$ and $v(y) = 5$.
Using valuations we express a concrete state as $\langle \ell, v \rangle$, where ℓ is the location the Timed Automaton is in and v describes the values of each clock.

Before properly defining the semantics of Timed Automata, we define operations on valuations to assist us in defining the transition relation. We say two valuations v and v' are equal, written $v = v'$ if for all clocks $x \in \mathcal{C} : v(x) = v'(x)$. By v_0 we denote the valuation where all clocks have the value zero.
To capture the delaying and resets performed by a Timed Automaton we introduce a delay and reset operator on valuations. For a valuation v, any delay $d \in \mathbb{R}_{\geq 0}$ and any reset $r \subseteq \mathcal{C}$ we define

the valuations $v + d$ and $v_{r=0}$ as

$$(v+d)(x) = \begin{cases} v(x)+d & \text{if } x \neq \mathbf{0} \\ v(x) & \text{if } x = \mathbf{0} \end{cases}$$

$$v_{r=0}(x) = \begin{cases} 0 & \text{if } x \in r \\ v(x) & \text{if } x \notin r \end{cases}$$

Lastly we need a way of expressing when a valuation satisfies a guard or invariant. For this matter we define a satisfaction relation $\vDash$ between a valuation v over $\mathcal{C}$ and a difference constraint $(x - y \lhd n) \in DC^{\mathbb{Z}}(\mathcal{C})$ as

$$v \vDash (x - y \lhd n) \text{ if } v(x) - v(y) \lhd n.$$

For a set of difference constraints $g \subseteq DC^{\mathbb{Z}}(\mathcal{C})$, we write $v \vDash g$ if for all $\delta \in g$, $v \vDash \delta$.

We now define the concrete semantics of Timed Automata.

Definition 3 (Concrete Semantics of Timed Automata)

The semantics of a Timed Automaton $A = (\mathcal{L}, \ell_0, \mathcal{C}, E, \mathtt{I})$ is a timed transition system $(S, s_0, Lab, \rightarrow)$, where

- $S = \{\langle \ell, v \rangle \mid \ell \in \mathcal{L} \ \wedge \ v : \mathcal{C} \rightarrow \mathbb{R} \ \wedge \ v \vDash \mathtt{I}(\ell)\}$,
- $s_0 = \langle \ell_0, v_0 \rangle$, and
- $Lab = (DC^{\mathbb{Z}}(\mathcal{C}) \times 2^{\mathcal{C}}) \cup \mathbb{R}_{\geq 0}$.

The transition relation $\rightarrow \subseteq S \times Lab \times S$, is defined as follows:

Delay $\langle \ell, v \rangle \xrightarrow{d} \langle \ell, v' \rangle$ where $d \in \mathbb{R}_{\geq 0}$ and $v' = v + d$.

Action $\langle \ell, v \rangle \xrightarrow{(g,r)} \langle \ell', v' \rangle$ if there exists an edge $\ell \xrightarrow{g,r} \ell' \in E$, $v \vDash g$, $v' = v_{r=0}$. $\diamondsuit$

We call $\langle \ell, v \rangle$ a concrete state. Note that since the state space is restricted by the invariant this also implies that certain delay and action transitions cannot be performed.

Using the concrete semantics of Timed Automata, a possible execution of the Timed Automaton in Figure 2.2 is

$$\langle \ell_0, [x=0] \rangle \xrightarrow{(\emptyset,\emptyset)} \langle \ell_1, [x=0] \rangle \xrightarrow{(\emptyset,\{x\})} \langle \ell_3, [x=0] \rangle \xrightarrow{5.8} \langle \ell_3, [x=5.8] \rangle \xrightarrow{(x \geq 5,\emptyset)} \langle \ell_0, [x=5.8] \rangle.$$

Chapter 3

Difference Constraints

In the previous chapter we used the concept of difference constraints to express guards and invariants of Timed Automata. The difference constraints were restricted to contain only integer bounds. In this chapter, we generalise difference constraints, present two alternative representations of difference constraints and provide general results for them all.

3.1 Difference Constraints

In this section, we formally define a general class of difference constraints that allows rational numbers as bounds as well as infinity bounds. Furthermore, well-known relations and operators for these constraints are introduced.

Definition 4 (Difference Constraints)
Let $\{\mathbf{0}\} \subseteq \mathcal{C}$ be a set of variables. We define the set of all difference constraints over $\mathcal{C}$ as

$$DC(\mathcal{C}) \stackrel{def}{=} \{x - y \lhd n \mid x, y \in \mathcal{C} \ \wedge \ \lhd \in \{<, \leq\} \ \wedge \ n \in \mathbb{Q} \cup \{\infty\}\}.$$

This set allows difference constraints such as $x - y \leq \frac{1}{2}$ and $x - y < \infty$. Note that when we write $x - y \lhd n$, either $n \in \mathbb{Q}$ or $n = \infty$. Recall the satisfaction relation $\vDash$ between a valuation and a set of difference constraints. Any constraint on the form$(x - y \lhd \infty) \in DC(\mathcal{C})$ is satisfied by any valuation. We call such a constraint a trivial constraint and say it is trivially satisfied.

For convenience, we call any constraint on the form $x - \mathbf{0} \lhd n$ where $x \neq \mathbf{0}$ an *upper bound*, as this constraint presents an upper limit on the value of clock x. Using a similar reasoning, we call any constraint on the form $\mathbf{0} - x \lhd n$ where $x \neq \mathbf{0}$ a *lower bound*. To avoid confusion throughout the paper we often write a lower bound as $\mathbf{0} - x \lhd -n$ which in practice means that $n \lhd x$. We call any constraint on the form $x - y \lhd n$ where $x \neq \mathbf{0}$ and $y \neq \mathbf{0}$ a *diagonal constraint*. Finally, we note that constraints on the form $x - x \lhd n$ are allowed. The satisfiability of these constraints does not depend on the value of x, as for any valuation v,

$$v(x) - v(x) = 0 \lhd n.$$

A valuation v is a solution to $\Delta \subseteq DC(\mathcal{C})$ if $v \vDash \delta$ for all $\delta \in \Delta$.
The set of all solutions to Δ is denoted $[\Delta]$ and we say that Δ is inconsistent if $[\Delta] = \emptyset$. In the later chapters, we will often refer to the set of solutions to Δ as the zone over Δ.

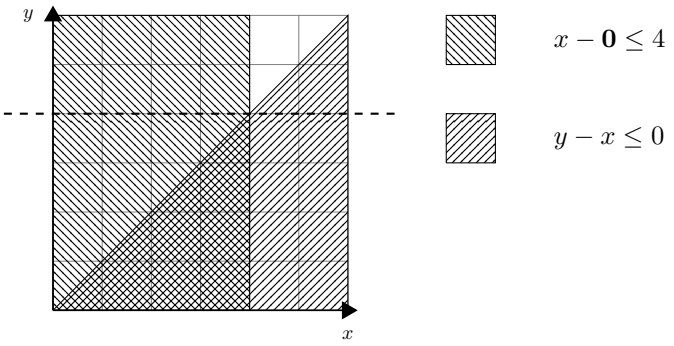


Figure 3.1: Valuations allowed by constraints.

Definition 4 allows multiple constraints over the same pair of variables. Whenever this is the case only one of them is significant. Consider the set $\{(x - \mathbf{0} \leq 5), (x - \mathbf{0} \leq 4)\}$. Obviously the first constraint has no impact on the set of solutions, as the second constraint is more restrictive. This is formalised as a tightness relation on difference constraints.

Definition 5 (Difference Constraint Relations)
Let $\delta = x - y \lhd n$ and let $\lambda = x - y \lhd' n'$ be difference constraints over the same variables. We write

- $\delta = \lambda$ if $\lhd = \lhd' \wedge n = n'$,
- $\delta \prec \lambda$ if $(n < n') \ \vee \ (n = n' \ \wedge \ \lhd =< \ \wedge \lhd' =\leq)$, or
- $\delta \preceq \lambda$ if $\delta = \lambda \ \vee \ \delta \prec \lambda$. $\diamondsuit$

Note that if $n \neq \infty$ then $n < \infty$. We say a constraint δ is tighter than λ if $\delta \prec \lambda$.

A set of difference constraints may contain implicit constraints i.e. constraints that are not present in the set of constraints, but still apply to the solutions. Consider the set

$$\{x - \mathbf{0} \leq 4, y - x \leq 0\}$$

and its solutions depicted in Figure 3.1. The marked areas represent the valuations that satisfy each constraint. The intersection of the marked areas contains the solutions of the set. Notice that there are no solutions for the set above the dashed line. This line can be represented as a constraint $y - \mathbf{0} \leq 4$, which can be added to the set of constraints without removing solutions. We wish to find these implicit constraints. To this end, we introduce addition between constraints.

Two constraints δ and λ are addition compatible if they are on the form

- $\delta = (x - y \lhd_1 n)$ and $\lambda = (y - z \lhd_2 m)$, or
- $\delta = (y - z \lhd_2 m)$ and $\lambda = (x - y \lhd_1 n)$

We define $\delta + \lambda$ to be the difference constraint $x - z \lhd m + n$, where $\lhd = \begin{cases} < & \text{if } \lhd_1 =< \vee \lhd_2 =< \\ \leq & \text{otherwise} \end{cases}$

For two addition compatible difference constraints δ and λ, we refer to $\delta + \lambda$ as their derived constraint. If δ and λ belong to the same set of difference constraints Δ, adding their derived constraint to Δ does not alter the zone, as stated in the following lemma.

Lemma 6
Let Δ be a set of difference constraints and $\delta, \lambda \in \Delta$ be addition compatible then $[\Delta \cup \{\delta + \lambda\}] = [\Delta]$. $\diamondsuit$

PROOF.
We split the proof into two cases:

Assume Δ is inconsistent.
Clearly $\Delta \cup \{\delta + \lambda\}$ cannot be consistent if Δ is inconsistent thus $\emptyset = [\Delta] = [\Delta \cup \{\delta + \lambda\}]$.

Assume Δ is consistent.
We wish to show that $[\Delta \cup \{\delta + \lambda\}] = [\Delta]$. We prove inclusion in both directions.

- $[\Delta \cup \{\delta + \lambda\}] \subseteq [\Delta]$
  Since adding constraints can only remove possible solutions from Δ the inclusion is trivial.
- $[\Delta] \subseteq [\Delta \cup \{\delta + \lambda\}]$
  Let $\delta = (x - y \lhd_1 n)$ and $\lambda = (z - x \lhd_2 m)$ then $\delta + \lambda = (z - y \lhd n + m)$. Let v be any solution to Δ and $n', m' \in \mathbb{R}_{\geq 0}$ such that

$$v(x) - v(y) = n' \lhd_1 n, \text{ and} \tag{3.1}$$
$$v(z) - v(x) = m' \lhd_2 m. \tag{3.2}$$

  From 3.1 and 3.2 we derive

$$v(z) - v(y) = n' + m' \lhd_1 n + m' \lhd_2 n + m,$$

  hence $v \vDash (\delta + \lambda)$. Then $[\Delta] \subseteq [\Delta \cup \{\delta + \lambda\}]$.

■

The notion of addition between constraints is generalised to a sum of multiple constraints in a straightforward manner. For a given pair of clocks, a set of difference constraints can contain any number of constraints over these. It is sometimes convenient to have exactly one constraints for each pair of variables. We say such a set of difference constraints is in simple form.

Definition 7 (Simple Form)
Let Δ be a set of difference constraints over the variables $\mathcal{C}$. We say Δ is in simple form if

$$\forall x, y \in \mathcal{C} : \exists!(x - y \lhd n) \in \Delta. \qquad \diamondsuit$$

For any set of difference constraints Δ, we let $\Delta^f = \Delta \cup \{x - y < \infty \mid x, y \in \mathcal{C}\}$ and define

$$\Delta^s \stackrel{def}{=} \{\delta \mid \delta \in \Delta^f \text{ s.t. } \forall \lambda \in \Delta^f, \delta \preceq \lambda\}.$$

This construction builds a set of difference constraints in simple form and leaves the set of solutions unchanged.

Lemma 8
Let $\Delta \subseteq DC(\mathcal{C})$, then Δ^s is in simple form and $[\Delta] = [\Delta^s]$. ♢

PROOF.
We first prove Δ^s is in simple form hence we prove that

$$\forall x, y \in \mathcal{C} : \exists!(x - y \lhd n) \in \Delta^m.$$

The intermediate set Δ^f clearly contains one constraints per pair of variables thus $\forall x, y \in \mathcal{C} : \exists(x - y \lhd n \in \Delta^f)$. As Δ^m contains one constraint from Δ^f per pair of variables, we conclude Δ^s is in simple form.
Since only trivial constraints are added and redundant constraints are removed it follows trivially that $[\Delta^s] = [\Delta]$. ■

3.2 Constraint Graph

In the previous section we introduced sets of difference constraints and properties concerning these and their zones. A zone is essentially a set of valuations. In this section we present another representation of a set of valuations.

A constraint graph is a directed graph where the vertices are variables and the edges correspond to constraints between variables. Figure 3.2 depicts a constraint graph over the variables $\mathbf{0}, x, y$.

Definition 9 (Constraint Graph)
A constraint graph over the variables $\{\mathbf{0}\} \subseteq \mathcal{C}$ is a triple $(\mathcal{C}, E, w)$ where,

- $E \subseteq \mathcal{C} \times \mathcal{C}$ is a set of edges, and
- $w : E \to (\mathbb{Q} \cup \{\infty\}) \times \{<, \leq\}$ is a weight function. ♢

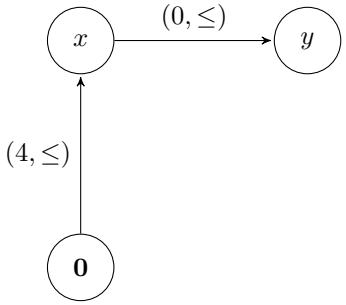


Figure 3.2: A constraint graph representing the constraints $x - \mathbf{0} \leq 4$ and $y - x \leq 0$.

Definition 10 (Solution to a constraint graph)
A solution to a constraint graph $(\mathcal{C}, E, w)$ is a valuation v over $\mathcal{C}$ such that for all $(x_i, x_j) \in E$ where $w(x_i, x_j) = (n, \lhd)$

$$v(x_j) - v(x_i) \lhd n.$$ ♢

A solution to the constraint graph in Figure 3.2 is any valuation v over $\{\mathbf{0}, x, y\}$ such that $v(x) - v(\mathbf{0}) \leq 4$ and $v(y) - v(x) \leq 0$.

For a constraint graph G we denote all solutions for G by $[G]$ and refer to it as the zone over G. It may seem that a constraint graph cannot express the same zone as a set of difference constraints. This is however not the case as we for any set of difference constraints can construct a constraint graph with the same zone. For a set of difference constraints Δ, we denote the constraint graph with the same zone by G^{Δ}.

Lemma 11
For a set of difference constraints Δ there exists a constraint graph G^{Δ} such that $[\Delta] = [G^{\Delta}]$. $\diamondsuit$

PROOF.
Let Δ be a set of difference constraints over the variables $\mathcal{C}$. By Lemma 8 $[\Delta^s] = [\Delta]$. We construct a constraint graph $G^{\Delta} = (\mathcal{C}, E, w)$ where

- $E = \{(x, y) \mid \exists (y - x \lhd n) \in \Delta$ for some $\lhd$ and $n\}$, and
- $w(x, y) = (n, \lhd)$ if $(y - x \lhd n) \in \Delta$.

The construction clearly gives $[G^{\Delta}] = [\Delta^s] = [\Delta]$. ∎

The edges in the constraint graph are annotated with weights $(n, \lhd)$. We define addition of these weights in a similar fashion to addition of constraints, hence

$$(m, \lhd_1) + (n, \lhd_2) = \begin{cases} (m + n, \leq) & \text{if } \lhd_1 = \leq \ \wedge \lhd_2 = \leq \\ (m + n, <) & \text{otherwise.} \end{cases}$$

We define the following comparison of weights

- $(m, \lhd_1) \prec (n, \lhd_2)$ if $m < n$ or $\lhd_1 = < \wedge \lhd_2 = \leq \ \wedge m = n$,
- $(m, \lhd_1) = (n, \lhd_2)$ if $m = n$ and $\lhd_1 = \lhd_2$, and
- $(m, \lhd_1) \preceq (n, \lhd_2)$ if $(m, \lhd_1) = (n, \lhd_2)$ or $(m, \lhd_1) \prec (n, \lhd_2)$.

A path in a constraint graph is a sequence of nodes connected by edges. Let $P = (x_1, \ldots, x_n)$ be a path. We define the weight of P as

$$w(P) = \sum_{i=0}^{n-1} w(x_i, x_{i+1}).$$

An interesting property of paths in constraint graphs is that their weights correspond to derived constraints. Let Δ be a set of difference constraints and let there exist a path P from x to y with weight $(n, \lhd)$. Then there exist a derived constraint δ of Δ such that $\delta = (y - x \lhd n)$. We call P a negative weight path, if $w(P) \prec (0, \leq)$.

We are often interested in finding the tighest constraint that can be derived for a pair of variables in a set of difference constraints. This constraint can be found by solving the shortest path problem in the corresponding constraint graph.

Lemma 12 (Tighest Constraint)
Let Δ be a set of difference constraints. If there exists a shortest path P from x to y in G^{Δ} and $w(P) = (n, \lhd)$, then $\nexists \delta_1, \delta_2, \ldots, \delta_k \in \Delta$ such that $\delta_1 + \delta_2 + \cdots + \delta_k \prec (y - x \lhd n)$.

PROOF.
Assume towards a contradiction that there exists a sequence of constraints $\delta_1, \delta_2, \delta_3, \ldots, \delta_k \in \Delta$ such that their sum $(y - x \lhd' n') \prec (y - x \lhd n)$. Due to the construction from Lemma 11 any constraint in Δ is also represented in G^{Δ}. It can be shown that this leads to a contradiction, as this implies that there exist a path P' such that $w(P') \prec w(P)$. ∎

3.3 Difference Bound Matrix

The final representation we consider is the standard representation used by model checkers such as UPPAAL [18]. Essentially the representation Difference Bound Matrix [6, 12] forms an adjacency matrix for a constraint graph. For a matrix H, let the element (i, j) be denoted by H_{ij}.

Definition 13 (Difference Bound Matrix (DBM))
Let $\mathcal{C} = \{x_0, x_1, x_2, x_3, \ldots, x_n\}$ be a set of variables such that $\mathbf{0} \in \mathcal{C}$. A Difference Bound Matrix H over the variables $\mathcal{C}$ is a $|\mathcal{C}| \times |\mathcal{C}|$ matrix where for all $i, j \in [0, n]$ $H_{ij} \in (\mathbb{Q} \cup \{\infty\}) \times \{<, \leq\}$.♢

Below is an example of a DBM over the variables $\{\mathbf{0}, x, y\}$.

	$\mathbf{0}$	x	y
$\mathbf{0}$	$(0, \leq)$	$(0, \leq)$	$(0, \leq)$
x	$(4, \leq)$	$(0, \leq)$	$(\infty, <)$
y	$(0, \leq)$	$(\infty, <)$	$(0, \leq)$

Definition 14 (Solution to a DBM)
Let H be a DBM over the variables $\mathcal{C} = \{x_0, x_1, x_2, \ldots, x_n\}$ and let $\mathbf{0} \in \mathcal{C}$. A solution to H is a valuation v over $\mathcal{C}$ such that for all $i, j \in [0, n]$,

$$v(x_i) - v(x_j) \lhd m,$$

where $H_{ij} = (m, \lhd)$. The set of all solutions to H is denoted $[H]$. ♢

DBMs can represent the exact same set of zones as a set of difference constraints. To this end, we first prove that a set of difference constraint can represent the same zone as as DBM.

Lemma 15
Let H be a DBM then there exists a set of difference constraints Δ such that $[H] = [\Delta]$. ♢

PROOF.
Proof by construction.
Let H be a DBM over the variables $\mathcal{C} = \{x_0, x_1, x_2, \ldots, x_n\}$. Let

$$\Delta = \{(x_i - x_j \lhd n) \mid H_{ij} = (n, \lhd)\}.$$

From the construction it follows easily that $[H] = [\Delta]$. ∎

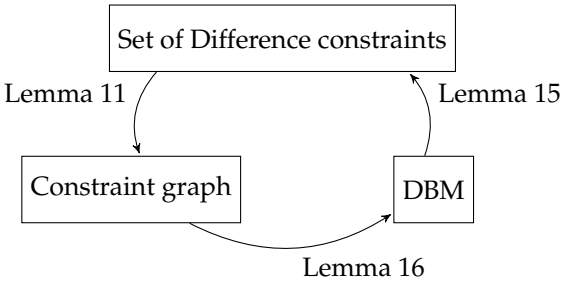
Secondly, we prove that a DBM can represent the same zone as a constraint graph and recall that a constraint graph can represent the same zone as a set of difference constraint. We have a circular equality with respect to their representative power.

Lemma 16
Let G be a constraint graph, then there exists a DBM H such that $[G] = [H]$. $\diamondsuit$

PROOF.
We construct H.
Let G be a constraint graph with nodes $\mathcal{C}$, edges E and weight function $w : E \to \mathbb{R} \times \{<, \leq\}$. Let each node in $\mathcal{C}$ have a unique index in the range $[0; |\mathcal{C}|[$ and let node $\mathbf{0}$ have the index 0. We convert G into a $|\mathcal{C}| \times |\mathcal{C}|$ matrix H where for all $i, j \in [0; |\mathcal{C}|[$:

$$H_{ij} = \begin{cases} (n, \lhd) & \text{if } (x_j, x_i) \in E \wedge w(x_j, x_i) = (n, \lhd) \\ (\infty, <) & \text{otherwise} \end{cases}$$

The construction makes it easy to see that $[G] = [H]$. ∎



The main difference between the representations is how restrictive they are with respect to the amount of constraints. In a set of difference constraints there is no limit whereas a constraint graph demands there is at most one constraint per pair of variable. The DBM representation is the most restrictive by insisting that there exists exactly one constraint per pair of variables.
As the representations are alike we take the liberty of using the representations interchangeably.

3.4 Closed Form

Sometimes a set of difference constraints contains constraints that in practice have no influence on the zone. Consider the set

$$\{y - \mathbf{0} \leq 5, x - \mathbf{0} \leq 4, y - x \leq 0\}.$$

The constraint $y - \mathbf{0} \leq 5$ has no influence on the possible solutions due to the derived constraint $(x - \mathbf{0} \leq 4) + (y - x \leq 0) = (x - \mathbf{0} \leq 4) \prec (y - \mathbf{0} \leq 5)$. A set of difference constraints is in closed form if it is in simple form and all constraints are as tight as possible. This concept was introduced by Dill [12].

Definition 17 (Closed Form)
A set of difference constraints Δ is closed if

- Δ is in simple form, and
- $\forall \delta, \delta' \in \Delta$ that are addition compatible, $\exists \lambda \in \Delta$ such that $\lambda \preceq \delta + \delta'$. $\diamondsuit$

We now describe the method proposed by Dill [12] to find the closed form of a set of difference constraints Δ for which G^{Δ} contains no negative weight cycles. To this end, we use the well-known Floyd-Warshall all-pairs shortest path algorithm [14] presented in Algorithm 1. For a set of difference constraints Δ we let Δ^c be the result of running Algorithm 1 on Δ.

Algorithm 1: Floyd-Warshall all-pairs shortest path algorithm.

Input: A DBM H over n variables
Output: An all-pairs shortest path closure of H

```
1 for k := 0 to n − 1 do
2     for i := 0 to n − 1 do
3         for j := 0 to n − 1 do
4             H_ij := min(H_ik + H_kj, H_ij);
5         end
6     end
7 end
8 return H
```

We first ensure that $[\Delta] = [\Delta^c]$ and afterwards that Δ^c is in closed form.

Lemma 18
Let Δ be a set of difference constraints then $[\Delta^c] = [\Delta]$. $\diamondsuit$

PROOF.
Observe that the only change Algorithm 1 applies to the DBM is the minimisation step in line 4. In this step, a constraint is possibly replaced by a tighter constraint, that is derived from other constraints in H. By Lemma 6, adding this constraint to H does not alter the zone. ■

Lemma 19
Let Δ be a set of difference where G^{Δ} has no negative weight cycles, then Δ^c is a closed form of Δ. $\diamondsuit$

PROOF.
For Δ^c to be in closed form the following conditions must be met.

- Δ^c must be in simple form
  The result of running Algorithm 1 is a DBM hence set of difference constraints in simple form.
- $\forall \delta, \delta' \in \Delta$ that are addition compatible, $\exists \lambda \in \Delta$ such that $\lambda \preceq \delta + \delta'$
  Because G^{Δ} has no negative weight cycles, we know a shortest path exists between every

pair of variables. As a shortest path exists, we know from Lemma 12 that the constraints in Δ^c are the tighest possible thus a tighter bound can not be derived. ∎

3.5 Solutions

In this section we focus on properties concerning the zone over a set of difference constraints. We first prove that whenever a constraint graph contains a negative weight cycle then it is inconsistent.

Lemma 20
Let Δ be a set of difference constraints. If G^Δ contains a negative weight cycle then Δ is inconsistent.

PROOF.
Assume that G^Δ contains a cycle

$$P = x_1, x_2, \ldots, x_n$$

where $x_1 = x_n$ such that $w(P) \preceq (0, <)$. Let $w(x_i, x_{i+1}) = (m_i, \lhd_i)$ for all $1 \leq i < n$. By the construction of G^Δ there exists a constraint in Δ for each edge thus $\{x_{i+1} - x_i \lhd_i m_i \mid 1 \leq i < n\} \subseteq \Delta$. Let $\Lambda = \{\sum_{i=1}^{n-1}(x_{i+1} - x_i \lhd_i m_i)\} \cup \Delta$. By Lemma 6 $[\Lambda] = [\Delta]$, hence proving $[\Lambda] = \emptyset$ will show Δ is inconsistent.

As $w(P) \preceq (0, <)$ we have two cases: Either $w(P) = (0, <)$ or $w(P) = (j, \lhd)$ for some $j < 0$.

- Assume $w(P) = (j, \lhd)$ for some $j < 0$
  In this case $\sum_{i=1}^{n-1}(x_{i+1} - x_i \lhd_i m_i) = (x_n - x_1 = j < 0)$. Since $x_n = x_1$, no valuation satisfies this constraint hence $[\Lambda] = \emptyset$.
- Assume $w(P) = (0, <)$
  As $w(P) = (0, <)$, there exists an $1 \leq i < n$ such that $\lhd_i = <$. Therefore $\sum_{i=1}^{n-1}(x_{i+1} - x_i \lhd_i m_i) = (x_n - x_1 < 0)$. Since $x_n = x_1$, this constraint can never be satisfied thus $\Lambda = \emptyset$. ∎

We now turn our attention towards finding a solution to a set of difference constraints. To this end, we use an algorithm that originates from the UPPAAL DBM library [21]. To the best of our knowledge this algorithm has not been proven correct.
We demonstrate the intuition of the algorithm by example.

Example 21
Let $\Delta = \{y - x \leq 5, x - y \leq -5, 0 - y \leq 0, y - 0 \leq 13, x - 0 \leq 8, 0 - x \leq -1\}$. We create a valuation v such that $v \vDash \Delta$. Note that Δ is in closed form. First we assign a value to y. This value must satisfy $0 \leq y \leq 13$. We let $v(y) = 10$. Obviously this restricts the possible values of x. We can convert the constraints involving x by substituting y with 10 and get

$$\begin{aligned} 10 - x \;&\leq\; 5 \;\Leftrightarrow\; 0 - x \;\leq\; -5, \text{ and} \\ x - 10 \;&\leq\; 5 \;\Leftrightarrow\; x - 0 \;\leq\; 15. \end{aligned}$$

*

By adding these constraints to Δ the possible values for x are tightened to $5 \leq x \leq 8$. We are are free to choose any value in this range for x, hence choosing 6 forms the valuation $x = 6$ and $y = 10$ satisfying Δ.

This intuition is formalised in Algorithm 2. The algorithm is, as Example 21 hints towards, iteratively choosing a value for each variable such that all constraints involving that variable and any previously set variable are satisfied. In Algorithm 2, we write $v[x \mapsto r]$ to indicate that v is updated such that $v(x) = r$. For ease of notation, we write $(n, \lhd)_{x,y}$ instead of $x - y \lhd n$.

Algorithm 2: Finding a valuation v that satisfies all constraints in a DBM H.

Input: A DBM H in closed form over the variables $\mathcal{C}$.
Output: A valuation v or "Inconsistent".

if $H_{00} \preceq (0, <)$ **then**
  return *Inconsistent*
end
$R = \{\mathbf{0}\}$;
$v[\mathbf{0} \mapsto 0]$;
forall $x \in (\mathcal{C} \setminus \{\mathbf{0}\})$ **do**
  $upper = H_{x,\mathbf{0}}$;
  $lower = H_{\mathbf{0},x}$;
  forall $r \in R$ **do**
    $upper = min(upper, H_{x,r} + (v(r), \leq)_{r,\mathbf{0}})$;
    $lower = min(lower, H_{r,x} + (-v(r), \leq)_{\mathbf{0},r})$;
  end
  if *there exists a value s that satisfies $lower$ and $upper$ and $H_{xx} \not\preceq (0, <)$* **then**
    choose value s satisfying both $upper$ and $lower$;
    $v[x \mapsto s]$;
    $R = R \cup \{x\}$;
  end
  else
    return *Inconsistent*
  end
end
return v

Algorithm 2 always terminates as it contains only for-all loops. We now wish to prove that the algorithm is correct. To this end, we first prove that if Algorithm 2 returns a valuation, then it is a solution to the DBM.

Lemma 22 (Soundness)
Let H be a DBM in closed form. If Algorithm 2 returns v then $v \in [H]$.

PROOF.
We use the following loop invariant:

- For each pair of variables $y, z \in R : v \vDash H_{y,z}$.

Initialisation Before the first iteration of the loop $R = \{\mathbf{0}\}$, hence we must show that $v \vDash H_{\mathbf{00}}$. Since v was returned, it follows that by line 1 that $H_{00} \not\preceq (0, <)$, hence $v[\mathbf{0} \mapsto 0] \vDash H_{\mathbf{00}}$.

Maintenance Let v' be v from the algorithm before line 15 and let v'' be v after the update in 15. Likewise, let R' be R from the algorithm before line 16 and let $R'' = R' \cup \{x\}$, hence reflecting the update in line 16. Assume the loop invariant holds for v' and R'. We show it also holds for v'' and R''.
By the loop invariant, $v' \vDash H_{yz}$ for all $y, z \in R'$. As $v''(y) = v'(y)$ for all $y \in R$ we need only show that $v'' \vDash H_{xx}$ and for all $y \in R'$, $v'' \vDash H_{xy}$ and $v'' \vDash H_{yx}$.

As v was returned it follows that $H_{xx} \not\preceq (0, <)$, hence any value of x satisfies H_{xx}. In line 14 a value s is chosen such that v'' satifies both $upper$ and $lower$. Due to transitivity of $\preceq$ it follows that

$$\begin{aligned} v'' &\vDash H_{x,\mathbf{0}}, \\ v'' &\vDash H_{\mathbf{0},x}, \\ v'' &\vDash (-v'(z), \leq)_{\mathbf{0},z} + H_{z,x} \text{ for all } z \text{ in } R', \text{ and} \\ v'' &\vDash (v'(z), \leq)_{z,\mathbf{0}} + H_{x,z} \text{ for all } z \text{ in } R'. \end{aligned}$$

Let z be any variable in R' and let $(n, \lhd)_{z,x} = H_{z,x}$. We know that

$$v'' \vDash (-v'(z), \leq)_{\mathbf{0},z} + H_{z,x} = (n - v'(z), \lhd)_{\mathbf{0},x},$$

which implies

$$v''(\mathbf{0}) - v''(x) \lhd n - v'(z) \Rightarrow -v''(x) \lhd n - v'(z) \Rightarrow v'(z) - v''(x) \lhd n \Rightarrow v''(z) - v''(x) \lhd n.$$

We have shown that v'' satisfies all constraints on the form $z - x \lhd n$ for $z \in R'$. A similar proof may be made for constraints on the form $x - z \lhd n$. It follows trivially that $v'' \vDash H_{xx}$, as $H_{xx} \not\preceq (0, <)$. Having this in mind we have proven that all constraints involving x are satisfied at the end of an iteration of the loop, hence the invariant is satisfied.

Termination At termination $R = \mathcal{C}$ and by the loop invariant all constraints are satisfied hence $v \in [H]$. ∎

The next step in proving the correctness of Algorithm 2 is to show that it can find any solution to the DBM.

Lemma 23 (Completeness)
Let H be a DBM in closed form. If $v \in [H]$, then there exist a computational branch of Algorithm 2 run on H such that it returns v.

PROOF.
Assume $v \in [H]$. We prove that Algorithm 2 can find a v' such that for all $x \in \mathcal{C} : v(x) = v'(x)$. We use the following loop invariant:

- $\forall x \in R : v(x) = v'(x)$

Initialisation First we argue that the if-statement in line 1 is not satisfied. By Lemma 20, if H is consistent, then H does not contain a negative weight cycle. Since $v \in [H]$, there are no negative cycles in H, hence the if-statement in line 1 is not satisfied. The next step Algorithm 2 performs is setting $v'(\mathbf{0}) = 0$. As $v \in [H]$ we know $v(\mathbf{0}) = 0$ hence $v(\mathbf{0}) = v'(\mathbf{0})$

Maintenance Upon entering the loop, we assume that the invariant holds, hence $\forall x \in R : v(x) = v'(x)$. To argue we can set $v'[x \mapsto v(x)]$ we need only argue that

- $H_{xx} \not\leq (0, <)$
  This follows from Lemma 20 and $v \in [H]$.

- $v \vDash upper$ and $v \vDash lower$, hence show that v satisfies

  $$H_{x,\mathbf{0}},$$
  $$H_{\mathbf{0},x},$$
  $$(-v(z), \leq)_{\mathbf{0},z} + H_{z,x} \text{ for all } z \in R, \text{ and}$$
  $$(v(z), \leq)_{z,\mathbf{0}} + H_{x,z} \text{ for all } x \in R.$$

  The first two follow trivially since $v \vDash H$. Assume for some z that $v \nvDash (-v(z), \leq)_{\mathbf{0},z} + H_{z,x}$. Let $H_{z,x} = (n_1, \lhd_1)_{z,x}$. As v is a solution we know that

  $$v(z) - v(x) \lhd_1 n_1$$

  At the same time we have assumed that

  $$v(\mathbf{0}) - v(x) \not\lhd_1 n_1 - v(z) \Rightarrow v(z) - v(x) \not\lhd_1 n_1$$

  thus we clearly have a contradiction leading to the conclusion that $v \vDash (-v(z), \leq)_{\mathbf{0},z} + H_{z,x}$. A similar proof can be made to show that $v \vDash (v(x), \leq)_{x,\mathbf{0}} + H_{z,x}$

As v satisfies both upper and lower setting $v'[x \mapsto v(x)]$ satisfies all constraints involving variables in R.

Termination At termination $R = \mathcal{C}$. By the loop invariant, for all $x \in R : v(x) = v'(x)$ thus there exists a computational branch of Algorithm 2 that returns v. ∎

Lemma 24
Let H be a DBM in closed form such that H does not contain a negative weight cycle. For any computational branch, Algorithm 2 run on H returns a valuation.

PROOF.
Assume that there exists a computational branch that returns inconsistent. We exclude the possibilities of returning inconsistent. The if-statement in line 1 is never satisfied as H does not contain a negative weight cycle. Assume towards reaching a contradiction that the algorithm

returns inconsistent due to failing to satisfy the if-statement in line 13. It is not the case that $H_{xx} \preceq (0, <)$, as H does not contain a negative weight cycle, hence we must have

$$upper = (u, \lhd_u)_{x,\mathbf{0}}, u \in \mathbb{R}_{\geq 0} \text{ and}$$
$$lower = (-l, \lhd_l)_{\mathbf{0},x}, l \in \mathbb{R}_{\geq 0}$$

such that either $l > u$ or $u = l$ and one of $\lhd_u$ or $\lhd_l$ is strict. In either case $lower + upper$ forms a negative cycle. $upper$ and $lower$ are either in H or derived from a constraint and a value already set.

- Assume both $upper$ and $lower$ are taken from H, i.e. $upper = H_{x,\mathbf{0}}$ and $lower = H_{\mathbf{0},x}$. This cannot be the case as $upper + lower$ forms a negative weight cycle and H does not contain any negative weight cycle.
- Assume $upper = H_{x,z} + (v(z), \leq)_{x,\mathbf{0}}$ and $lower = H_{y,x} + (-v(y), \leq)_{\mathbf{0},y}$ where $z, y \in R$ and let $H_{x,z} = (n, \lhd_1)_{x,z}$ and $H_{y,x} = (-m, \lhd_2)_{y,x}$. By the loop invariant of Lemma 22 v satisfies all constraints involving variables from R. In particular $v \vDash H_{y,z}$. Due to closed form of H we know that $H_{y,z} \preceq H_{x,z} + H_{y,x}$ thus $v \vDash H_{x,z} + H_{y,x} \Rightarrow v(y) - v(z) \lhd_2 \lhd_1 n - m$.

  We divide the remaining of this case into two

  - $m + v(y) > n + v(z) \Rightarrow v(y) - v(z) > n - m$
    We see that $v(y) - v(z) \lhd_2 \lhd_1 n - m < v(y) - v(z)$ thus clearly a contradiction.
  - $m + v(y) = n + v(z)$ and $< \in \{\lhd_1, \lhd_2\}$
    We rewrite and obtain $v(y) - v(z) = n - m$. As v is a solution, we know that $v(y) - v(z) \lhd_2 \lhd_1 n - m$. As $< \in \{\lhd_1, \lhd_2\}$ we conclude that $v(y) - v(z) < n - m$. Combining these equations gives

    $$v(y) - v(z) < n - m = v(y) - v(z)$$

    thus a contradition has been reached.
- The cases in which either $upper$ or $lower$ (not both) is fetched from H are merely special cases of the above.

We have proven that it is never the case that we return inconsistent, hence we always return a valuation. ∎

We allow calling Algorithm 2 with a set of difference constraints or a constraint graph, and the results obtained for the algorithm and a DBM hold for these representations as well. We use this in the following theorem regarding the connection between inconsistency and negative weight cycles. This theorem is related to the work of Bellman [6], but is considered a common result.

Theorem 25
A set of difference constraints Δ is inconsistent iff G^{Δ} contains a negative weight cycle.

PROOF.
We divide the proof into the two directions.

- inconsistency $\Leftarrow$ negative cycle
  This direction is already proven in Lemma 20
- inconsistency $\Rightarrow$ negative cycle
  Proof by contraposition. Assume G^{Δ} does not contain a negative weight cycle. We prove that Δ is consistent.
  Since G^{Δ} does not contain a negative weight cycle, by Lemma 19 we can without loss of generality assume that G^{Δ} is in closed form and that $[\Delta] = [G^{\Delta}]$. By Lemma 24, Algorithm 2 run on G^{Δ} returns a valuation v. By Lemma 22 $v \in [G^{\Delta}]$, hence G^{Δ} is consistent. Since $[G^{\Delta}] = [\Delta]$, Δ is consistent as well. This completes this direction.

We use this result to state the main theorem of this chapter.

Theorem 26
Let H be a DBM in closed form. If

a) H is inconsistent, then Algorithm 2 run on H terminates and returns "Inconsistent" independent of the nondeterministic choices made, and

b) there exists $v \in [H]$, then Algorithm 2 run on H terminates and there exists a computation which returns v and it never returns "Inconsistent". $\diamondsuit$

PROOF.

We show that both a) and b) are true.

a) Assume H is inconsistent. Algorithm 2 terminates, thus the algorithm must return either a valuation v' or "Inconsistent".
   Assume towards reaching a contradiction that the algorithm returns a valuation v'. By Lemma 22 v' is a solution, hence contradicting that H is inconsistent.

b) Assume there exists a valuation $v \in [H]$. By Lemma 23 there exists a computational branch that returns v. Since $v \in [H]$, H is consistent and by Theorem 25 H contains no negative cycle. By Lemma 24, Algorithm 2 always returns a valuation and never "Inconsistent". ■

Summary In this chapter, we have introduced three representations for zones. Furthermore, we have shown the relation between inconsistency and negative weight cycles. Algorithm 2 introduced in this chapter can be used to find valuations in a consistent zone. From this point forward, we refer to Algorithm 2 by the name FINDPOINT.

Chapter 4

Symbolic Model Checking

In this chapter we consider a different semantics for Timed Automata called symbolic semantics. This is used in model checking tools due to its efficiency, but is also the cause of the problem we attempt to solve in this thesis. Henzinger et al. [16] originally introduced the symbolic semantics, using a different terminology.
After having introduced the symbolic semantics we prove that the semantics can be effectively represented and computed using difference constraints.

4.1 Symbolic Semantics

The state space of a Timed Automaton is possibly infinite since there exists infinitely many delays between two integers in every location. Model checking tools use an abstraction to overcome this problem. Instead of considering only one concrete state at a time, they consider a set of concrete states.

Example 27
Consider the Timed Automaton in Figure 4.1. We start in ℓ_0 with only the initial valuation i.e. $x = 0$. The set of valuations that can be reached by performing a delay is the set $x \geq 0$. Obviously there exists valuations in this set that satisfy the guard on the edge towards ℓ_1. The guard states that only the states where $x \geq 5$ can take the edge. We consider the aforementioned set and determine that the set of valuations where $x \geq 5$ satisfies the guard. The invariant in ℓ_1 states that $x \leq 15$ hence we restrict the set to $5 \leq x \leq 15$.
Consider which valuations are reachable by delaying. As the invariant $x \leq 15$ still applies to ℓ_1 this restricts the possible set of valuation to those where $x \leq 15$, thus the reachable set remains $5 \leq x \leq 15$.
From here we take the edge to ℓ_2. This transition has no guards, thus every value of x is accepted. Since x is reset, we reach ℓ_2 while $x = 0$. We again consider delaying and which results in the set $x \geq 0$.
The conclusion of the exploration is that we can reach the following set of states of the Timed Automaton

$$\begin{aligned} &\{\langle \ell_0, v\rangle \mid 0 \leq v(x)\} \\ \cup\, &\{\langle \ell_1, v\rangle \mid 5 \leq v(x) \leq 15\} \\ \cup\, &\{\langle \ell_2, v\rangle \mid 0 \leq v(x)\} \end{aligned}$$

$*$

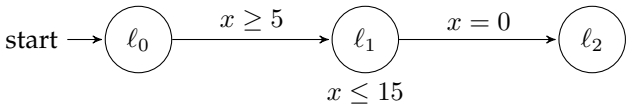


Figure 4.1: A Timed Automaton.

The exploration of the state space performed in Example 27 is known as a symbolic exploration. In order to perform this exploration we need two new operations on sets of valuations: A symbolic delay operator ($^{\uparrow}$) and a symbolic reset operator ($_{r=0}$). The result of applying the delay operator is the set of all valuations that can be obtained by delaying from some valuation in the original set. The reset operator results in a set of valuations where each valuation has been reset by r.

Definition 28
Let Π be a set of valuations over $\mathcal{C}$.

We define the unary operator $^{\uparrow}$ as:

$$\Pi^{\uparrow} \stackrel{def}{=} \{v + d \mid v \in \Pi \wedge d \in \mathbb{R}_{\geq 0}\},$$

and for $r \subseteq \mathcal{C}$ where $\mathbf{0} \notin r$, we define the operator $_{r=0}$ as:

$$\Pi_{r=0} \stackrel{def}{=} \{v_{r=0} \mid v \in \Pi\} \qquad \diamondsuit$$

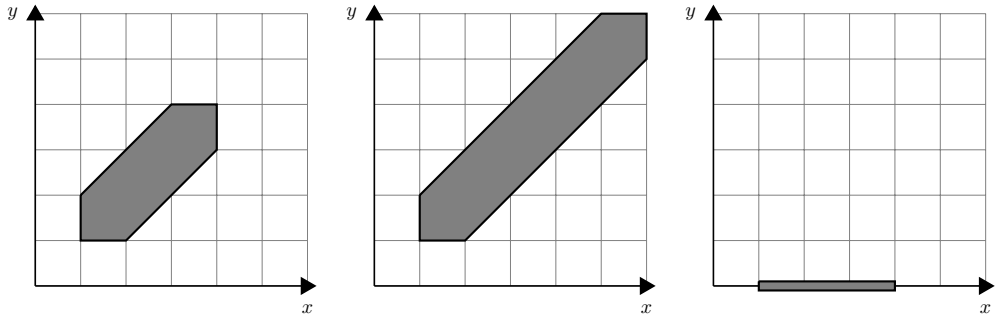
An example of applying the operations is shown in Figure 4.2.

Using the operators $^{\uparrow}$, $_{r=0}$ and $\cap$ we define a transition system over a Timed Automaton in which a symbolic state consists of a location and a set of valuations that satisfy the invariant.

Definition 29 (Symbolic Semantics of Timed Automata)
The symbolic semantics of a Timed Automaton $(\mathcal{L}, \ell_0, \mathcal{C}, E, \mathtt{I})$ is a transition system $(S, s_0, Lab, \rightsquigarrow)$ where

- $S = \{\langle \ell, \Pi \rangle \mid \ell \in \mathcal{L}, \forall v \in \Pi : v \vDash \mathtt{I}(\ell)\}$ is a set of states,
- $s_0 = \langle \ell_0, \{v_0\} \rangle \in S$ is the initial state, and
- $Lab = DC^{\mathbb{Z}}(\mathcal{C}) \times 2^{\mathcal{C}}$, and
- $\rightsquigarrow$ is the transition relation defined as
  - $\langle \ell, \Pi \rangle \rightsquigarrow \langle \ell, \Pi^{\uparrow} \cap [\mathtt{I}(\ell)] \rangle$
  - $\langle \ell, \Pi \rangle \stackrel{g,r}{\rightsquigarrow} \langle \ell', (\Pi \cap [g])_{r=0} \cap [\mathtt{I}(\ell')] \rangle$ if $\ell \xrightarrow{g,r} \ell'$ $\qquad \diamondsuit$

We write $\rightsquigarrow^*$ to denote taking 0 or more transitions by the symbolic transition system and $\rightarrow^*$ to denote taking 0 or more transitions in the timed transition system. When a state's successors are computed using the transition relation $\rightsquigarrow$, we call it a forwards exploration. A symbolic



(a) D (b) $D^{\uparrow}$ (c) $D_{\{y\}=0}$

Figure 4.2: Operations on sets of valuations. (a) is a set of valuations. In (b) the $^{\uparrow}$ operator has been applied and in (c) the $_{r=0}$ operator where $r = \{y\}$ has been applied.

state is reachable if it can be reached through a forwards exploration from the initial state. The following result regarding the symbolic semantics stated in [9, 13] connects the symbolic semantics and the concrete semantics with respect to reachability:

Theorem 30 (Bengtsson and Yi [9])
Let $(\mathcal{L}, \ell_0, \mathcal{C}, E, \mathtt{I})$ be a timed automaton with initial state $\langle \ell_0, v_0 \rangle$. Then

(*soundness*) $\langle \ell_0, \{v_0\} \rangle \rightsquigarrow^* \langle \ell, \Pi \rangle$ implies $\langle \ell_0, v_0 \rangle \rightarrow^* \langle \ell, v \rangle$ for all $v \in \Pi$.

(*completeness*) $\langle \ell_0, v_0 \rangle \rightarrow^* \langle \ell, v \rangle$ implies $\langle \ell_0, \{v_0\} \rangle \rightsquigarrow^* \langle \ell, \Pi \rangle$ and for some Π such that $v \in \Pi$. $\diamondsuit$

It is possible to consider a state in the concrete semantics of a Timed Automaton as an element in a symbolic state. The symbolic semantics is sound in the sense that any symbolic state reachable in the symbolic transition system implies that all elements in it are reachable in the concrete semantics. Completeness states that any state reachable by the concrete semantics is a member of some symbolic state that is reachable.

4.2 Zones and Operations

In the preceding chapter we introduced zones and different representations of these. We let $\mathcal{D}^{\mathcal{C}}$ denote the set of all sets of valuation that can be represented by difference constraints over the variables $\mathcal{C}$ i.e. all zones over $\mathcal{C}$. Formally

$$\mathcal{D}^{\mathcal{C}} \stackrel{def}{=} \{[\Lambda] \mid \Lambda \subseteq DC(\mathcal{C})\}.$$

In the following, we show that zones are closed under the operations $^{\uparrow}$, $_{r=0}$ and $\cap$. The results in this section are common knowledge - for an overview see [7, 9]. To the best of our knowledge zones closure under $^{\uparrow}$ and $_{r=0}$ has never proven when considering strict constraints. In the following, we prove that this is the case. Our proofs relate to the work of Bengtsson and Larsson [10], who proves these constructions correct when only non-strict constraints are allowed. Finally we use these results to show all the sets of valuations encountered during symbolic state space exploration are zones.

Lemma 31
Let $\Pi \in \mathcal{D}^{\mathcal{C}}$. Then $\Pi^{\uparrow} \in \mathcal{D}^{\mathcal{C}}$. $\diamondsuit$

PROOF.
We consider the cases when Π is empty and non empty:

Case $\Pi = \emptyset$.
Since $\Pi = \emptyset$, Δ must be inconsistent and $\Pi^{\uparrow} = \emptyset$. Let $\Delta^{\emptyset}$ be a set of difference constraints that is inconsistent. Then $\Pi^{\uparrow} = [\Delta^{\emptyset}]$ and $\Pi^{\uparrow} \in \mathcal{D}^{\mathcal{C}}$.

Case $\Pi \neq \emptyset$.
Since $\Pi \in \mathcal{D}^{\mathcal{C}}$, there exists a consistent set of difference constraints Δ such that $[\Delta] = \Pi$. Let $\Lambda = \{x - y \lhd n \mid x - y \lhd n \in \Delta^c \wedge y \neq \mathbf{0}\}$. Our goal is to prove that $[\Delta^c]^{\uparrow} = [\Lambda]$.

Direction $[\Delta^c]^{\uparrow} \subseteq [\Lambda]$.
Let v' be an assignment in $[\Delta^c]^{\uparrow}$. By definition, $v' = v + d$ for some v in $[\Delta^c]$ and $d \in \mathbb{R}_{\geq 0}$. We show that $v' \in [\Lambda]$. To this end, we consider the constraints in Λ and prove that each of them is satisfied by v'. Consider the following forms:

- $x - y \lhd n$, where $x \neq \mathbf{0} \wedge y \neq \mathbf{0}$
  Since $v' = v + d$, we have

  $$v'(x) - v'(y) = (v(x) + d) - (v(y) + d) = v(x) - v(y) \lhd n,$$

  hence all constraints on this form are satisfied by v'.

- $\mathbf{0} - x \lhd -L_x$, where $x \neq \mathbf{0}$
  Since $v' = v + d$, we have

  $$v'(\mathbf{0}) - v'(x) = v(\mathbf{0}) - (v(x) + d) \ \leq \ v(\mathbf{0}) - v(x) \lhd -L_x,$$

  hence all constraints on this form are satisfied by v'.

This completes this direction of the proof.

Direction $[\Lambda] \subseteq [\Delta^c]^{\uparrow}$.
Let $v' \in [\Lambda]$. We prove that there exists a $v \in \Delta^c$ and $d \in \mathbb{R}_{\geq 0}$ such that $v' = v + d$. To this end we let

$$d_u = max\{v'(x) - u_x \mid (x - \mathbf{0} \lhd u_x) \in \Delta^c\}, \text{ and}$$
$$d_l = min\{v'(x) - l_x \mid (\mathbf{0} - x \lhd -l_x) \in \Delta^c\}.$$

First observe that $d_l \geq 0$ always holds. If not, there exists an $x \in \mathcal{C}$ such that

$$v'(x) - l_x < 0 \Rightarrow -l_x < -v'(x),$$

which implies v' is not a solution to Λ, thus contradicting that $v' \in [\Lambda]$. Next consider that if $d_u < 0$ it holds for all clocks that

$$v'(x) - u_x < 0 \Rightarrow v'(x) < u_x,$$

then v' satisfies all upper bound, and it can be shown that $v' \in \Delta$.

We proceed in three directions depending on the relation between d_u and d_l. See Figure 4.3 for examples. Note that some directions have been ommitted due to the previous arguments.

- $0 \leq d_u < d_l$
  This case is depicted in Figure 4.3(a). Let d be any value such that $d_u < d < d_l$ and for all $x \in \mathcal{C} \setminus \{\mathbf{0}\}$ let $v(x) = v'(x) - d$. Clearly $v' = v + d$ thus we need only prove that $v \in [\Delta^c]$. We continue by analysing all constraints in Δ.

  - $x - y \lhd n$ where $x \neq \mathbf{0}$ and $y \neq \mathbf{0}$
    First notice that these constraint are all present in Λ and therefore v' satisfies them. Let $x - y \lhd n$ be any constraints on this form in Δ^c then

    $$v(x) - v(y) = v(x) + d - (v(y) + d) = v'(x) - v'(y) \lhd n.$$

  - $\mathbf{0} - y \lhd -l_y$ where $y \neq \mathbf{0}$
    These constraints are all contained in Λ thus v' satisfies the
  - $x - \mathbf{0} < \infty$, where $x \neq \mathbf{0}$
    All constraints on this form are trivially satisfied, hence v' satisfies them.se constraints. Let $\mathbf{0} - y \lhd -l_y$ be any constraint on this form in Δ^c and recall that $d < d_l \leq v'(y) - l_y$ then

    $$v(\mathbf{0}) - v(y) = -(v'(y) - d) = -v'(y) + d < -v'(y) + v'(y) - l_y = -l_y.$$

  - $x - \mathbf{0} \lhd u_x$ where $x \neq \mathbf{0}$

    Let $x - \mathbf{0} \lhd u_x$ be any constraint on this form and recall that $v'(x) - u_x \leq d_u < d$ then

    $$v(x) - v(\mathbf{0}) = v'(x) - d < v'(x) - (v'(x) - u_x) = u_x.$$

- $0 \leq d_u = d_l$
  This case is depicted in Figure 4.3(b).

  Let x and y be the clocks such that $d_u = v'(x) - u_x$ and $d_l = v'(y) - l_y$. Then

  $$v'(x) - u_x = v'(y) - l_y \Rightarrow v'(x) - v'(y) = u_x - ly.$$

  Since Δ^c is in closed form there exists a constraint $(x - y \lhd n) \in \Delta^c$ such that $n \leq u_x - l_y$. This constraint is also present in Λ hence

  $$v'(x) - v'(y) = u_x - l_y \leq n.$$

  It follows that

  $$n \leq u_x - l_y \leq n \Rightarrow n = u_x - l_y.$$

The above derivations imply that there must exist a constraint

$$\delta = (x - y \lhd u_x - l_y),$$

in Λ. We wish to derive that $\lhd = \leq$. Because v' satisfies all the constraints in Λ the following holds

$$\begin{aligned} v'(x) - v'(y) = u_x - l_y \lhd n = u_x - l_y \\ \Rightarrow u_x - l_y \lhd u_x - l_y. \end{aligned}$$

The above can only be true if $\lhd = \leq$ hence $\delta = (x - y \leq u_x - l_y)$. It can be shown that $\lhd = \leq$ implies that the upper and lower bounds on x and y are also non-strict. Let $d = d_u$, then both upper on x and lower bound on y are satisfied. Arguing that the remaining constraints are satisfied is done using a strategy similar to the one from the case $0 \leq d_u < d_l$.

- $0 \leq d_l < d_u$
  This case is depicted in Figure 4.3(c).
  Let $v'(x) - u_x = d_u$ and $v'(y) - l_y = d_l$ then

$$v'(y) - l_y < v'(x) - u_x \Rightarrow u_x - l_x < v'(x) - v'(y).$$

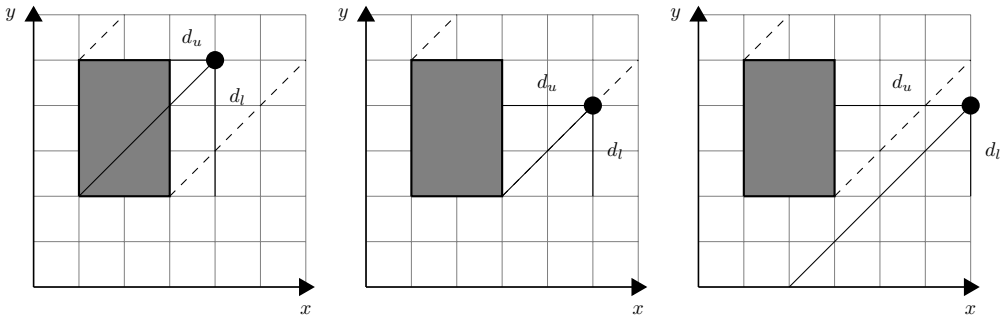
From the fact that Δ^c is in closed form there exists a constraint $(x - y \lhd n)$ such that $n \leq u_x - l_y$. This constraint is also in Λ hence

$$v'(x) - v(y) \leq n \leq u_x - l_y.$$

We combine and obtain

$$u_x - l_x < v'(x) - v'(y) \leq u_x - l_y,$$

hence we have reached a contradiction and conclude it can never be the case that $d_l < d_u$.■



(a) $0 \leq d_u < d_l$ (b) $0 \leq d_u = d_l$ (c) $0 \leq d_l < d_u$

Figure 4.3: The cases used in Lemma 31. The circle in each figure denotes a valuation and the area between the dashed lines denotes the upped zoned.

We continue by showing that the elements in $\mathcal{D}^{\mathcal{C}}$ are closed under the reset operation:

Lemma 32
Let $\Pi \in \mathcal{D}^{\mathcal{C}}$ be a set of valuations over the variables $\mathcal{C}$. Then $\Pi_{r=0} \in \mathcal{D}^{\mathcal{C}}$ for $r \subseteq \mathcal{C}$. $\diamondsuit$

PROOF.
We consider the cases when Π is empty and non empty:

Case $\Pi = \emptyset$
Since $\Pi = \emptyset$ it follows that $\Pi_{r=0} = \emptyset$. Let $\Delta^{\emptyset}$ be a set of difference constraints that is inconsistent. Then $\Pi_{r=0} = [\Delta^{\emptyset}]$ thus $\Pi_{r=0} \in \mathcal{D}^{\mathcal{C}}$.

Case $\Pi \neq \emptyset$
Since $\Pi \in \mathcal{D}^{\mathcal{C}}$, there exists a consistent set of difference constraints Δ such that $[\Delta] = \Pi$. Let

$$\Lambda = \{x - y \lhd n \mid x - y \lhd n \in \Delta^c \wedge x \notin r \wedge y \notin r\} \cup \{\mathbf{0} - x \leq 0 \mid x \in r\} \cup \{x - \mathbf{0} \leq 0 \mid x \in r\}.$$

Our goal is to prove that $[\Delta^c]_{r=0} = [\Lambda]$.

Direction $[\Delta^c]_{r=0} \subseteq [\Lambda]$
Let $v' \in [\Delta^c]_{r=0}$. Then there exists a solution $v \in [\Delta^c]$ such that $v' = v_{r=0}$. We show that $v' \in [\Lambda]$. We consider the constraints in the construction of Λ:

- $x - y \lhd n$ where $(x - y \lhd n) \in \Delta^c$, $x \notin r$ and $y \notin r$
  Since $v'(x) = v(x)$ for $x \notin r$ we have
  $$v'(x) - v'(y) = v(x) - v(y) \lhd n.$$
- $x - \mathbf{0} \leq 0$ where $x \in r$
  Since $v'(x) = v_{r=0}(x)$ for all $x \in r$, we have
  $$v'(x) - v'(\mathbf{0}) = v_{r=0}(x) - v(\mathbf{0}) = 0 \leq 0.$$
- $\mathbf{0} - x \leq 0$ where $x \in r$
  Using the same reasoning as in the case above, we have
  $$v'(\mathbf{0}) - v'(x) = v(\mathbf{0}) - v_{r=0}(x) = 0 \leq 0.$$

Direction $[\Lambda] \subseteq [\Delta^c]_{r=0}$**.**
Let $v' \in [\Lambda]$. We prove that $v' \in [\Delta^c]_{r=0}$ by showing that there exists a $v \in [\Delta^c]$ such that $v' = v_{r=0}$.
We run the FINDPOINT algorithm on Δ^c using the following strategy to make the nondeterministic choices:

- In the first $|\mathcal{C} \setminus (r \cup \mathbf{0})|$ iterations of the loop in line 6, choose $x \in \mathcal{C} \setminus (r \cup \mathbf{0})$.
- In line 14, choose the value $v'(x)$.

Choosing these values is possible. To understand why, consider that only the following constraints are considered for this part of FINDPOINTs computation

$$\Delta' = \{x - y \lhd n \mid x \notin r \wedge y \notin r \wedge (x - y \lhd n) \in \Lambda\}.$$

Note that $\Delta' \subseteq \Delta^c$. Since $v' \in [\Lambda]$, these constraints are satisfied by the values chosen.

In the remaining iterations use any strategy. Since Δ^c is consistent, by Theorem 26, FINDPOINT returns a valuation v and by Lemma 22, $v \in [\Delta^c]$. Since for all $x \in (\mathcal{C} \setminus r) : v(x) = v'(x)$, surely $v_{r=0} = v'$.

This completes the proof. ∎

The last operator we consider is conjunction.

Lemma 33
Let $\Pi, \Pi' \in \mathcal{D}^{\mathcal{C}}$. Then $\Pi \cap \Pi' \in \mathcal{D}^{\mathcal{C}}$. ♢

PROOF.
We consider the cases when Π and Π' are empty and non empty:

Case $\Pi = \emptyset$ or $\Pi' = \emptyset$
We have that $\Pi \cap \Pi' = \emptyset$. Let Δ be an inconsistent set of difference constraints. Then $[\Delta] = \Pi \cap \Pi'$, and $\Pi \cap \Pi' \in \mathcal{D}^{\mathcal{C}}$.

Case $\Pi \neq \emptyset$ and $\Pi' \neq \emptyset$
Since $\Pi, \Pi' \in \mathcal{D}^{\mathcal{C}}$, there exists sets of difference constraints Λ and Λ' such that $\Pi = [\Lambda]$ and $\Pi' = [\Lambda']$. Let $\Delta = \Lambda \cup \Lambda'$. We prove inclusion in both directions:

Direction $[\Delta] \subseteq \Pi \cap \Pi'$.
Assume that $v \in [\Delta]$. Then $v \vDash \Lambda \cup \Lambda'$ and this implies that $v \vDash \Lambda$ and $v \vDash \Lambda'$, thus $v \in \Pi \cap \Pi'$.

Direction $\Pi \cap \Pi' \subseteq [\Delta]$.
Assume $v \in \Pi \cap \Pi'$. Then $v \in \Pi = [\Lambda]$ and $v \in \Pi' = [\Lambda']$. Since v satisfies all constraints in Λ and Λ', it also satisfies $\Lambda \cup \Lambda'$, hence $v \in [\Delta]$.
This completes the proof. ∎

A result of the lemmas shown in this section is that if a forwards exploration starts in a zone, then all sets of valuations encountered during the forwards exploration are zones.

Corollary 34
Let $(\mathcal{L}, \ell_0, \mathcal{C}, E, \mathtt{I})$ be a Timed Automaton and let $\langle \ell, \Pi \rangle \rightsquigarrow^* \langle \ell', \Pi' \rangle$. If $\Pi \in \mathcal{D}^{\mathcal{C}}$ then $\Pi' \in \mathcal{D}^{\mathcal{C}}$. ♢

We show that $\{v_0\} \in \mathcal{D}^{\mathcal{C}}$ and then it follows from Corollary 34 that the symbolic transition system consist of zones only. We construct the set of difference constraints

$$\Delta = \{x - \mathbf{0} \leq 0 \mid x \in \mathcal{C}\} \cup \{\mathbf{0} - x \leq 0 \mid x \in \mathcal{C}\}.$$

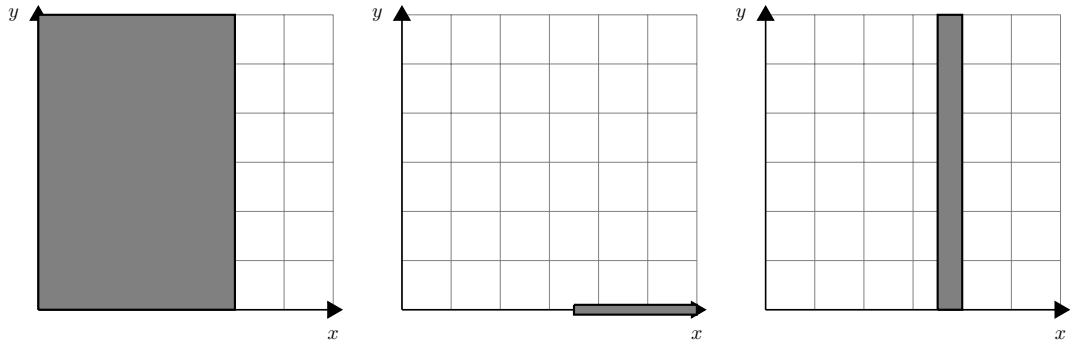
Clearly $[\Delta] = \{v_0\}$.

4.3 Backwards Exploration Operations

Instead of exploring the state space forwardly, it is sometimes useful to do the exploration in a backwards manner by computing predecessors. We present the backwards exploration operations and prove that zones are closed under these.
In the following we describe predecessor operators that are non-standard.

Action Predecessor

The action predecessor operation can be used to find all states that can reach a set of states. Consider the symbolic transition $\langle \ell_0, \Pi_0 \} \rangle \overset{x \geq 3, \{y\}}{\rightsquigarrow} \langle \ell_1, \Pi_1 \rangle$, where $\Pi_0 = \{v \mid v(x \geq 4\}$ and $\Pi_1 = \{v \mid v(x) \geq 3.5\}$. Now consider finding all the valuations in Π_0 which can reach a valuation in $\Pi' = \{v \mid v(x) \geq 3.5 \wedge v(y) = 0\} \subseteq \Pi_1$. Intuitively, it should be clear that this set is $\{v \mid 3.5 \leq v(x) \leq 4\}$. The sets Π_0, Π' are depicted in Figure 4.4.



(a) Π_0 (b) Π' (c) $pre_{action}(\Pi_0, x \geq 3, \{y\}, \Pi')$

Figure 4.4: Example of using the action predecessor operator. (c) depict all the valuations in (a) that can reach a valuation in (b) with respect to the guard $x \geq 3$ and reset of y.

Definition 35 (Action Predecessor)
Let Π, Π' be sets of valuations over the variables $\mathcal{C}$ and let $r \subseteq \mathcal{C}$. Furthermore let $g \in DC(\mathcal{C})$. We define

$$pre_{action}(\Pi, g, r, \Pi') \overset{def}{=} \{v \mid v \in \Pi \text{ s.t. } v \vDash g \text{ and } v_{r=0} \in \Pi'\}$$

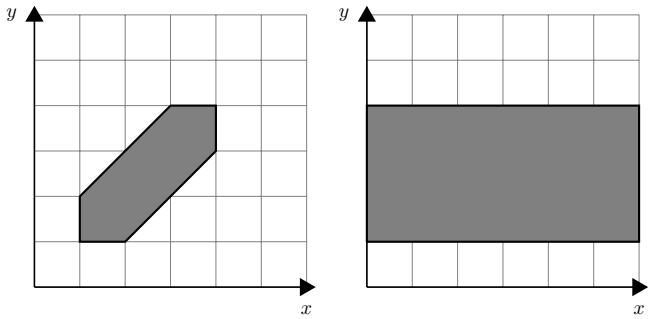
$\diamondsuit$

In order to implement the action predecessor we need some way to find all valuations that could potentially reach a set of valuations after a reset. To this end, letr *free* be an operator which intuitively frees a set of clocks, hereby allowing each freed clock to have any value. Note that free cannot be applied to **0**.

Definition 36 (Free)
Let $D \in \mathcal{D}^{\mathcal{C}}$. For any $r \subseteq \mathcal{C}$ such that $\mathbf{0} \notin r$ we define

$$free(D, r) \overset{def}{=} \{v \mid \exists v' \in D \text{ such that } \forall x \in (\mathcal{C} \setminus r) : v(x) = v'(x)\}.$$

An example of using *free* is shown in Figure 4.5.



(a) D (b) $free(D, \{x\})$

Figure 4.5: Applying free to a set of valuations. (a) depicts a zone D and (b) depicts $free(D, \{x\})$

Lemma 37
Let $\Pi \in \mathcal{D}^{\mathcal{C}}$ and $r \subseteq \mathcal{C}$ such that $\mathbf{0} \notin r$. Then $free(\Pi, r) \in \mathcal{D}^{\mathcal{C}}$. $\diamondsuit$

PROOF.
We consider the cases in which Π is empty and non-empty.

Case $\Pi = \emptyset$.
If $\Pi = \emptyset$ then $free(D, r) = \emptyset$ hence any inconsistent set of difference constraints represents $free(D, r)$.

Case $\Pi \neq \emptyset$.
Because $\Pi \in \mathcal{D}^{\mathcal{C}}$ there exists a set of difference constraints Δ such that $\Pi = [\Delta]$. We construct the set $\Lambda = \{x - y \lhd n \mid x - y \lhd n \in \Delta^c \wedge x \notin r \wedge y \notin r\}$.

We show that $free([\Delta^c], r) \subseteq [\Lambda]$ and $[\Lambda] \subseteq free([\Delta^c], r)$ and thereby that $[\Lambda] = free([\Delta^c], r)$.

Direction $free([\Delta^c], r) \subseteq [\Lambda]$
Let $v \in free([\Delta^c], r)$ then there exist a $v' \in \Delta^c$ such that $\forall x \in (\mathcal{C} \setminus r) : v(x) = v'(x)$. We continue by arguing that v satisfies all constraints in Λ

- $\{x - y \lhd n \mid x - y \lhd n \in \Delta^c \wedge x \notin r \wedge y \notin r\}$
  The constraints of this form are also present in Δ^c hence v' satisfies these constraints. Let $x - y \lhd n$ be a constraint from the set, then

  $$v(x) - v(y) = v'(x) - v'(y) \lhd n.$$

  We see that v satisfies the constraints.

Direction $[\Lambda] \subseteq free([\Delta^c], r)$
Let $v \in [\Lambda]$. To prove $v \in free([\Delta^c], r)$ we find a valuation $v' \in \Delta^c$ such that $v'_{r=0} = v$ and for all $x \in (\mathcal{C} \setminus r) : v(x) = v'(x)$. Consider running FINDPOINT with the following strategy:

- In the first $|\mathcal{C} \setminus (r \cup \mathbf{0})|$ iterations of the loop choose values for all clocks x in $\mathcal{C} \setminus (r \cup \mathbf{0})$.
- In line 14, choose the value $v'(x)$.

Proving this possible is a matter of using a strategy similar to the one used in Lemma 23. FINDPOINT will terminate and since Δ^c is consistent, it will return a solution to Δ^c. The result of FINDPOINT with the strategy is the valuation we needed. ■

We show that the action predecessor can be constructed using $free$ and intersection.

Lemma 38
Let Π_1, Π_2 be sets of valuation over the variables $\mathcal{C}$, $r \subseteq (\mathcal{C} \setminus \{\mathbf{0}\})$ and $g \in DC(\mathcal{C})$ such that $\Pi_2 \subseteq (\Pi_1 \cap [g])_{r=0}$ then $pre_{action}(\Pi_1, g, r, \Pi_2) = free(\Pi_2, r) \cap [g] \cap \Pi_1$. ♢

PROOF.
We show two inclusions.

Direction $pre_{action}(\Pi_1, g, r, \Pi_2) \subseteq free(\Pi_2, r) \cap g[\cap]\Pi_1$
Assume $v \in pre_{action}(\Pi_1, g, r, \Pi_2)$ then showing $v \in free(\Pi_2, r) \cap [g] \cap \Pi_1$ proves this direction. By definition of the action predecessor there exists a $v' \in \Pi_1$ such that $v'_{r=0} = v$ thus clearly $v \in free(\Pi_2, r)$. Furthermore $v \vDash g$ hence $v \in [g]$. Additionally we derive from definition of action predecessor that $v \in \Pi_1$. As $v \in free(\Pi_2, r)$, $v \in [g]$ and $v \in \Pi_1$ then $v \in free(\Pi_2, r) \cap [g] \cap \Pi_1$ thus proving this direction.

Direction $free(\Pi_2, r) \cap [g] \cap \Pi_1 \subseteq pre_{action}(\Pi_1, g, r, \Pi_2)$
Assume $v \in free(\Pi_2, r) \cap [g] \cap \Pi_1$ thus v is contained in $free(\Pi_2, r)$, $[g]$ and Π_1. Clearly $v \in \{v' \mid v' \in \Pi_1 \text{ s.t. } v' \vDash g \text{ and } v'_{r=0} \in \Pi_2\} = pre_{action}(\Pi_1, g, r, \Pi_2)$ ■

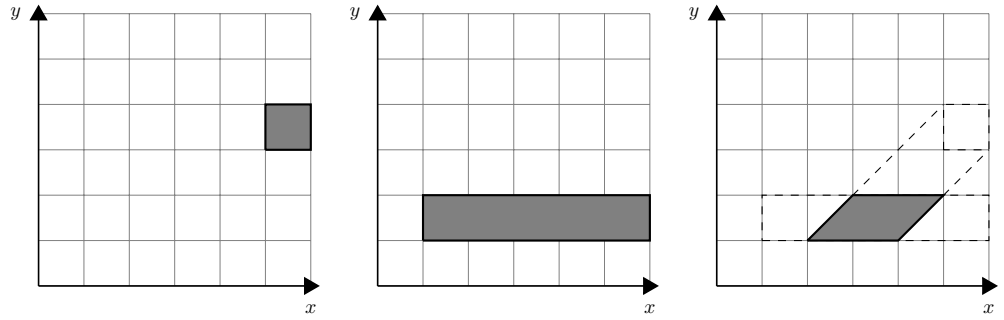
The action predecessor operation is implemented by operations under which zones are closed hence zones are closed under the action predecessor function.

Time Predecessor

The time predecessor finds any valuation from a set of valuations which may delay up to the current set of valuations.
Consider the set $\Pi' = \{v \mid 5 \leq v(x) \leq 6 \ \wedge \ 3 \leq v(y) \leq 4\}$ and the set $\Pi = \{v \mid 1 \leq v(x) \leq 6 \ \wedge \ 1 \leq v(y) \leq 2\}$. We are interested in finding all valuations in Π that can delay up to a valuation in Π' hence the set $\{v \mid 2 \leq v(x) \leq 4 \ \wedge \ 1 \leq v(y) \leq 2 \ \wedge \ -1 \leq v(y) - v(x) \leq -3\}$. The sets are all depicted Figure 4.6.

Definition 39 (Time Predecessor)
Let Π, Π' be set of valuations. We define

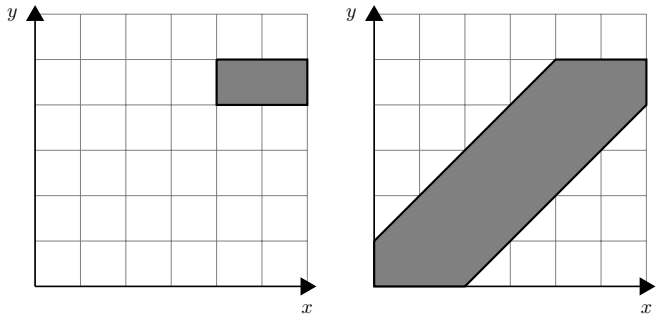
- $pre_{time}(\Pi, \Pi') \stackrel{def}{=} \{v \mid v \in \Pi \text{ and there exists } d \in \mathbb{R}_{\geq 0} \text{ such that } (v + d) \in \Pi'\}$ ♢



(a) Π' (b) Π (c) $pre_{time}(\Pi', \Pi)$

Figure 4.6: Example of using the time predecessor. (c) contains all the valuations in (b) that can reach a valuation in (a).

To construct the time predecessor we first need a way to find all valuations that can delay to a set of valuations. To this end we introduce the operator $^{\downarrow}$.
An example of applying $^{\downarrow}$ is shown in Figure 4.7



(a) Π (b) $\Pi^{\downarrow}$

Figure 4.7: Example of applying the $^{\downarrow}$ operator.

Definition 40 (Down)
Let $\Pi \in \mathcal{D}^{\mathcal{C}}$. We define

- $\Pi^{\downarrow} \stackrel{def}{=} \{v \mid \exists v' \in D \text{ and } d \in \mathbb{R}_{\geq 0} \text{ such that } v' = v + d\}$ ♢

We show zones are closed under $^{\downarrow}$.

Lemma 41
Let $\Pi \in \mathcal{D}^{\mathcal{C}}$ then $\Pi^{\downarrow} \in \mathcal{D}^{\mathcal{C}}$. ♢

PROOF.
Can be shown with a similar proof strategy as in Lemma 31. ■

Lemma 42
Let Π_1, Π_2 be sets of valuations over $\mathcal{C}$ such that $\Pi_2 \subseteq \Pi_1^{\uparrow}$ then $pre_{time}(\Pi_1, \Pi_2) = \Pi_2^{\downarrow} \cap \Pi_1$. $\diamondsuit$

PROOF.
We show two inclusions:

- $pre_{time}(\Pi_1, \Pi_2) \subseteq \Pi_1 \cap \Pi_2^{\downarrow}$
  Let $v \in pre_{time}(\Pi_1, \Pi_2)$ then there exists a valuation $v' \in \Pi_2$ and $d \in \mathbb{R}_{\geq 0}$ such that $v' = v + d$, hence

  $$v \in \{v_1 \mid \exists v_1' \in \Pi_1 \text{ and } d \in \mathbb{R}_{\geq 0} \text{ such that } v_1' = v_1 + d\} = \Pi_2^{\downarrow}.$$

  As $v \in pre_{time}(\Pi_1, \Pi_2)$, we also derive that $v \in \Pi_1$. Since v is in both Π_1 and $\Pi_2^{\downarrow}$, $v \in (\Pi_2^{\downarrow} \cap \Pi_1)$.

- $\Pi_1 \cap \Pi_2^{\downarrow} \subseteq pre_{time}(\Pi_1, \Pi_2)$
  Let $v \in (\Pi_1 \cap \Pi_2^{\downarrow})$ then $v \in \Pi_2^{\downarrow}$ and $v \in \Pi_1$. As v is in $\Pi_2^{\downarrow}$ we know that there exists a $d \in \mathbb{R}_{\geq 0}$ and $v_2 \in \Pi_2$ such that $v_2 = v + d$. Clearly

  $$v \in \{v_1 \mid v_1 \in \Pi_1 \text{ and there exists } d' \in \mathbb{R}_{\geq 0} \text{ such that } (v_1 + d') \in \Pi_2\} = pre_{time}(\Pi_1, \Pi_2).$$

The time predecessor is constructed by operations that zones are closed under, hence the time predecessor is a zone.

In the preceding we defined two operations to explore the symbolic state space backwards. The operations however assumed we somehow knew which valuations potentially could reach those we wished to find predecessors for. Compared to the predecessor operations used by for instance Bouajjani et al. [11] this is a non-standard way to define predecessor operations.

4.4 Implementation

To implement the zone operations, the dbm describing the zone must always be in closed form. Unfortunately the constructions made in our proofs do not preserve closed form. Since closing a DBM is a $\mathcal{O}(|\mathcal{C}|^3)$ time operation, it would be prudent to minimise the number of times we need to close a DBM. Most of the operations presented can be constructed such that the resulting set of difference constraints is in closed form [7]. Unfortunately no such construction has been found for the intersection between zones, thus the result of intersection between two zones must be closed afterwards. The intersection between a DBM and a single difference constraint, however, may be computed in $\mathcal{O}(|\mathcal{C}|^2)$ time [20].
. In Table 4.1 we summarise the complexity of the closed form preserving implementations described by Bengtsson and Rokicki.

4.5 Summary

In this chapter we presented the symbolic semantics of Timed Automata and different operations that facilitate symbolic state exploration. During symbolic state space exploration, the

Operation	Complexity
$D \cap \{x - y \lhd n\}$	$\mathcal{O}(\lvert\mathcal{C}\rvert^2)$
$D \cap D'$	$\mathcal{O}(\lvert\mathcal{C}^3\rvert)$
$D^{\uparrow}$	$\mathcal{O}(\lvert\mathcal{C}\rvert)$
$D_{r=0}$	$\mathcal{O}(\lvert\mathcal{C}\rvert \cdot \lvert r\rvert)$
$D^{\downarrow}$	$\mathcal{O}(\lvert\mathcal{C}\rvert^2)$
$free(D, r)$	$\mathcal{O}(\lvert\mathcal{C}\rvert \cdot \lvert r\rvert)$
$pre_{action}(D', g, r, D)$	$\mathcal{O}(\lvert\mathcal{C}\rvert^3)$
$pre_{time}(D', D)$	$\mathcal{O}(\lvert\mathcal{C}^3\rvert)$

Table 4.1: Complexity of the zone operations.

zones of symbolic states can be represented by DBMs in closed form. Most of the operations preserve this form, allowing efficient computation of the symbolic transition system. In the remainder of this thesis we always assume that DBMs are in closed form. Furthermore, we also assume that all sets of valuations we consider are zones.

Chapter 5

Safety Properties

Model checking tools provide diagnostic traces as a proof of the violation of a safety property. A diagnostic trace is a sequence of symbolic states and operations (guards and resets) such that the zones are connected by the resets and the guards. An initial attempt towards defining a trace is matching the operations from the symbolic semantics. We call such a trace a forward reachability trace.

Definition 43
A sequence $\langle \ell_1, D_1 \rangle \overset{g_1,r_1}{\rightsquigarrow} \langle \ell_2, D_2 \rangle \overset{g_2,r_2}{\rightsquigarrow} \dots \overset{g_{n-1},r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$ is a forward reachability trace for the Timed Automaton $(\mathcal{L}, \ell_0, \mathcal{C}, E, \mathtt{I})$ if there for all $i, 1 \leq i < n$, exists $\ell_i \xrightarrow{g_i,r_i} \ell_{i+1} \in E$ and

- $\ell_1 = \ell_0$,
- $D_1 = \{v_0\}^{\uparrow} \cap \mathtt{I}(\ell_1)$,
- $v_0 \in D_1$,
- for all i, $1 \leq i \leq n$, $D_i \neq \emptyset$, and
- for all i, $1 \leq i < n$, $D_{i+1} = ((D_i \cap [g_i])_{r_i=0})^{\uparrow} \cap [\mathtt{I}(\ell_{i+1})]$ $\diamondsuit$

An example of a Timed Automaton and a forward reachability trace is depicted in Figure 5.1.

A forward reachability trace guarantees that any valuation in a zone can be reached by some valuation in the preceding zone. This property is called post-stability [11]. Sometimes, we seek a guarantee that any valuation in a zone can reach a valuation in the following zone. This property is known as pre-stability [11].

Definition 44
A zone D is pre-stable wrt. a zone D' for $g \in DC(\mathcal{C})$ and $r \subseteq \mathcal{C}$ if

$$\forall v \in D, \exists d \in \mathbb{R}_{\geq 0} \text{ such that } (v+d) \vDash g, (v+d) \in D \text{ and } ((v+d)_{r=0} \in D'.$$

A zone D' is post-stable wrt. a zone D for $g \in DC(\mathcal{C})$ and $r \subseteq \mathcal{C}$ if

$$\forall v' \in D', \exists v \in D \text{ and } d \in \mathbb{R}_{\geq 0} \text{ such that } v \vDash g, v \in D \text{ and } (v_{r=0} + d) = v'. \quad \diamondsuit$$

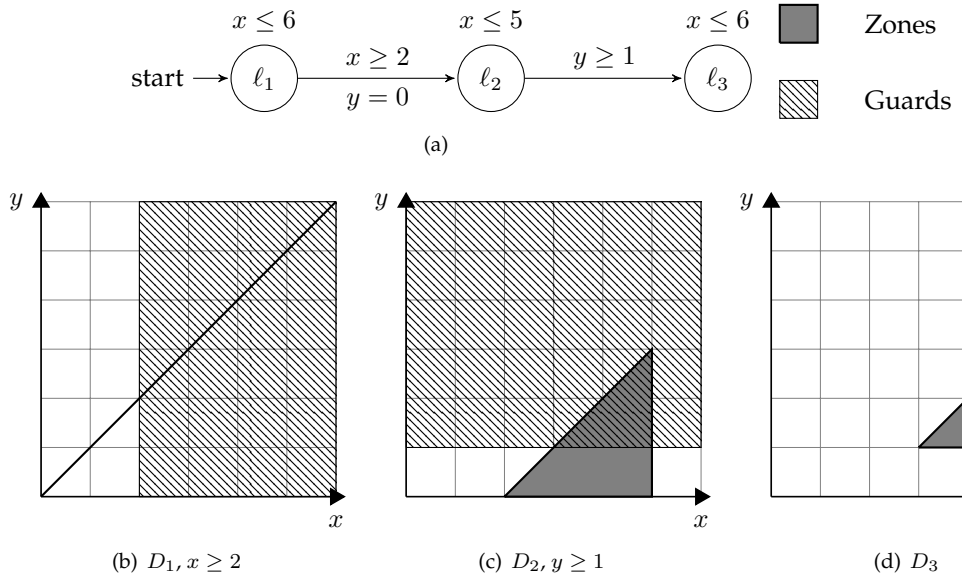


Figure 5.1: A forward reachability trace $\langle \ell_1, D_1 \rangle \overset{x \geq 2, \{y\}}{\rightsquigarrow} \langle \ell_2, D_2 \rangle \overset{y \geq 1, \{y\}}{\rightsquigarrow} \langle \ell_3, D_3 \rangle$ over the Timed Automaton depicted in (a). (b) depicts D_1 and $x \geq 2$, (c) depicts D_2 and $y \geq 1$, and (d) depicts D_3.

A forward reachability trace does not necessarily have the pre-stable property. Consider the forward reachability trace $\langle \ell_1, D_1 \rangle \overset{x \geq 2, \{y\}}{\rightsquigarrow} \langle \ell_2, D_2 \rangle \overset{y \geq 1, \{y\}}{\rightsquigarrow} \langle \ell_3, D_3 \rangle$ in Figure 5.1. Consider the zone D_2 and the valuation $v = [x = 5, y = 0.5]$ where $v(x) = 5$ and $v(y) = 0.5$. Then $v \in D_2$, but there does not exist a delay d such that $(v + d) \vDash g_2$ and $(v + d) \in D_2$, hence the forward reachability trace is not pre-stable.

To allow more pre-stable traces we introduce a general trace definition. This definition is less strict about the form of each zone, although we still require that each zone contains the valuations reached from the valuations in the previous zone that satisfy the guard.

Definition 45 (Symbolic trace)
The sequence $\langle \ell_1, D_1 \rangle \overset{g_1, r_1}{\rightsquigarrow} \langle \ell_2, D_2 \rangle \overset{g_2, r_2}{\rightsquigarrow} \dots \overset{g_{n-1}, r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$ is a *symbolic trace* for the TA $(\mathcal{L}, \ell_0, \mathcal{C}, E, \mathtt{I})$ if for all i, $1 \leq i < n$, $\ell_i \xrightarrow{g_i, r_i} \ell_{i+1} \in E$, $((D_i \cap [g_i])_{r_i = 0}) \cap \mathtt{I}(\ell_{i+1}) \subseteq D_{i+1}$ and for all $1 \leq i \leq n$

- $D_i \in \mathcal{D}^{\mathcal{C}}$,
- $D_i \neq \emptyset$,
- $D_i \subseteq [\mathtt{I}(\ell_i)]$, and
- $D_i \cap [g_i] \neq \emptyset$. ♢

Note that a forward reachability trace is a symbolic trace.
A post-stable symbolic trace $\langle \ell_1, D_1 \rangle \overset{g_1, r_1}{\rightsquigarrow} \langle \ell_2, D_2 \rangle \overset{g_2, r_2}{\rightsquigarrow} \dots \overset{g_{n-1}, r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$ is proof that D_n is reachable, but how it is reached is often too abstract for users to comprehend. A concretiza-

tion of a symbolic trace is a precise explanation of how the last state is reached. It provides a valuation of the clocks when entering each location and the amount of time to wait in each location.

Definition 46
A concretization of a symbolic trace $\langle \ell_1, D_1 \rangle \overset{g_1,r_1}{\rightsquigarrow} \langle \ell_2, D_2 \rangle \overset{g_2,r_2}{\rightsquigarrow} \dots \overset{g_{n-1},r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$ is a sequence

$$(\ell_1, v_1, d_1), (\ell_2, v_2, d_2), \dots, (\ell_n, v_n, d_n)$$

where $d_1, \dots, d_n \in \mathbb{R}_{\geq 0}$ such that for all i where $1 \leq i \leq n$

- $v_i \in D_i$,
- $(v_i + d_i) \in D_i$,

and for all i where $1 \leq i < n$

- $(v_i + d_i) \vDash g_i$, and
- $(v_i + d_i)_{r_i=0} = v_{i+1}$. ♢

We define the set of all concretizations of Ψ as $[\![\Psi]\!]^{trace} \overset{def}{=} \{\psi \mid \psi \text{ is a concretization of } \Psi\}$.

In the following we establish a link between the notion of symbolic traces and concretizations and the reachability of states. We first prove that finding concretizations of a symbolic trace amounts to finding transitions.

Lemma 47
Let $\Psi = \langle \ell_1, D_1 \rangle \overset{g_1,r_1}{\rightsquigarrow} \langle \ell_2, D_2 \rangle \overset{g_2,r_2}{\rightsquigarrow} \dots \overset{g_{n-1},r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$ be a symbolic trace for the Timed Automaton $(\mathcal{L}, \ell_0, \mathcal{C}, E, \mathtt{I})$. If $(\ell_1, v_1, d_1), (\ell_2, v_2, d_2), \dots, (\ell_n, v_n, d_n)$ is a concretization of Ψ then there exist transitions

$$\langle \ell_1, v_1 \rangle \xrightarrow{d_1} \langle \ell_1, v_1 + d_1 \rangle \xrightarrow{g_1,r_1} \langle \ell_2, v_2 \rangle \xrightarrow{d_2} \dots \xrightarrow{g_{n-1},r_{n-1}} \langle \ell_n, v_n \rangle \xrightarrow{d_n} \langle \ell_n, v_n + d_n \rangle.$$

PROOF.
We first show that the delay transitions exist. This amount to showing that for all i, $1 \leq i \leq n$, $v_i \vDash \mathtt{I}(\ell_i)$ and $v_i + d_i \vDash \mathtt{I}(\ell_i)$
First notice from the definition of a concretization that $v_i \in D_i$ and that $(v_i + d_i) \in D_i$. From the definition of a symbolic trace we know that $D_i \subseteq [\mathtt{I}(\ell_i)]$ hence obviously $v_i \in [\mathtt{I}(\ell_i)]$ and $(v_i + d_i) \in [\mathtt{I}(\ell_i)]$.

We now show that the conditions for the action transitions are met i.e. that for all i, $1 \leq i < n$, $(v_i + d_i) \vDash g_i$, $v_{i+1} = (v_i + d_i)_{r_i=0}$ and $v_{i+1} \vDash \mathtt{I}(\ell_{i+1})$.
From the definition of a concretization $(v_i + d_i) \vDash g_i$ and $v_{i+1} = (v_i + d_i)_{r_i=0}$. Showing that the invariant is satisfied was done in the previous case. ■

We are now ready to define the problem.

5.1 Problem Definition

In this chapter we are concerned with finding concretizations of symbolic traces for safety properties. The problems we consider are defined only on forward reachability traces as it is not

unreasonable to expect a violation of a safety property to be expressed as a forward reachability trace.
Initially we are concerned with finding any concretization.

Problem 1 For a forward reachability trace $\Psi = \langle \ell_1, D_1 \rangle \overset{g_1, r_1}{\rightsquigarrow} \dots \overset{g_{n-1}, r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$ where $v_0 \in D_1$, find a concretization $\psi = (\ell_1, v_1, d_1), \dots, (\ell_n, v_n, d_n)$ such that $\psi \in [\![\Psi]\!]^{trace}$ and $v_1 = v_0$.

The elapsed time of a concretization is the total time required to bring the Timed Automaton from the initial location to the final location. For a concretization $\psi = (\ell_1, v_1, d_1), \dots, (\ell_n, v_n, d_n)$ we define the elapsed time of ψ as

$$time(\psi) = \sum_{i=1}^{n-1} d_i.$$

The second problem we consider in this chapter is to find a concretization with the smallest possible elapsed time.

Problem 2 For a forwards reachability trace Ψ, find a concretization ψ solving Problem 1 such that for all ψ' solving Problem 1, $time(\psi) \leq time(\psi')$.

Problem 2 is not always solvable. Consider the timed automaton in Figure 5.2.

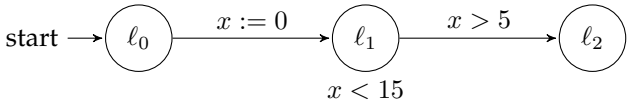


Figure 5.2: Timed Automaton for which no fastest concretization exist

A fastest concretization from ℓ_0 to ℓ_2 does not exist. The first edge can be taken at any time thus in particular when $x = 0$. The second edge can only be taken when $x > 5$. An initial attempt at finding a fastest concretization would be to take that edge when $x = 5.1$. A faster concretization would be to take the edge when $x = 5.05$ and an even faster trace would be obtained by taking the edge when $x = 5.025$. We can continue like this forever thus no fastest concretization exists.
Instead of finding the fastest concretization we consider finding one that is fastest with respect to some $\varepsilon \in \mathbb{R}$.

Problem 3 For a forward reachability trace Ψ and an $\varepsilon > 0$, find a concretization ψ solving Problem 1 such that for all ψ' solving Problem 1, $time(\psi) \leq time(\psi') + \varepsilon$.

The remainder of this chapter is focused on solving the problems above. Two different approaches are presented and afterwards compared. For both algorithms we present a solution to Problem 1 and modify this into solving Problem 3 and possibly Problem 2.

5.2 Entry Time Approach

The intuition behind this approach is that by observing a forward reachability trace, we can deduce when the symbolic states of it are entered relative to each other. To this end, we introduce the notion of absolute time. The absolute time is the total time passed in the model. For a forward reachability trace, the entry time of a symbolic state is the absolute time when entering the state. We formally define an entry time sequence.

Definition 48 (Entry Time Sequence)
A sequence $t_1, t_2, \ldots, t_n$ is an entry time sequence if for all i, $1 \leq i \leq n$, $t_i \in \mathbb{R}_{\geq 0}$, $t_1 = 0$ and for all i, $1 \leq i < n$, $t_i \leq t_{i+1}$. $\diamondsuit$

An entry time sequence is compatible with a symbolic trace if there exists a concretization where for every symbolic state in the symbolic trace, the sum of all delays in the preceding states corresponds to the entry time of the symbolic state.

Definition 49 (Compatibility with Symbolic Trace)
An entry time sequence $t_1, t_2, \ldots, t_n$ is compatible with a forward reachability trace $\Psi = \langle \ell_1, D_1 \rangle \overset{g_1, r_1}{\rightsquigarrow} \ldots \overset{g_{n-1}, r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$ if there exists a concretization $\psi = (\ell_1, v_1, d_1), \ldots, (\ell_n, v_n, d_n)$ where

- $\psi \in [\![\Psi]\!]^{trace}$, and
- for all i, $1 \leq i < n$, $d_i = t_{i+1} - t_i$. $\diamondsuit$

Note that no restrictions are placed on the delay d_n, except that it is part of the concretization. We proceed by exemplifying the approach used to find all compatible entry time sequences. To this end, let us fix a set of variables $\mathcal{E} = \{\mathbf{0}, e_1, \ldots, e_n\}$ representing entry times for the rest of this section.

Example 50
Consider the forward reachability trace $\langle \ell_1, D_1 \rangle \overset{g_1, r_1}{\rightsquigarrow} \langle \ell_2, D_2 \rangle \overset{g_2, r_2}{\rightsquigarrow} \langle \ell_3, D_3 \rangle$ in Figure 5.1. The entry time of each symbolic state naturally depends on the entry time of the previous symbolic state. We analyse the guards and resets in the trace and derive constraints for the entry time of each state. We use the variables e_1, e_2 and e_3 to represent the entry times of the symbolic states $\langle \ell_1, D_1 \rangle, \langle \ell_2, D_2 \rangle$ and $\langle \ell_3, D_3 \rangle$. A first observation is that the symbolic states are entered in the order $\langle \ell_1, D_1 \rangle$, $\langle \ell_2, D_2 \rangle$, $\langle \ell_3, D_3 \rangle$, hence the values of e_1, e_2, e_3 must be non-decreasing. To enforce this, we add the constraints $e_1 \leq e_2$ and $e_2 \leq e_3$.
As we begin at absolute time zero, we also have the constraint $e_1 = 0$. The invariant at location ℓ_1 must be satisfied both when entering and leaving $\langle \ell_1, D_1 \rangle$ (i.e. entering $\langle \ell_2, D_2 \rangle$). The invariant of ℓ_1 is $x \leq 6$, hence we add the constraint $e_1 \leq 6$. When leaving $\langle \ell_1, D_1 \rangle$, time has passed thus we need to add another constraint. The constraint we add in this case is $e_2 - e_1 \leq 6$.
Leaving $\langle \ell_1, D_1 \rangle$ is restricted by the guard $(x \geq 2)$ thus we add the constraint $e_2 - e_1 \geq 2$.
Upon entering $\langle \ell_2, D_2 \rangle$ the invariant $x \leq 5$ must be satisfied. As x has not been reset the value of x is $e_2 - e_1$ hence we add the constraint $e_2 - e_1 \leq 5$. Similarly, we add the constraint $e_3 - e_1 \leq 5$, since the invariant must also be satisfied when leaving $\langle \ell_2, D_2 \rangle$.
Entering $\langle \ell_3, D_3 \rangle$ is restricted by the guard $y \geq 1$. Because y was last reset upon entering $\langle \ell_2, D_2 \rangle$ we add $\ell_3 - \ell_2 \geq 1$. Finally we create the constraints for the invariant of ℓ_3 as in the previous cases. Combining the constraints gives the following set of difference constraints.

$$
\begin{aligned}
\{&e_1 \leq e_2, e_2 \leq e_3, \\
&e_1 \leq 6, e_2 - e_1 \leq 6, \\
&e_2 - e_1 \geq 2, \\
&e_2 - e_1 \leq 5, e_3 - e_1 \leq 5, \\
&e_3 - e_2 \geq 1, \\
&e_3 - e_1 \leq 6, e_3 - e_1 \leq 6\}
\end{aligned}
$$

Finding a solution to the constraints provides an entry time sequence for the Timed Automaton. Consider the solution $e_1 = 0$, $e_2 = 3$ and $e_3 = 5$. This solution implies that $e_2 - e_1 = 3$ is the amount of time that we should delay in ℓ_1 of the Timed Automaton before taking the transition to ℓ_2. We see that this is possible and that this allows us to move to ℓ_2. In the same way, $e_3 - e_2 = 2$ is the delay that we should perform in ℓ_2. We see that this is also possible, and that this allows us to move to ℓ_3.

Note that we have not used the zones D_1, D_2 and D_3 from the forward reachability trace at all. $*$

We devote the remainder of this section to formalising the approach presented in Example 50. We proceed by defining a function that given the index of a state and a clock returns the index of the state where the clock was last reset.

Definition 51 (Last Reset At)
Let $\Psi = \langle \ell_1, D_1 \rangle \overset{g_1,r_1}{\rightsquigarrow} \dots \overset{g_{n-1},r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$ be a forward reachability trace, $i \in \mathbb{Z}_{\geq 0}$ a non-negative integer and $x \in \mathcal{C}$ a clock. We define a function last reset at as

$$lra(\Psi, i, x) \overset{def}{=} max(\{j \mid (x \in r_{j-1} \wedge j \leq i)\} \cup \{1\}).$$

Using the last reset at function, we develop two funtions that construct constraints as in Example 50. The idea is that these functions shall create constraints over the the entry times based on delay/action respectively.

Definition 52
Let $\Psi = \langle \ell_1, D_1 \rangle \overset{g_1,r_1}{\rightsquigarrow} \dots \overset{g_{n-1},r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$ be a forward reachability trace, $\mathcal{E} = \{\mathbf{0}, e_1, e_2, \dots, \dots e_n\}$ a set of variables, δ a difference constraint and $i \in \mathbb{Z}_{\geq 0}$ a non-negative integer.
The function $afterAction$ is defined as:

$$afterAction(\Psi, i, \delta) \overset{def}{=} \begin{cases} e_i - e_{lra(\Psi,i,x)} \lhd m & \text{if } \delta = (x - \mathbf{0} \lhd m) \text{ where } x \neq \mathbf{0} \\ e_{lra(\Psi,i,x)} - e_i \lhd m & \text{if } \delta = (\mathbf{0} - x \lhd m) \text{ where } x \neq \mathbf{0} \\ e_{lra(\Psi,i,y)} - e_{lra(\Psi,i,x)} \lhd m & \text{if } \delta = (x - y \lhd m) \text{ and } x \neq \mathbf{0}, y \neq \mathbf{0} \end{cases}$$

The function $afterDelay$ is defined as:

$$afterDelay(\Psi, i, \delta) \overset{def}{=} \begin{cases} e_{i+1} - e_{lra(\Psi,i,x)} \lhd m & \text{if } \delta = (x - \mathbf{0} \lhd m) \text{ where } x \neq \mathbf{0} \\ e_{lra(\Psi,i,x)} - e_{i+1} \lhd m & \text{if } \delta = (\mathbf{0} - x \lhd m) \text{ where } x \neq \mathbf{0} \\ e_{lra(\Psi,i,y)} - e_{lra(\Psi,i,x)} \lhd m & \text{if } \delta = (x - y \lhd m) \text{ and } x \neq \mathbf{0}, y \neq \mathbf{0} \end{cases}$$

Finally, we define the entry time constraints generated by a forward reachability trace.

Definition 53 (Entry Time Constraints)
For a forward reachability trace $\Psi = \langle \ell_1, D_1 \rangle \overset{g_1,r_1}{\rightsquigarrow} \dots \overset{g_{n-1},r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$ of the Timed Automaton

$(\mathcal{L}, \ell_0, \mathcal{C}, E, \mathtt{I})$ we define the set of entry time constraints over Ψ as

$$\begin{aligned} \mathit{entryConstraints}(\Psi) \stackrel{def}{=} & \{\mathit{afterAction}(\Psi, i, \delta) \mid i \in \mathbb{Z}_{\geq 0} \wedge 1 \leq i \leq n \wedge \delta \in \mathtt{I}(\ell_i)\} \\ & \cup \{\mathit{afterDelay}(\Psi, i, \delta) \mid i \in \mathbb{Z}_{\geq 0} \wedge 1 \leq i < n \wedge \delta \in \mathtt{I}(\ell_i) \cup g_i\} \\ & \cup \{(e_1 - \mathbf{0} \leq 0), (\mathbf{0} - e_1 \leq 0)\} \\ & \cup \{e_i - e_{i+1} \leq 0 \mid i \in \mathbb{Z}_{\geq 0} \wedge 1 \leq i < n\}. \end{aligned}$$ ♢

To avoid confusion, we use the symbol π to represent a valuation over $\mathcal{E}$.

Theorem 54 (Soundness)
Let $\Psi = \langle \ell_1, D_1 \rangle \overset{g_1, r_1}{\rightsquigarrow} \ldots \overset{g_{n-1}, r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$ be a forward reachability trace for the Timed Automaton $(\mathcal{L}, \ell_0, \mathcal{C}, E, \mathtt{I})$. If π is a solution to $\mathit{entryConstraints}(\Psi) \subseteq DC(\mathcal{E})$ then the sequence $\pi(e_1), \pi(e_2), \ldots, \pi(e_n)$ is an entry time sequence and it is compatible with Ψ. ♢

PROOF.
Assume π is a solution to $\mathit{entryConstraints}(\Psi)$. We prove that $\pi(e_1), \pi(e_2), \ldots, \pi(e_n)$ is an entry time sequence. According to Definition 48 we must show that $\pi(e_1) = 0$ and for all i, $1 \leq i < n$, $\pi(t_i) \leq \pi(e_{i+1})$. This is trivially true due to

$$\{e_i - e_{i+1} \leq 0 \mid 1 \leq i < n\} \cup \{(e_1 - \mathbf{0} \leq 0), (\mathbf{0} - e_1 \leq 0)\} \subseteq \mathit{entryConstraints}(\Psi).$$

We have proven that $\pi(e_1), \pi(e_2), \ldots, \pi(e_n)$ is an entry time sequence. We prove that this entry time sequence is compatible with Ψ.

By Definition 49, $\pi(e_1), \pi(e_2), \ldots, \pi(e_n)$ is compatible with Ψ if there exists a concretization $\psi = (\ell_1, v_1, d_1), \ldots, (\ell_n, v_n, d_n)$ where $\psi \in [\![\Psi]\!]^{trace}$, and for all i, $1 \leq i < n$, $d_i = \pi(e_{i+1}) - \pi(e_i)$. We prove the existence of ψ by constructing it.
Let $\psi = (\ell_1, v_1, d_1), \ldots, (\ell_n, v_n, d_n)$ such that $v_1 = v_0$, $d_n = 0$ and for all i, $i \leq 1 < n$, $d_i = \pi(e_{i+1}) - \pi(e_i)$ and $v_{i+1} = (v_i + d_i)_{r_i = 0}$. What remains is to show that ψ is a concretization of Ψ. By Definition 46 we must show that for all i, $1 \leq i \leq n$

- $v_i \in D_i$,
- $(v_i + d_i) \in D_i$,

and for all i, $1 \leq i < n$

- $(v_i + d_i) \vDash g_i$,
- $(v_i + d_i)_{r_i = 0} = v_{i+1}$.

Note that $(v_i + d_i)_{r_i = 0} = v_{i+1}$ follows by construction. We show by induction on i that for all $1 \leq i < n$, $(v_i + d_i) \in D_i$, $v_i + d_i \vDash g_i$ and $v_{i+1} \in D_{i+1}$ using as our induction hypothesis that $v_i \in D_i$.

Base case We show that $v_1 \in D_1$. By Definition 43 we know $v_0 \in D_1$ and by construction $v_1 = v_0 \in D_1$ hence $v_1 \in D_1$.

For the remainder of this proof we let

- $D_0 = \{v_0\}$,
- $g_0 = \emptyset$ and
- $r_0 = \emptyset$.

Inductive step By the induction hypothesis $v_i \in D_i$. We show the following cases.

- $(v_i + d_i) \in D_i$
  By Definition 43 and the above definition of D_0, g_0 and r_0 we have that $D_i = ((D_{i-1} \cap [g_{i-1}])_{r_{i-1}=0})^{\uparrow} \cap [\mathtt{I}(\ell_i)]$. Since $v_i \in ((D_{i-1} \cap [g_{i-1}])_{r_{i-1}=0})^{\uparrow}$ then by Definition 28, $v_i + d_i \in ((D_{i-1} \cap [g_{i-1}])_{r_{i-1}=0})^{\uparrow}$ thus we need only show that $v_i + d_i \in [\mathtt{I}(\ell_i)]$.
  We consider the constraints on the following forms in $\mathtt{I}(\ell_i)$ and prove that $v_i + d_i$ satisfies them.

  - $x - y \lhd m$, where $x \neq \mathbf{0}$ and $y \neq \mathbf{0}$
    Since $v_i \in \mathtt{I}(\ell_i)$ we have

    $$(v_i(x) + d_i) - (v_i(y) + d_i) = v_i(x) - v_i(y) \lhd m.$$

  - $x - \mathbf{0} \lhd m$
    Let $j = lra(\Psi, i, x)$ be the index of the last location at which x was reset. Then $v_i(x) = d_j + d_{j+1} + \cdots + d_{i-1} = (\pi(e_{j+1}) - \pi(e_j)) + (\pi(e_{j+2}) - \pi(e_{j+1})) + \cdots + (\pi(e_i) - \pi(e_{i-1})) = \pi(e_i) - \pi(e_j)$. Recall that $d_i = \pi(e_{i+1}) - \pi(e_i))$, hence

    $$(v_i(x) + d_i) - v_i(\mathbf{0}) = \pi(e_i) - \pi(e_j) + (\pi(e_{i+1}) - \pi(e_i)) = \pi(e_{i+1}) - \pi(e_j.)$$

    Since $(e_{i+1} - e_j \lhd n) = \mathit{afterDelay}(\Psi, i, (x - \mathbf{0} \lhd m)) \in \mathit{entryConstraints}(\Psi)$ and π is a solution for $\mathit{entryConstraints}(\Psi)$ it follows that $\pi(e_{i+1}) - \pi(e_j) \lhd m$ and

    $$(v_i(x) + d_i) - v_i(\mathbf{0}) = \pi(e_{i+1}) - \pi(e_j) \lhd m.$$

  - $\mathbf{0} - x \lhd m$
    Let $j = lra(\Psi, i, x)$ is the index of the last location at which x was reset. As in the previous case $v_i(x) = \pi(e_i) - \pi(e_j)$ and $d_i = \pi(e_{i+1}) - \pi(e_i)$. Since $(e_j - e_{i+1} \lhd m) = \mathit{afterDelay}(\Psi, i, \mathbf{0} - x \lhd n) \in \mathit{entryConstraints}(\Psi)$ and π is a solution to $\mathit{entryConstraints}(\Psi)$ it follows that $\pi(e_j) - \pi(e_{i+1}) \lhd m$ and

    $$\mathbf{0} - (v_i + d_i) = -(\pi(e_i) - \pi(e_j) + \pi(e_{i+1}) - \pi(e_i)) = \pi(e_j) - \pi(e_{i+1}) \lhd m.$$

- $(v_i + d_i) \vDash g_i$
  We only provide the proof strategy used in this case, as it is similar to the previous.

  Rewrite $v_i + d_i$ to $\pi(e) - \pi(e')$. Let $(x - y \lhd m) \in g_i$, then derive that $\mathit{afterDelay}(\Psi, i, (x - y \lhd m)) = e - e' \lhd m$ and $\mathit{afterDelay}(\Psi, i, (x - y \lhd m)) \in \mathit{entryConstraints}(\Psi)$. Since π is a solution to $\mathit{entryConstraints}(\Psi)$, it satisfies $e - e' \lhd m$ and $(v_i + d_i) \vDash g_i$.

- $(v_{i+1}) \in D_{i+1}$
  From the previous cases we have that $(v_i + d_i) \in D_i \cap [g_i]$. From the definition of a forward reachability trace, we know that $((D_i \cap [g_i])_{r_i=0})^{\uparrow} \cap [\mathtt{I}(\ell_{i+1})] = D_{i+1}$. By construction $v_{i+1} = (v_i + d_i)_{r_i=0} \in (D_i \cap [g_i])_{r_i=0}$ hence we need only show that $v_{i+1} \in \mathtt{I}(\ell_{i+1})$. Doing this is a matter of repeating the arguments for the case $(v_i + d_i) \in D_i$, but appealing to the constraints from the $\mathit{afterAction}$ function.

Because $v_n \in D_n$ and $v_n + d_n = v + 0 = v_n$ we conclude that $v_n + d_n \in D_n$.
We conclude that $\pi(e_1), \pi(e_2), \ldots, \pi(e_n)$ is compatible with Ψ. ∎

Theorem 55 (Completeness)
Let $\Psi = \langle \ell_1, D_1 \rangle \overset{g_1,r_1}{\rightsquigarrow} \dots \overset{g_{n-1},r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$ be a forward reachability trace for the Timed Automaton $(\mathcal{L}, \ell_0, \mathcal{C}, E, \mathtt{I})$ and let $t_1, t_2, \dots, t_n$ be an entry time sequence that is compatible with Ψ. Then π where i where $1 \leq i \leq n$ is a solution to $entryConstraints(\Psi) \subseteq DC(\mathcal{E})$. $\diamondsuit$

PROOF.
Proof by construction.
Since $t_1, t_2, \dots, t_n$ is compatible with Ψ by Definition 49 there exists a concretization $\psi = (\ell_1, v_1, d_1), \dots, (\ell_n, v_n, d_n)$ such that for all i, $1 \leq i < n$

$$d_i = t_{i+1} - t_i.$$

Throughout this proof we use the following observation: Let x be any clock, i any location index and $lra(\Psi, i, x) = j$ then

$$v_i(x) = d_j + d_{j+1} + \dots + d_{i-1} = (t_{j+1} - t_j) + (t_{j+2} - t_{j+1}) + \dots + (t_i - t_{i+1}) = t_i - t_j = t_i - t_{lra(\Psi,i,x)}.$$

Let π be a valuation such that for all i, where $1 \leq i \leq n$, $\pi(e_i) = t_i$. We prove that π is a solution to $entryConstraints(\Psi)$ by showing that all constraints in $entryConstraints(\Psi)$ are satisfied. Recall that

$$\begin{aligned} entryConstraints(\Psi) = &\{afterAction(\Psi, i, \delta) \mid i \in \mathbb{Z}_{\geq 0} \wedge 1 \leq i \leq n \wedge \delta \in \mathtt{I}(\ell_i)\} \\ &\cup \{afterDelay(\Psi, i, \delta) \mid i \in \mathbb{Z}_{\geq 0} \wedge 1 \leq i < n \wedge \delta \in \mathtt{I}(\ell_i) \cup g_i\} \\ &\cup \{(e_1 - \mathbf{0} \leq 0), (\mathbf{0} - e_1 \leq 0)\} \\ &\cup \{e_i - e_{i+1} \leq 0 \mid i \in \mathbb{Z}_{\geq 0} \wedge 1 \leq i < n\}. \end{aligned}$$

We split into four cases corresponding to each subset of $entryConstraints(\Psi)$.

- $\{afterDelay(\Psi, i, \delta) \mid 1 \leq i \leq n-1 \wedge \delta \in \mathtt{I}(\ell_i) \cup g_i\}$
  For any i, $1 \leq i \leq n-1$, we consider the constraints that *afterDelay* creates and prove that π satisfies these. Recall from Definition 46 that $(v_i + d_i) \in D_i$ and by Definition 43 $D_i \subseteq [\mathtt{I}(\ell_i)]$ hence $(v_i + d_i) \vDash \mathtt{I}(\ell_i)$. By Definition 46 we also have that $(v_i + d_i) \vDash g_i$.

  By Definition 52 *afterAction* generates the following forms of constraints.

  - $e_{i+1} - e_{lra(\Psi,i,x)} \lhd m$ if $\delta = (x - \mathbf{0} \lhd m)$ where $x \neq \mathbf{0}$
    Since $(v_i + d_i) \vDash \mathtt{I}(\ell_i)$, $(v_i + d_i) \vDash g_i$ and $\delta \in \mathtt{I}(\ell_i) \cup g_i$ we have $(v_i + d_i)(x) - (v_i + d_i)(\mathbf{0}) \lhd m \Rightarrow (v_i + d_i)(x) \lhd m$. We now prove that π satisfies $(e_{i+1} - e_{lra(\Psi,i,x)} \lhd m)$

    $$\begin{aligned} \pi(e_{i+1}) - \pi(e_{lra(\Psi,i,x)}) &= t_{i+1} - t_{lra(\Psi,i,x)} \\ &= t_{i+1} - t_{lra(\Psi,i,x)} + (t_i - t_i) \\ &= (t_i - t_{lra(\Psi,i,x)}) + (t_{i+1} - t_i) \\ &= v_i(x) + d_i \\ &= (v_i + d_i)(x) \lhd m. \end{aligned}$$

  - $e_{lra(\Psi,i,x)} - e_{i+1} \lhd m$ if $\delta = (\mathbf{0} - x \lhd m)$ where $x \neq \mathbf{0}$
    Since $(v_i + d_i) \vDash \mathtt{I}(\ell_i)$, $(v_i + d_i) \vDash g_i$ and $\delta \in \mathtt{I}(\ell_i) \cup g_i$ we have $(v_i + d_i)(\mathbf{0}) - (v_i +$

$d_i)(x) \lhd m \Rightarrow -(v_i + d_i)(x) \lhd m$. Then

$$\begin{aligned}\pi(e_{lra(\Psi,i,x)}) - \pi(e_{i+1}) &= t_{lra(\Psi,i,x)} - t_{i+1}\\ &= t_{lra(\Psi,i,x)} + t_i - t_i - t_{i+1}\\ &= -((t_i - t_{lra(\Psi,i,x)}) + (t_{i+1} - t_i))\\ &= -(v_i(x) + d_i)\\ &= -(v_i + d_i)(x) \lhd m.\end{aligned}$$

– $e_{lra(\Psi,i,y)} - e_{lra(\Psi,i,x)} \lhd m$ if $\delta = (x - y \lhd m)$ and $x \neq \mathbf{0}, y \neq \mathbf{0}$
Since $(v_i + d_i) \vDash \mathtt{I}(\ell_i)$, $(v_i + d_i) \vDash g_i$ and $\delta \in \mathtt{I}(\ell_i) \cup g_i$ we have $(v_i + d_i)(x) - (v_i + d_i)(y) \lhd m$. Then

$$\begin{aligned}\pi(e_{lra(\Psi,i,y)}) - \pi(e_{lra(\Psi,i,x)}) &= t_{lra(\Psi,i,y)} - t_{lra(\Psi,i,x)}\\ &= t_i - t_i + t_{lra(\Psi,i,y)} - t_{lra(\Psi,i,x)}\\ &= t_i - t_{lra(\Psi,i,x)} - (t_i - t_{lra(\Psi,i,y)})\\ &= v_i(x) - v_i(y)\\ &= v_i(x) + d_i - d_i - v_i(y)\\ &= v_i(x) + d_i - (v_i(y) + d_i)\\ &= (v_i + d_i)(x) - (v_i + d_i)(y) \lhd m.\end{aligned}$$

■

- $\{afterAction(\Psi, i, \delta) \mid 1 \leq i \leq n \ \wedge \ \delta \in \mathtt{I}(\ell_i)\}$
A similar strategy as above can be used to prove that π satisfies these constraints.

- $\{(e_1 - \mathbf{0} \leq 0), (\mathbf{0} - e_1 \leq 0)\}$
By Definition 48, $t_1 = 0$. Then
$$\pi(e_1) = t_1 = 0.$$

- $\{e_i - e_{i+1} \leq 0 \mid 1 \leq i < n\}$
By Definition 48 we have for all i, $1 \leq i < n$, that $t_i \leq t_{i+1} \Rightarrow t_i - t_{i+1} \leq 0$. Then for all i where $1 \leq i < n$,
$$\pi(e_i) - \pi(e_{i+1}) = t_i - t_{i+1} \leq 0.$$

Complexity

The approach discussed in this section creates constraints and finds a solution to these. Creating the constraints involves using the *afterDelay* and *afterAction* functions which use the last-reset-at function. Assuming the last-reset-at function is a constant operation we derive that creating the constraints is an $\mathcal{O}(m)$ operation, where m is the total number of constraints in the symbolic trace.

The result of Definition 53 is a set of difference constraints over n variables, where n is the number of states. Solving these constraints can be done by using FINDPOINT which is quadratic in the number of variables. To use FINDPOINT the set of difference constraints must however be in closed form. Closing a set of difference constraints is cubic in the number of variables.

The result of the above is that the complexity of the approach presented in this section is $\mathcal{O}(m + n^3)$.

Strategy

Let $\Psi = \langle \ell_1, D_1 \rangle \overset{g_1,r_1}{\rightsquigarrow} \ldots \overset{g_{n-1},r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$ be a forward reachability trace. We wish to find a solution v to $entryConstraints(\Psi)$ such that we by using the construction from Theorem 54 solves Problem 1, 2 and 3.

Problem 1 Solving this problem is a matter of finding any solution to $entryConstraints(\Psi)$.

Problem 2 Let $\mathbf{0} - e_n \lhd -l$ be the lower bound on entering $\langle \ell_n, D_n \rangle$. In case, $\lhd = \leq$ we solve Problem 2 by finding a solution π such that $\pi(e_n) = l$.
In case $\lhd = <$ solving Problem 2 is not possible.

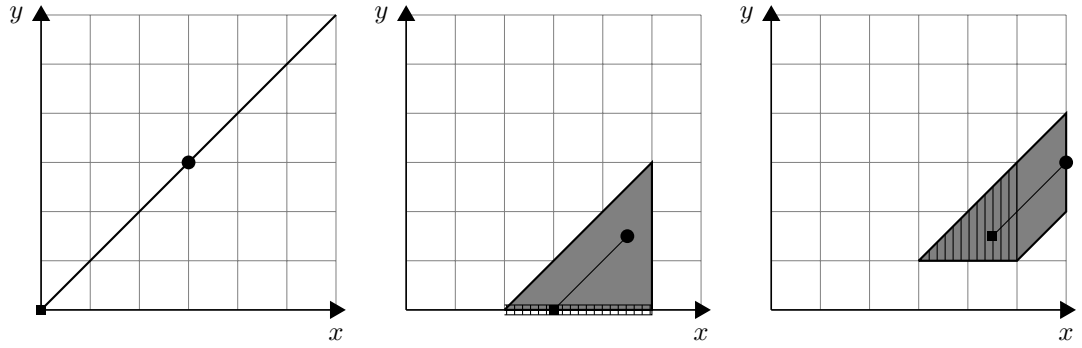
Problem 3 Let $\mathbf{0} - e_n \lhd -l$ be the lower bound on entering $\langle \ell_n, D_n \rangle$. Solving Problem 3 is a matter of finding a solution π such that $\pi(e_n) = l + \varepsilon$.

5.3 Backwards Approach

In this section we present an algorithm that instead of observing the entire symbolic trace focuses on one of the zones of a symbolic trace at a time. The algorithm chooses valuations in the zones while ensuring the chosen valuations are connected by resets and delays. In each zone two different valuations are selected. The first valuation gives the value of each clock when entering the zone whereas the second gives the value when leaving the zone.

Example 56
Let $\langle \ell_1, D_1 \rangle \overset{g_1,r_1}{\rightsquigarrow} \ldots \overset{g_{3-1},r_{3-1}}{\rightsquigarrow} \langle \ell_3, D_3 \rangle$ be the forward reachability in Figure 5.1 trace for the Timed Automaton in Figure 5.1(a).
To find a concretization over the trace we first find any valuation $v_3' \in D_3$ (indicated by the circle in Figure 5.3(c)). Secondly we find a second valuation $v_3 \in D_3$. The second valuation (indicated by a square in Figure 5.3(c)) is selected such that there exists a d_3 such that $v_3' = v_3 + d_3$ and v_3 in $(D_2 \cap g_2)_{r_2=0}$ (indicated by the vertical lines in Figure 5.3(c)).
In the second zone we choose v_2' such that $(v_2')_{r_2=0} = v_3$. As $r_2 = \emptyset$ $v_2' = v_3$ (indicated by a circle in Figure 5.3(b)). A second valuation v_2 is found in D_2 such that it is contained within $(D_1 \cap g_1)_{r_1=0}$ and there exists a d_2 for which $v_2' = v_2 + d_2$ (the square in Figure 5.3(b)).
In D_1 we select a valuation v_1' such that $(v_1')_{r_1=0} = v_2$ (the circle in Figure 5.3(a)). Finally we set v_1 such that it can delay up to v_1' (the square in Figure 5.3(a). $*$

We formalise the approach that was exemplified in Example 56 in Algorithm 3.



(a) D_1 (b) D_2 (c) D_3

Figure 5.3: The backward operations applied to the zones over the Timed Automaton in Figure 5.1

Algorithm 3: Backwards Exploration algorithm

Input: A post-stable symbolic trace $\Psi = \langle \ell_1, D_1 \rangle \overset{g_1,r_1}{\rightsquigarrow} \dots \overset{g_{n-1},r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$
Output: A concretization of Ψ

$v'_n = \text{FINDPOINT}(D_n)$;
v_n=FINDPOINT$(pre_{time}(D_n \cap (D_{n-1} \cap g_{n-1})_{r_{n-1}=0}, \{v'_n\}))$;
Set d_n such that $v'_n = v_n + d_n$;
for $i = n-1$ **to** 2 **do**
    v'_i = FINDPOINT$(pre_{action}(D_i, g_i, r_i, \{v_{i+1}\}))$;
    v_i=FINDPOINT$(pre_{time}(D_i \cap (D_{i-1} \cap g_{i-1})_{r_{i-1}=0}, \{v'_i\}))$;
    Set d_i such that $v_i + d_i = v'_i$;
end
v'_1 = FINDPOINT$(pre_{action}(D_1, g_1, r_1, \{v_2\}))$;
$v_1 = \text{FINDPOINT}(pre_{time}(D_1, \{v'_1\}))$;
Set d_1 such that $v'_1 = v_1 + d_1$;
return $(\ell_1, v_1, d_1), \dots, (\ell_n, v_n, d_n)$

Algorithm 3 consists of one for loop hence it always terminates. The FINDPOINT algorithm returns "inconsistent" if called with an empty zone. In the following we assert that this is never the case.

Lemma 57
Let $D, D' \in \mathcal{D}^{\mathcal{C}}$ such that D is post-stable wrt. D' for $g \subseteq DC(\mathcal{C})$ and $r \subseteq \mathcal{C}$. For any $\Pi \subseteq D'$, $pre_{time}(D' \cap (D \cap g)_{r=0}, \Pi) \neq \emptyset$. ♢

PROOF.
From Definition 39

$pre_{time}(D' \cap (D \cap g)_{r=0}, \Pi) = \{v \mid v \in D' \cap (D \cap g)_{r=0}$ and there exists d such that $v+d \in \Pi\}$.

Let $v \in \Pi$. Because $\Pi \subseteq D'$ and D is post-stable wrt. D' we know there exists a $v' \in D \cap [g]$ and delay $d \in \mathbb{R}_{\geq 0}$ such that $v'_{r=0} + d = v$. Obviously $v'_{r=0}$ is contained in $D' \cap (D \cap g)_{r=0}$ thus $v'_{r=0}$ is contained in $pre_{time}(D' \cap (D \cap g)_{r=0}, \Pi)$. ■

Lemma 58
Let $D, D' \in \mathcal{D}^{\mathcal{C}}$ such that D is post-stable wrt. D' for $g \subseteq DC(\mathcal{C})$ and $r \subseteq \mathcal{C}$ and $(D \cap [g])_{r=0} \subseteq D'$. For any $\Pi \subseteq (D \cap [g])_{r=0}$ $pre_{action}(D, g, r, \Pi) \neq \emptyset$. $\diamondsuit$

PROOF.
Let $v' \in \Pi$. We show that $pre_{action}(D, g, r, \Pi) \neq \emptyset$ by finding a $v \in D$ such that $v \vDash g$ and $v' = v_{r=0}$. Because D is post stable wrt. D' for g and r we know that there exists a $v \in D$, $d \in \mathbb{R}_{\geq 0}$ such that $v \vDash g$ and $v' = v_{r=0} + d$. Since $v' \in \Pi \subseteq ((D \cap [g])_{r=0})$ we derive that $d = 0$, hence $v' = v_{r=0}$.
We have found a valuation v such that $v \vDash g$ and $v' = v_{r=0}$ hence $v \in pre_{action}(D, g, r, \Pi)$. We have found a valuation in $v \in pre_{action}(D, g, r, \Pi)$ and conclude that $v \in pre_{action}(D, g, r, \Pi) \neq \emptyset$. ■

Correctness

We are now ready to state that the output of Algorithm 3 is a concretization of its input.

Lemma 59 (Soundness)
Let $\Psi = \langle \ell_1, D_1 \rangle \overset{g_1, r_1}{\rightsquigarrow} \dots \overset{g_{n-1}, r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$ be a symbolic trace and $\psi = (\ell_1, v_1, d_1), \dots, (\ell_n, v_n, d_n)$ the output of Algorithm 3 on Ψ, then $\psi \in [\![\Psi]\!]^{trace}$. $\diamondsuit$

PROOF.
Let $\psi = (\ell_1, v_1, d_1), \dots, (\ell_n, v_n, d_n)$ be the output of Algorithm 3 on Ψ. We prove the lemma by showing the conditions from Definition 46 are all satisfied. To this in we use the loop invariant: for all j , $i < j \leq n$ that

- $v_j \in D_j$,
- $(v_j + d_j) \in D_j$,

and for all j, $1 \leq j < n$

- $(v_j + d_j) \vDash g_j$,
- $(v_j + d_j)_{r_j=0} = v_{j+1}$.

Initialisation We show the requirements are satisfied before entering the loop i.e. for $i = n$. Only two requirements apply namely $v_n \in D_n$ and $(v_n + d_n) \in D_n$.
In line 1 v' is selected from D_n and as d_n in line 3 is selected such that $v' = v_n + d_n$ it follows that $(v_n + d_n) \in D_n$.
In line 2 v_n is selected from $pre_{time}(D_n \cap (D_{n-1} \cap g_{n-1})_{r_{n-1}=0}, \{v'_n\})$ thus obviously $v_n \in D_n$.

Maintenance By the loop invariant we know that $v_{i+1} \in D_{i+1}$. In line 5 v' is selected from $pre_{action}(D_i, g_i, r_i, \{v_{i+1}\})$ thus by definition $v' \vDash g_i$, $v' \in D_i$ and $v'_{r_i=0} = v_i$. In line 7 d_i is set such that $v' = v_i + d_i$.
Finally we assert $v_i \in D_i$. From the above we know that $v' \in D_i$ and v_i is selected from $pre_{time}(D_i \cap (D_{i-1} \cap g_{i-1})_{r_{i-1}=0}, \{v'_i\})$ hence it trivially follows that $v_i \in D_i$.

Termination At termination we have that all conditions are satisfied for all i where $2 \leq i \leq n$. What remains is to prove that they are also satisfied for $i = 1$. However, the arguments for this are almost equivalent to the above hence we do not repeat them. ∎

Lemma 60 (Completeness)
Let Ψ be a symbolic trace and let $\psi \in [\![\Psi]\!]^{trace}$. There exists a computational branch of Algorithm 3 run on Ψ that returns ψ. ♢

PROOF.
Let $\Psi = \langle \ell_1, D_1 \rangle \overset{g_1,r_1}{\rightsquigarrow} \dots \overset{g_{n-1},r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$ and $\psi = (\ell_1, v_1, d_1), \dots, (\ell_n, v_n, d_n)$.
We show that Algorithm 3 is capable of choosing the valuations and delays in ψ. More specifically we show that the valuations in ψ are contained in the zones from which Algorithm 3 chooses its valuations.

We first show that Algorithm 3 can find $v_n + d_n$ and v_n. As $\psi \in [\![\Psi]\!]^{trace}$ we know $v_n + d_n$ is contained in D_n. Due to completeness of FINDPOINT it follows the algorithm can find $v_n + d_n$ in line 1. Since $v_n = (v'_{n-1} + d_{n-1})_{r_{n-1}=0}$ we are assured v_n is contained in $(D_{n-1})_{r_{n-1}}$. Clearly $v_n \in pre_{time}(D_{n-1_{r_{n-1}=0}} \cap D_n, \{v_n + d_n\})$, Algorithm 3 can choose v_n in line 2.

We now inductively prove that the algorithm for all i, $1 < i < n$, can find v_i and $v_i + d_i$ using as hypothesis that it has found v_{i+1}.
We first prove it can find the valuation $v_i + d_i$. By Definition 46 $v_i + d_i \vDash g_i$ and $(v_i + d_i)_{r_i=0} = v_{i+1}$. This corresponds exactly to the definition of $pre_{action}(D'_i, g_i, r_i, \{v_{i+1}\})$ thus $(v_i + d_i) \in pre_{action}(D'_i, g_i, r_i, \{v_{i+1}\})$. FINDPOINT is complete hence in line 5 Algorithm 3 can choose $v_i + d_i$.
Assume Algorithm 3 has found $v_i + d_i$. From ψ we know $v_i = (v_{i-1} + d_{i-1})_{r_{i1}=0}$ thus $v_i \in (D_{i-1})_{r_{i_1}=0}$. We may therefore derive that $v_i \in pre_{time}(D_i \cap (D_{i-1})_{r_{i-1}=0}, \{v_i + d_i\})$. This is exactly the set from which Algorithm 3 chooses its valuations thus in line 6 it can find v_i.

Finally we need to assert that the algorithm can find v_1. As this is similarly proven we will not repeat the statements here. ∎

Complexity

Algorithm 3 iterates through all the states and applies operations in each iteration. In each iteration the action predecessor, time predecessor and FINDPOINT functions. Additionally the intersection operation is used in each iteration of the for-loop. The intersection, action predecessor and time predecessor are all cubic in the number of clocks whereas FINDPOINT is quadratic in the number of clocks. If we let n denote the number of symbolic states in the trace and let $\mathcal{C}$ be the set of clocks, the complexity is $\mathcal{O}(n \cdot |\mathcal{C}|^3)$.

Strategy

Algorithm 3 is nondeterministic since is uses FINDPOINT. In the following we describe how tol solve the following problems.

Problem 1 Use Algorithm 3 to find a concretization and when FINDPOINT is called in line 10, choose v_0.
This is possible since $v_0 \in D_1$ and all valuation in D_1 are on the form $v_0 + d$.

Problem 2 In order to solve Problem 2 we assume the presence of a clock that has never been reset. Let x be such a clock and let $\mathbf{0} - x \lhd -m$ be the lower bound in the last location of the symbolic trace. If $\lhd = \leq$ we solveProblem 2 by choosing a valuation in line 1 such that $x = m$. The remaining valuations to be selected are found using the same strategy as in Problem 1.
In case $\lhd = <$ no solution exists for Problem 2.

Problem 3 Solving Problem 3 in the way as Problem 2 with the exception that if $\lhd = <$ a valuation is chosen in 1 such that $v(x) = m + \varepsilon$.

From now on forward we refer to Algorithm 3 by BACKWARDS. For some applications it may be useful to find a concretization which reaches a specific valuation in the last location. In fact we can make an alternative implementation of Algorithm 3. The idea is that instead of nondeterministically choose a valuation in line 1 we might as well just be given a valuation which we know is contained in the last zone. Formally we are given a valuation v and a trace $\Psi = \langle \ell_1, D_1 \rangle \overset{g_1, r_1}{\rightsquigarrow} \dots \overset{g_{n-1}, r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$ such that $v \in D_n$ and wishes to find a concretization $(\ell_1, v_1, d_1), \dots, (\ell_n, v_n, d_n)$ such that $v = v_n + d_n$. To do this we alter the algorithm in line 1 to set $v' = v$. We call this variation of the backwards algorithm by BACKWARDS(Ψ, v).

5.4 Experiments

The theoretical complexity of the backwards and entry time aproach indicate the backwards solution ought to be better if the number of clocks is relatively small compared to the number of states. At the same time we observe that the entry time should be better if there are relatively few states compared to the number of clocks.
The algorithms have been implemented in the tool CTU in order to do practical experiments concerning the time complexity and memory consumption.

Method

In the experiments we wish to test the scalability of the two algorithms in two dimensions: Number of states and number of clocks. Additionally we wish to test how well the algorithms performs on some well-known models and a model which UPPAAL has previously failed to generate a concretization of. Below is a brief explanation of the traces and their origin.

Bridge This model comes with UPPAAL. The problem modelled is that four vikings moving at different speed wish to cross an old bridge during the night. The vikings can only pass the bridge two at a time and one must carry a torch. Unfortunately they only have one torch. For this model we wish to find the fastest way to move all vikings to the other side of the bridge.

bug467 This model creates a forwards reachability trace for which UPPAAL in the past has failed to generate concretizations for. The model and query were found in the UPPAAL bug tracking system[1].

[1] http://bugsy.grid.aau.dk/cgi-bin/bugzilla/show_bug.cgi?id=467

bocdp Originally this model was developed to find a bug in a communication protocol used by B&O [15].
As the model used commited locations, which is currently not supported by CTU, it has been slightly modified. A clock, some invariants and resets have been added such that the system cannot delay in locations that are commited.

Scalability wrt. number of states The models used for the scalability testing are based on the model shown in Figure 5.4. By altering the value of `top` the number of states in the trace can be changed while keeping the number of clocks constant. In the forward reachability trace there are four clocks of which only two are actively used.
The asked query is if there exists a path to the `end` location.

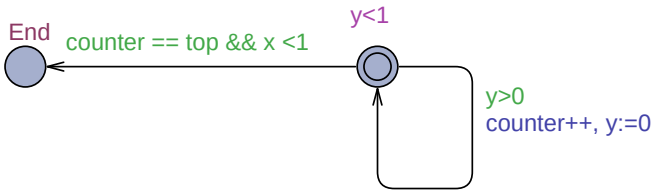


Figure 5.4: The model used to test the scalability with respect to the number of states. In this model x and y are clocks and $counter$ and top are integer variables. By altering the of value top we can change the number of states in the symbolic trace.

Scalability wrt. number of clocks The models used to test the scalability with respect to number of clocks are derived from the model shown in Figure 5.5. By altering the size of z we can modify the number of clocks in the Timed Automaton. There are always 122 states in the symbolic trace. The asked asked is if there exists a path to the `end` location.

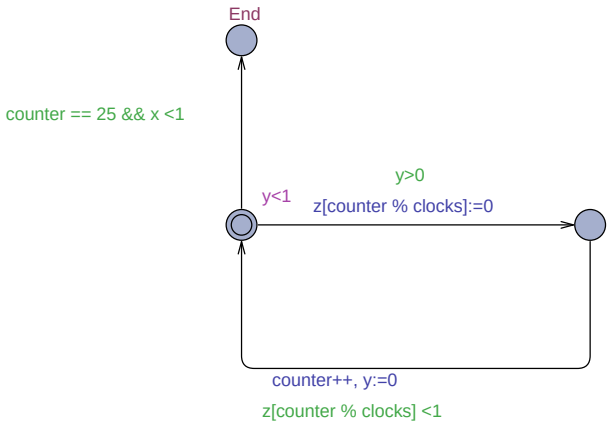


Figure 5.5: The model used to test the scalability with respect to the number of clocks. In this mode$z$ is an array of clocks of size $clocks$ and x and y are clocks as well.

CTU was run on an Intel Core 2 Duo (1.8 GHz) machine with 1GB of memory running Debian 5.0. The time and memory usage was measured using the `memtime` utility developed by Bengtsson [8]. The utility observes the ram usage of a process through the `/proc` filesystem. The maximum memory usage is estimated using samples hence the results are not always accurate. Additionally, it measures the time the process used to complete its task.
A *dummy* test that only loads the trace. This allows measuring the time spend loading the trace and thereby estimate the actual running time of the algorithms. Each test was run ten times and a mean value was found.

Results

The results from the tests are shown in Table 5.1, Figure 5.6 and Figure 5.7.

Model	E-RAM	B-RAM	D-RAM	E-TIME	B-Time	D-Time
bocdp	77628	54708	54734	10.00	5.96	2.20
bridge	10612	12773	5179	.12	.12	.10
bug467	4590	6724	2748	.10	.11	.10

Table 5.1: Test results. The E-columns refer to the results from the entry approach, B-columns to those for the backwards algorithm and finally the D-columns the results from the dummy test. The unit of measure for memory usage is kilobytes and for time is seconds.

Discussion

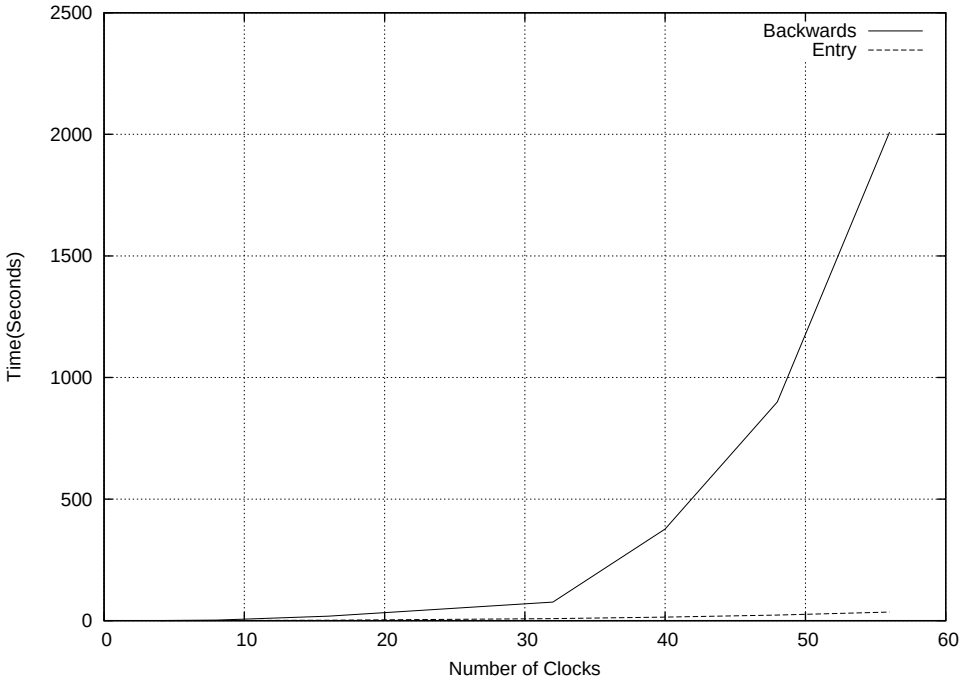
The results in Table 5.1 show that the backwards approach is faster on the large industrial case (`bocdp`) but at the same time they show no significant difference on the other examples. We therefore turn our attention to the tests concerning scalability.

Scalability wrt. clocks The test concerning clocks, see Figure 5.6, indicates the backwards approach is the slowest and that the entry time approach follows the loading time quite closely. The tests also show that the memory usage almost is the same for the two approachs. We believe the minor deviations sten from inaccuracies in the memory usage measurements. Note that the memory usage of the backwards and entry time approach do not vary much the dummy test. This indicate that loading the symbolic trace is the most memory consuming part of both approachs. This is important as the entry time approach does not use much information from the trace whereas the backwards algorithm uses most of it. For instance the DBMs are not used by the entry time approach.

Scalability wrt. state The state tests, see Figure 5.7, indicate that the backwards approach is slower than the entry time approach for all number of states lower than 1500 states. The time complexity of the entry time approach appears to be much worse than that of the backwards approach but has a starting point much lower. This combined with the fact that a trace rarely contains more than a 1000 states ought to indicate the entry time approach is the fastest. However, it should be mentioned that the implementation of the zone operations in the backwards approach is by no means optimised. We think it is possible to implement the operations more efficiently by avoiding temporary zones.
The memory usage of the backwards algorithm is almost the same as the dymmy test. The entry

Clocks	E-RAM	B-RAM	D-RAM	E-TIME	B-TIME	D-TIME
4	24029	24094	24340	.23	.70	.24
8	32152	32125	32284	.53	2.41	.51
16	59866	59880	59880	1.70	18.78	1.70
32	162200	162200	162200	8.19	76.32	8.19
40	236932	236932	236932	14.56	376.97	14.21
48	327512	327512	327512	22.95	899.67	22.83
56	433724	433764	437672	35.19	2009.19	34.79

(a) The E-columns refer to the results from the entry approach, B-columns to those for the backwards algorithm and finally the D-columns the results from the dummy test. The unit of measure for memory usage is kilobytes and for time is seconds.

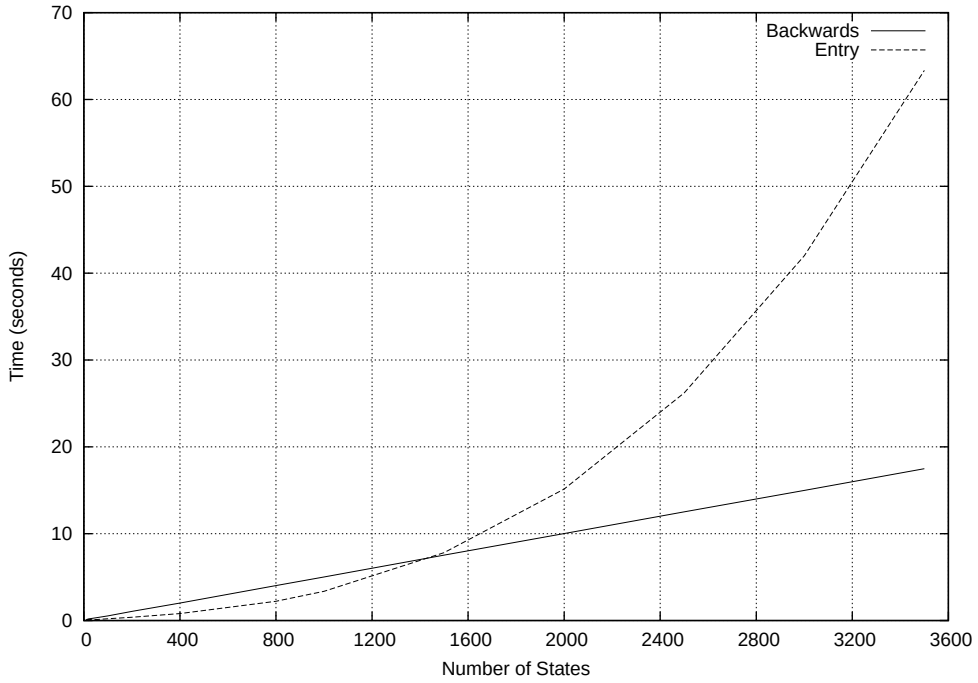


(b) A plot of the running time of the backwards and entry time approach.

Figure 5.6: The results of the scalability tests wrt. the number of clocks. The values in (a) are the mean value of ten runs and the plot in (b) is a plot of the E-Time and B-Time columns as functions over the Clock column.

States	E-RAM	B-RAM	D-RAM	E-TIME	B-TIME	D-TIME
10	6611	5261	10801	.10	.11	.10
20	8423	10766	20318	.12	.19	.10
50	13181	11014	21572	.11	.31	.11
100	23318	23292	23544	.21	.53	.20
200	27410	27032	27488	.38	1.05	.31
400	35516	35384	35516	.80	2.02	.61
800	54014	51116	51432	2.21	4.02	1.10
1000	63385	59329	59338	3.36	5.02	1.40
1500	97544	79380	79384	7.81	7.53	2.10
2000	131564	99225	99334	15.14	10.02	2.75
2500	169742	119386	119358	26.21	12.51	3.43
3000	211961	139344	139344	42.01	14.99	4.11
3500	258128	159382	159382	63.35	17.48	4.80

(a) The E-columns refer to the results from the entry approach, B-columns to those for the backwards algorithm and finally the D-columns the results from the dummy test. The unit of measure for memory usage is kilobytes and for time is seconds.



(b) A plot of the running time of the backwards and entry time approach.

Figure 5.7: The results of the scalability tests wrt. the number of states. The values in (a) are the mean value of ten runs and the plot in (b) is a plot of the E-Time and B-Time columns as functions over the States column.

time approach uses much more memory than the dummy test once the number exceeds 1500. We expect the entry time DBM is the cause.

Again the memory consumption is almost identical although with a slight advantage to the backwards approach. Keeping in mind that most of the information in the trace is not required by the entry time approach it appears it is actually the best memory wise.

The tests give no clear indication as to which of the algorithms is the best. It is highly dependent on the number of clocks and the number of states which is the fastest. Our test indicate the backwards approach should be used, whereas the entry time approach should be used when dealing with few states and many clocks.

5.5 Summary

Two approaches for finding concretizations have been proposed. Experiments have been conducted and indicate the entry time approach in many cases is both the fastest and best memory wise.
The efficiency of the entry time approach comes at the expense of generality. The entry time approach can for instance not find a concretization that reaches a specific valuation as the backwards solution is capable of. We shall see an application of this in the following chapter.

Chapter 6

Liveness Properties

In this chapter, we consider traces that represent infinite behaviour, i.e. liveness traces. A liveness trace is a symbolic trace that contains a cycle in its symbolic states, such that it is possible to iterate through the states of the cycle forever. Thus a symbolic trace representing infinite behaviour has no meaning unless it is pre-stable. Furthermore, we also require that the symbolic trace is post-stable. The following definition is influenced by Bouajjani et al. [11].

Definition 61
A symbolic lasso looping at point i is a symbolic trace $\langle \ell_1, D_1 \rangle \overset{g_1,r_1}{\rightsquigarrow} \dots \overset{g_{i-1},r_{i-1}}{\rightsquigarrow} \langle \ell_i, D_i \rangle \overset{g_i,r_i}{\rightsquigarrow} \dots \overset{g_{n-1},r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$ that is pre-stable, post-stable and where $D_i = D_n$ and $\ell_i = \ell_n$. $\diamondsuit$

We call $\langle \ell_1, D_1 \rangle \overset{g_1,r_1}{\rightsquigarrow} \dots \overset{g_{i-1},r_{i-1}}{\rightsquigarrow} \langle \ell_i, D_i \rangle$ the "head" of the lasso as this is only traversed once and $\langle \ell_{i+1}, D_{i+1} \rangle \overset{g_{i+1},r_{i+1}}{\rightsquigarrow} \dots \overset{g_{n-1},r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$ the "loop" of the lasso since it can be traversed infinitely many times.

6.1 Problem Definition

We are interested in finding a concretization of a symbolic lasso with delays that allow iterating through the loop forever. Finding delays that lead us through one or several iterations of the loop is not enough, as there is no guarantee that the same delays can be performed in the following iterations. Instead, we require that after traversing the loop a number of times, we return to the valuation in which we started.

Definition 62
A concretization of a symbolic lasso $\langle \ell_1, D_1 \rangle \overset{g_1,r_1}{\rightsquigarrow} \dots \overset{g_{i-1},r_{i-1}}{\rightsquigarrow} \langle \ell_i, D_i \rangle \overset{g_i,r_i}{\rightsquigarrow} \dots \overset{g_{n-1},r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$ looping at point i is a sequence

$$\begin{aligned}(\ell_1, v_1, d_1), \dots, (\ell_i, v_i, d_i), &(\ell_{i+1}, v^1_{i+1}, d^1_{i+1}), \dots, (\ell_n, v^1_n, d^1_n),\\ &(\ell_{i+1}, v^2_{i+1}, d^2_{i+1}), \dots, (\ell_n, v^2_n, d^2_n),\\ &\vdots\\ &(\ell_{i+1}, v^k_{i+1}, d^k_{i+1}), \dots, (\ell_n, v^k_n, d^k_n)\end{aligned}$$

where

1. $k \geq 1$,
2. $(v_i + d_i) = (v_n^k + d_n^k)$,
3. $(\ell_1, v_1, d_1), \ldots, (\ell_{i+1}, v_{i+1}^1, d_{i+1}^1)$ is a concretization of $\langle \ell_1, D_1 \rangle \overset{g_1, r_1}{\rightsquigarrow} \ldots \overset{g_i, r_i}{\rightsquigarrow} \langle \ell_{i+1}, D_{i+1} \rangle$,

and $\forall j$ such that $1 \leq j \leq k$

4. $(\ell_{i+1}, v_{i+1}^j, d_{i+1}^j), \ldots, (\ell_n, v_n^j, d_n^j)$ is a concretization of $\langle \ell_{i+1}, D_{i+1} \rangle \overset{g_{i+1}, r_{i+1}}{\rightsquigarrow} \ldots \overset{g_{n-1}, r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$,

and $\forall j$ such that $1 \leq j < k$

5. $(\ell_n, v_n^j, d_n^j), (\ell_{i+1}, v_{i+1}^{j+1}, d_{i+1}^{j+1})$ is a concretization of $\langle \ell_n, D_n \rangle \overset{g_i, r_i}{\rightsquigarrow} \langle \ell_{i+1}, D_{i+1} \rangle$. ♢

For notational convenience, we define the set of all concretizations over a symbolic lasso Ψ as

$$[\![\Psi]\!]^{lasso} \overset{def}{=} \{\psi \mid \psi \text{ is a concretization of } \Psi\}.$$

A concretization of a symbolic lasso contains the information that allows iterating through the symbolic loop forever by performing the same sequence of delays over and over. This is possible since we always return to the first valuation in the loop.

There is no guarantee that a concretization on this form exists for all traces.

Lemma 63
There exists a symbolic lasso $\Psi = \langle \ell_1, D_1 \rangle \overset{g_1, r_1}{\rightsquigarrow} \ldots \overset{g_{i-1}, r_{i-1}}{\rightsquigarrow} \langle \ell_i, D_i \rangle \overset{g_i, r_i}{\rightsquigarrow} \ldots \overset{g_{n-1}, r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$ looping at point i such that $[\![\Psi]\!]^{lasso} = \emptyset$. ♢

PROOF.
Let $\Psi = \langle \ell_0, D_1 \rangle \overset{y>0, \{x\}}{\rightsquigarrow} \langle \ell_0, D_2 \overset{y>0, \{x\}}{\rightsquigarrow} \langle \ell_0, D_2 \rangle$ be a symbolic lasso over the Timed Automaton in Figure 6.1(a) such that D_1 and D_2 are depicted in Figure 6.1. Proof by contradiction. Assume a concretization of Ψ exists. Then condition 2 of Definition 62 states that there exists a valuation $v \in D_2$ that v is revisited after iterating through the symbolic loop at least once. Assume v is such a valuation. Consider that in each iteration of the symbolic loop, y is reset, but to satisfy the guard, y must be greater than zero. Hence each iteration must add a delay $d > 0$. Since x is never reset and is increased, v does not repeat. ■

In the following, we restrict ourselves to symbolic lassos with certain properties and prove that there exists at least one concretization of those. To this end, we introduce a restricted set of difference constraints, in which all bounds are integers and non-strict.

Definition 64 (Non-strict Integer Constraints)
The set of all non-strict integer constraints $NDC(\mathcal{C}) \subseteq DC(\mathcal{C})$ is defined as

$$NDC(\mathcal{C}) \overset{def}{=} \{(x - y \leq n) \mid (x - y \leq n) \in DC(\mathcal{C}), n \in \mathbb{Z}\}.$$ ♢

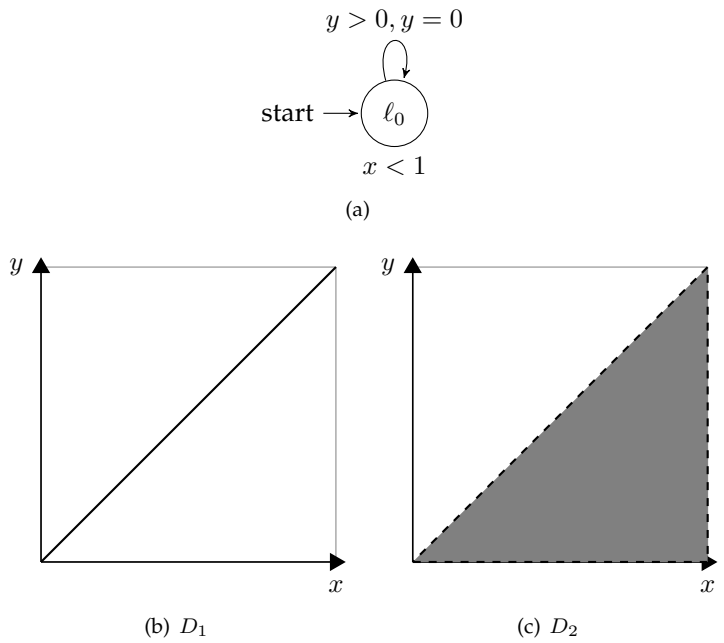


Figure 6.1: Let $\langle \ell_0, D_1 \rangle \overset{y>0,\{x\}}{\rightsquigarrow} \langle \ell_0, D_2 \overset{y>0,\{x\}}{\rightsquigarrow} \langle \ell_0, D_2 \rangle$ a symbolic lasso over the Timed Automaton in (a). (b) depicts D_1 and (c) depicts D_2. Note that the dashed lines represent strict bounds.

We define the set of zones $\mathcal{D}^{\mathcal{C}\leq}$ which can be created using $NDC(\mathcal{C})$ such that there exists an upper bound on each clock, formally

$$\mathcal{D}^{\mathcal{C}\leq} \overset{def}{=} \{[\Lambda] \mid \Lambda \subseteq NDC(\mathcal{C}) \text{ where for all } x \in \mathcal{C} \text{ there exists an } n \text{ such that } (x - \mathbf{0} \leq n) \in \Lambda\}.$$

To avoid symbolic lassos as the one constructed in Lemma 63, we restrict ourselves to considering symbolic lassos where all zones are in $\mathcal{D}^{\mathcal{C}\leq}$.

In the following we wish to consider only a finite subset of the valuations a zone describes. The subset we consider consists of all the valuations where the clocks are evaluated to integers.

Definition 65
A valuation v is an integer valuation if for all $x \in \mathcal{C} : v(x) \in \mathbb{Z}_{\geq 0}$. $\diamondsuit$

Consider a zone $D \in \mathcal{D}^{\mathcal{C}\leq}$ such that $D \neq \emptyset$. There exists at least one integer valuation in D. Furthermore, it is possible to add a strategy to FINDPOINT such that FINDPOINT(D) returns an integer valuation. This strategy consists of always assigning integer values to clocks. Since the constraints that describe D are non-strict and the bounds are integers this is always possible. We write FINDINTEGERPOINT when FINDPOINT is used with this strategy. Since delaying from one integer valuation to another requires an integer delay we also restrict ourselves to consider only integer delays.

We are now ready to state the main theorem of this chapter.

Theorem 66
Let $\Psi = \langle \ell_1, D_1 \rangle \overset{g_1,r_1}{\rightsquigarrow} \dots \overset{g_{i-1},r_{i-1}}{\rightsquigarrow} \langle \ell_i, D_i \rangle \overset{g_i,r_i}{\rightsquigarrow} \dots \overset{g_{n-1},r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$ be a symbolic lasso that loops at point i where $D_1 \subseteq \{v_0\}^{\uparrow}$ such that $v_0 \in D_1$, for all j, $1 \leq j < n$, $g_j \subseteq NDC(\mathcal{C})$ and for all j, $1 \leq j \leq n$, $D_j \in \mathcal{D}^{\mathcal{C}\leq}$. There exists a concretization $\psi = (\ell_1, v_1, d_1), \dots, (\ell_i, v_i, d_i), (\ell_{i+1}, v^1_{i+1}, d^1_{i+1}), \dots, (\ell_n, v^1_n, d^1_n), \dots, (\ell_{i+1}, v^k_{i+1}, d^k_{i+1}), \dots, (\ell_n, v^k_n, d^k_n)$ such that

- $\psi \in [\![\Psi]\!]^{lasso}$,
- $v_1 = v_0$,
- $d_1, \dots, d_i \in \mathbb{Z}_{\geq 0}$, and
- for all j, $1 \leq j \leq k$: $d^j_{i+1}, \dots, d^j_n \in \mathbb{Z}_{\geq 0}$. ♢

The following section is devoted to proving this theorem.

6.2 A Delay-Based Approach to Symbolic Lassos

In the effort of creating a concretization for a symbolic lasso, we define a function that given an integer valuation in a zone computes the maximal integer delay that can be added to this valuation.

Definition 67
Let $D \in \mathcal{D}^{\mathcal{C}\leq}$ be a zone and $v \in D$ a valuation. Let MAXDELAY be a function defined as

$$\text{MAXDELAY}(v, D) \overset{def}{=} \max \{d \mid d \in \mathbb{R}_{\geq 0} \text{ and } (v + d) \in D\}. \quad ♢$$

Note that if $D \in \mathcal{D}^{\mathcal{C}\leq}$ is a zone and $v \in D$ an integer valuation, then $\text{MAXDELAY}(v, D) \in \mathbb{Z}_{\geq 0}$. In this chapter, we will use MAXDELAY only when v is an integer valuation.
Consider how $\text{MAXDELAY}(v, D)$ can be computed when v is an integer valuation and $D \in \mathcal{D}^{\mathcal{C}\leq}$. Let H be a DBM in closed form such that $D = [H]$. Since $D \in \mathcal{D}^{\mathcal{C}\leq}$, H contains only non-strict constraints. Consider which constraints prevent us from delaying. Clearly the lower bounds do not. Consider the diagonal constraint $(x - y \leq n) \in D$. For any $d \in \mathbb{R}_{\geq 0}$ it holds that

$$(v(x) + d) - (v(y) + d) = v(x) - v(y) \leq n,$$

thus the diagonal constraints does not prevent us from delaying either. Hence to compute, only the upper bounds must be considered. The value of $\text{MAXDELAY}(v, D)$ can be computed as $\min_{x \in (\mathcal{C} \setminus \mathbf{0})} \{n - v(x) \mid (x - \mathbf{0} \leq n) \in H\}$. As the DBM contains exactly one upper bound for each clock, the value of this function can be computed in $\mathcal{O}(|\mathcal{C}|)$ time.

The MAXDELAY function has a useful property when considering pre-stable traces. Let $D \in \mathcal{D}^{\mathcal{C}\leq}$ and $D' \in \mathcal{D}^{\mathcal{C}\leq}$ such that D is pre-stable wrt. D' for g and r. By the definition of pre-stable, for any $v \in D$, there exists a delay such that it is possible to transit to D'. The we prove that MAXDELAY function returns such a delay.

Lemma 68
Let D, D' be zones such that D is pre-stable wrt. D' for g, r. If $v \in D$ is an integer valuation and $d = \text{MAXDELAY}(v, D)$, then $(v + d) \in D$, $(v + d) \vDash g$ and $(v + d)_{r=0} \in D'$. $\diamondsuit$

PROOF.
By the definition of MAXDELAY, $(v + d) \in D$. Since D is pre-stable wrt. D', there exists a delay d' such that $((v + d) + d') \in D$, $((v + d) + d') \vDash g$ and $((v + d) + d')_{r=0} \in D'$.

Consider what value d' can have. To satisfy $((v + d) + d') \in D$, d' must satisfy $d' \leq \text{MAXDELAY}(v + d, D) = 0$. Thus there exists only such value, namely $d' = 0$. As $((v + d) + d') = (v + d)$, we have that $(v + d) \in D$, $(v + d) \vDash g$ and $(v + d)_{r=0} \in D'$, hereby completing the proof.■

We now introduce the intuition for finding concretizations by example.

Example 69
Let $\langle \ell_0, D_1 \rangle \overset{y \leq 3, \{y\}}{\rightsquigarrow} \langle \ell_1, D_2 \rangle \overset{x \geq 2, \{x\}}{\rightsquigarrow} \langle \ell_2, D_3 \rangle \overset{y \geq 6, \{y\}}{\rightsquigarrow} \langle \ell_2, D_2 \rangle$ be the symbolic lasso over the Timed Automaton in Figure 6.2(a) such that D_1, D_2 and D_3 are depicted in Figure 6.2. Note that for all $D_1, D_2, D_3 \in \mathcal{D}^{\mathcal{C}\leq}$.

The goal is to find a valuation v in D_2 such that v reaches itself by iterating through the symbolic loop. A set, VISITED, containing valuations visited in D_2 is used to detect when a valuation that has been visited before is reached. Let v_2^1 be an integer valuation such that $v_2^1(x) = 1$ and $v_2^1(y) = 0$ be the initial integer valuation that is the result of FINDINTEGERPOINT(D_2). Add v_2^1 to VISITED and compute d_2^1 and v_3^1 as follows: Let $d_2^1 = \text{MAXDELAY}(v_2^1, D_2) = 2$ and $v_3^1 = (v_2^1 + d_2^1)_{r_2=0}$. By Lemma 68, $v_3^1 \in D_3$. In a similar fashion, d_3^1 and v_4^1 are computed. Then $d_3^1 = 3$, $v_4^1(x) = 3$ and $v_4^1(y) = 0$.

Let $v_2^2 = v_4^1$. Since $v_2^2 \notin$ VISITED, we add v_2^2 to VISITED and compute delays and valuations as we did before, this time starting from v_2^2. Then $d_2^2 = 0$, $d_3^2 = 3$, $v_4^2(x) = 3$ and $v_4^2(y) = 0$. Let $v_4^2 = v_2^3$ and see that $v_2^3 \in$ VISITED.

To create a concretization where v_2^3 repeats, consider which valuations and delays should be part of the concretization. Since $v_2^2 = v_2^3$, we only need to consider the valuations and delays that lead us from v_2^2 to v_2^3. This means that $v_2^1, d_2^1, v_3^1, d_3^1$ can be discarded. To create a concretization, we need valuations and delays for the head of the lasso such that v_2^2 is reached. This is computed using BACKWARDS$(\langle \ell_1, D_1 \rangle \overset{g_1, r_1}{\rightsquigarrow} \langle \ell_2, D_2 \rangle,\ v_2^2)$. Note that to use BACKWARDS, the input must be post-stable. $*$

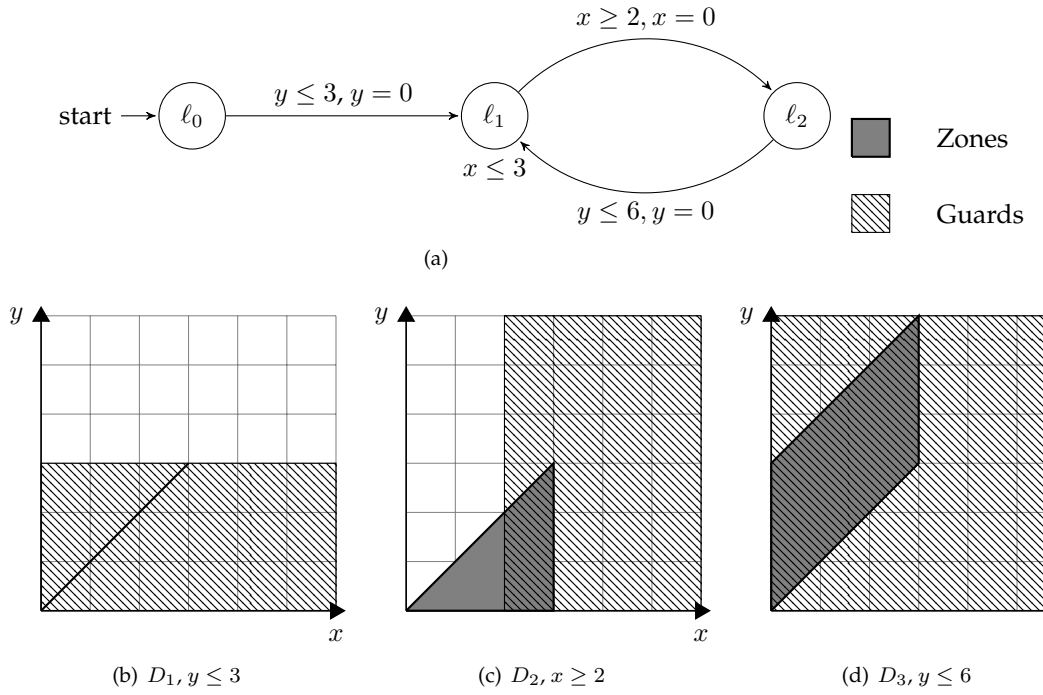


Figure 6.2: Let $\langle \ell_0, D_1 \rangle \overset{y \leq 3, \{y\}}{\rightsquigarrow} \langle \ell_1, D_2 \rangle \overset{x \geq 2, \{x\}}{\rightsquigarrow} \langle \ell_2, D_3 \rangle \overset{y \geq 6, \{y\}}{\rightsquigarrow} \langle \ell_2, D_2 \rangle$ be a symbolic lasso over the Timed Automaton in (a). (b) depicts D_1, (c) depicts D_2 and finally (d) depicts D_3.

The intuition exemplified in Example 69 is formalised in Algorithm 4.

Algorithm 4: Find a concretization for a symbolic lasso using valuations.

Input : A symbolic lasso $\Psi = \langle \ell_1, D_1 \rangle \overset{g_1,r_1}{\rightsquigarrow} \dots \overset{g_{i-1},r_{i-1}}{\rightsquigarrow} \langle \ell_i, D_i \rangle \overset{g_i,r_i}{\rightsquigarrow} \dots \overset{g_{n-1},r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$ that loops at point i, is post-stable and for all j, $1 \leq j \leq n$, $D_j \in \mathcal{D}^{\mathcal{C}\leq}$.

Output: A concretization $\psi = (\ell_1, v_1, d_1), \dots, (\ell_i, v_i, d_i), (\ell_{i+1}, v_{i+1}^j, d_{i+1}^j), \dots, (\ell_n, v_n^j, d_n^j), \dots, (\ell_{i+1}, v_{i+1}^k, d_{i+1}^k), \dots, (\ell_n, v_n^k, d_n^k)$ such that $\psi \in [\![\Psi]\!]^{lasso}$.

1 VISITED = $\emptyset$;
2 int $k = 0$;
3 v = FINDINTEGERPOINT(D_i) ;
4 $v = v +$ MAXDELAY(v, D_i) ;
5 **while** *for any* $j : (v, j) \notin$ VISITED **do**
6 $k = k + 1$;
7 add (v, k) to VISITED;
8 **for** $h = i + 1$ **to** n **do**
9 $v_h^k = v_{r_{h-1}=0}$;
10 d_h^k = MAXDELAY(v_h^k, D_h) ;
11 $v = v_h^k + d_h^k$;
12 **end**
13 **end**
14 let $(v, j) \in$ VISITED ;
15 $(\ell_1, v_1, d_1), \dots, (\ell_i, v_i, d_i) =$ BACKWARDS$(\langle \ell_1, D_1 \rangle \overset{g_1,r_1}{\rightsquigarrow} \dots \overset{g_{i-1},r_{i-1}}{\rightsquigarrow} \langle \ell_i, D_i \rangle, v)$;
16 **return**

$$(\ell_1, v_1, d_1), \dots, (\ell_i, v_i, d_i), (\ell_{i+1}, v_{i+1}^j, d_{i+1}^j), \dots, (\ell_n, v_n^j, d_n^j),$$
$$(\ell_{i+1}, v_{i+1}^{j+1}, d_{i+1}^{j+1}), \dots, (\ell_n, v_n^{j+1}, d_n^{j+1}),$$
$$\vdots$$
$$(\ell_{i+1}, v_{i+1}^k, d_{i+1}^k), \dots, (\ell_n, v_n^k, d_n^k)$$

Note that the VISITED structure contains pairs consisting of a valuation and an integer such that the valuation is in D_i. The integer is used to store the iteration of the while loop in which the valuation was added to visited. This information is used to discard the irrelevant valuations and delays from the concretization.

Theorem 70
Let $\Psi = \langle \ell_1, D_1 \rangle \overset{g_1,r_1}{\rightsquigarrow} \dots \overset{g_{i-1},r_{i-1}}{\rightsquigarrow} \langle \ell_i, D_i \rangle \overset{g_i,r_i}{\rightsquigarrow} \dots \overset{g_{n-1},r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$ be a symbolic lasso that loops at point i where for all j, $1 \leq j \leq n$, $D_j \in \mathcal{D}^{\mathcal{C}\leq}$. Then Algorithm 4 run on Ψ terminates and returns a concretization ψ such that $\psi \in [\![\Psi]\!]^{lasso}$. ♢

PROOF.
We first show that the valuations found in the while-loop are integer valuations and form concretizations of the symbolic loop. Consider the while-loop in line 5-13. It declares a sequence of

valuations on the following form:

$$
\begin{aligned}
&(\ell_{i+1}, v^1_{i+1}, d^1_{i+1}), \ldots, (\ell_n, v^1_n, d^1_n),\\
&(\ell_{i+1}, v^2_{i+1}, d^2_{i+1}), \ldots, (\ell_n, v^2_n, d^2_n),\\
&(\ell_{i+1}, v^3_{i+1}, d^3_{i+1}), \ldots, (\ell_n, v^3_n, d^3_n),\\
&\vdots
\end{aligned}
$$

Observe that this sequence is created by moving the valuation v through the symbolic loop. Initially, v is an integer valuation and $v \in D_i$. The for-loop in line 8-12 moves v from D_{h-1} to D_h by applying reset r_{h-1} to v and afterwards adding the maximal delay. Since $D_n = D_i$, it is always the case that D_{h-1} is pre-stable wrt. D_h for g_{h-1} and r_{h-1}. Hence by Lemma 68 the resulting valuation is always in D_h. Note that applying a reset or adding the maximal delay preserves that v is an integer valuation.

To prove that Algorithm 4 terminates, we must show that the while-loop does not run forever. The loop condition requires that there does not exist a pair that contains valuation. In every iteration of the while-loop, (v, k) is added to VISITED. Since there are only finitely many integer valuations in D_i, at some point, one of these valuations repeats.

Assume that Algorithm 4 returned $\psi =$

$$
\begin{aligned}
(\ell_1, v_1, d_1), \ldots, (\ell_i, v_i, d_i), &(\ell_{i+1}, v^j_{i+1}, d^j_{i+1}), \ldots, (\ell_n, v^j_n, d^j_n),\\
&(\ell_{i+1}, v^{j+1}_{i+1}, d^{j+1}_{i+1}), \ldots, (\ell_n, v^{j+1}_n, d^{j+1}_n),\\
&\vdots\\
&(\ell_{i+1}, v^k_{i+1}, d^k_{i+1}), \ldots, (\ell_n, v^k_n, d^k_n).
\end{aligned}
$$

We prove that $\psi \in [\![\Psi]\!]^{lasso}$.
Consider the conditions in Definition 62.

- Condition 1
  We must show that $j \leq k$. This follows since the value of j can never exceed the value of k.

- Condition 2
  We must show that $(v_i + d_i) = (v^k_n + d^k_n)$. This follows since $v = v^k_n + d^k_n$ when BACKWARDS is called with v as argument.

- Condition 4
  We must show that for all m where $j \leq m \leq k : (\ell_{i+1}, v^m_{i+1}, d^m_{i+1}), \ldots, (\ell_n, v^m_n, d^m_n)$ is a concretization of $\langle \ell_{i+1}, D_{i+1} \rangle \overset{g_{i+1}, r_{i+1}}{\rightsquigarrow} \ldots \overset{g_{n-1}, r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$. This follows by the observation made in the beginning of the proof.

- Condition 3
  We must show that $(\ell_1, v_1, d_1), \ldots, (\ell_{i+1}, v^j_{i+1}, d^j_{i+1})$ is a concretization of $\langle \ell_1, D_1 \rangle \overset{g_1, r_1}{\rightsquigarrow} \ldots \overset{g_{i+1-1}, r_{i+1-1}}{\rightsquigarrow} \langle \ell_{i+1}, D_{i+1} \rangle$. Note that $(\ell_1, v_1, d_1), \ldots, (\ell_i, v^1_i, d^1_i)$ is created by BACKWARDS, hence by Lemma 59 $(\ell_1, v_1, d_1), \ldots, (\ell_i, v^1_i, d^1_i)$ is a concretization of $\langle \ell_1, D_1 \rangle \overset{g_1, r_1}{\rightsquigarrow} \ldots \overset{g_{i-1}, r_{i-1}}{\rightsquigarrow} \langle \ell_i, D_i \rangle$. Combined with condition 4, this proves the case.

- Condition 5
  We must show that for all m such that $j \leq m < k : (\ell_n, v^m_n, d^m_n), (\ell_{i+1}, v^{m+1}_{i+1}, d^{m+1}_{i+1})$ is a

concretization of $\langle \ell_n, D_n \rangle \overset{g_i, r_i}{\rightsquigarrow} \langle \ell_{i+1}, D_{i+1} \rangle$. This also follows by the observation made in the beginning of this proof. ∎

We are now ready to prove Theorem 66.

PROOF.
Proof by construction. We run Algorithm 4 on Ψ. By Theorem 70 this algorithm terminates, and returns ψ. We show the following:

- $\psi \in [\![\Psi]\!]^{lasso}$.
  This follows by Theorem 70.

- For all m, $1 \leq m \leq k : d^m_{i+1}, \ldots, d^m_n \in \mathbb{Z}_{\geq 0}$.
  This follows trivially since MAXDELAY returns integers when called with an integer valuation.

- $v_1 = v_0$ and $d_1, \ldots, d_i \in \mathbb{Z}_{\geq 0}$.
  Recall that for all j, $1 \leq j < n$, $g_j \subseteq NDC(\mathcal{C})$ and for all j, $1 \leq j \leq n$, $D_j \in \mathcal{D}^{\mathcal{C}\leq}$. We use the following strategy in BACKWARDS.

  - Let all occurences of FINDPOINT be replaced by FINDINTEGERPOINT.
  - Let the last occurence of FINDPOINT choose the valuation v_0. ∎

  We must argue that this is always possible. Note that an integer valuation v is given to BACKWARDS as part of a strategy and that all zones of Ψ are in $\mathcal{D}^{\mathcal{C}\leq}$. It can be shown that pre_{action} and pre_{time} create zones in $\mathcal{D}^{\mathcal{C}\leq}$ as well, since all guards are subsets of $NDC(\mathcal{C})$. Thus all occurences of FINDPOINT can be replaced by FINDINTEGERPOINT . Recall that $D_1 \subseteq \{v_0\}^{\uparrow}$ such that $v_0 \in D_1$, hence it can be shown that last occurence of FINDPOINT can always return v_0.

Complexity We analyse the complexity of Algorithm 4.
Let Ψ be a symbolic lasso $\langle \ell_1, D_1 \rangle \overset{g_1, r_1}{\rightsquigarrow} \ldots \overset{g_{i-1}, r_{i-1}}{\rightsquigarrow} \langle \ell_i, D_i \rangle \overset{g_i, r_i}{\rightsquigarrow} \ldots \overset{g_{n-1}, r_{n-1}}{\rightsquigarrow} \langle \ell_n, D_n \rangle$ that loops at point i. Let n denote the length of the symbolic lasso. We split the complexity of the algorithm into the following parts:

for-loop Consider the complexity of the for-loop in line 8-11. The for-loop runs $\mathcal{O}(n)$ iterations, and the operations inside the loop can be performed in $\mathcal{O}(|\mathcal{C}|)$. The complexity of the for-loop is $\mathcal{O}(n \cdot |\mathcal{C}|)$.

while-loop First we consider the number of iterations the loop must perform before no longer satisfying the loop condition. The algorithm finds only integer valuations, hence we restrict ourselves to those. Let j be the largest integer present in an upper bound in D_i. Then there are $\mathcal{O}(j^{|\mathcal{C}|})$ integer valuations in the zone. Then the complexity of the while-loop becomes $\mathcal{O}(j^{|\mathcal{C}|} \cdot (n \cdot |\mathcal{C}|))$.

BACKWARDS As stated in Section 5.3, the complexity of Algorithm 3 is $\mathcal{O}(|\mathcal{C}|^3 \cdot n)$.

Combined, the complexity of the algorithm is $\mathcal{O}(j^{|\mathcal{C}|} \cdot (n \cdot |\mathcal{C}|) + |\mathcal{C}|^3 \cdot n)$ for any $j \geq 2$.

6.3 Experiments

In this section we test whether the approach is viable in practice by using Algorithm 4 method on different symbolic lassos.

Test method To test Algorithm 4, it is run on several symbolic lassos. The symbolic lassos used stem from UPPAAL, and are created by various models. In these models, we use invariants on all locations to ensure that the zones in the symbolic lasso belong to $\mathcal{D}^{\mathcal{C}\leq}$. The normal version of UPPAAL does not always return symbolic lassos on the form we expect, as the last zone of the lasso is not necessarily the same as the zone in the loop point of the lasso. Instead, an experimental version of UPPAAL has been used that returns symbolic lassos matching Definition 61. Note that the call to FINDINTEGERPOINT in Algorithm 4 has been made deterministic for the tests. It always returns the valuation where all clocks have their upper bounds as value.

In the experiments, the following values are measured:

- CPU-time used
- RAM-usage
- Number of iterations of the while-loop
  The number of times the while-loop has been performed indicates how hard it is to find a concrete loop-point. This differs from the length of the concretization, as the algorithm might perform many iterations of the while-loop before finding a valuation that repeats iself, and this valuation might be able to reach itself in fewer iterations. In the previous section, we argued that the number of iterations the while-loop performs is $\mathcal{O}(j^{|\mathcal{C}|})$, where j is the maximal constant in an upper bound of the DBM in the loop point. Meassuring this allows us to see whether this complexity is realistic.
- Length of the concretization
  From a user perspective, a concrete trace is easier to comprehend if the number of times the concretization iterates through the symbolic loop is as small as possible. Thus if

$$\begin{aligned}(\ell_1, v_1, d_1), \ldots, (\ell_i, v_i, d_i), &(\ell_{i+1}, v^1_{i+1}, d^1_{i+1}), \ldots, (\ell_n, v^1_n, d^1_n),\\ &(\ell_{i+1}, v^2_{i+1}, d^2_{i+1}), \ldots, (\ell_n, v^2_n, d^2_n),\\ &\vdots\\ &(\ell_{i+1}, v^k_{i+1}, d^k_{i+1}), \ldots, (\ell_n, v^k_n, d^k_n)\end{aligned}$$

  is a concretization for some symbolic trace, k is measured. Note that this differs from the number of iterations of the while-loop, since some concretizations of the symbolic loop are not included in the final output.

A dummy test has been implemented that loads the trace and performs BACKWARDS. Measuring the time this takes allows allows us to compute the while-loops running time.

The tests are split into 3 parts.

- Scalability wrt. the number of states
  We test how the number of states in a symbolic trace affects the performance of the algorithm. A parameterised state model is used, such that the symbolic lassos returned have a fixed number of clocks and the amount of states varies according to a parameter. The parameterised model is depicted in Figure 6.3.

- Scalability wrt. the number of clocks
  We test the performance of the algorithm with a different number of clocks in each zone. A parameterised model is used, such that the number of clocks in the symbolic lassos returned varies according to the parameter given. The number of states in this model is fixed. The model is depicted in Figure 6.4.

- Performance on lassos created by ordinary models
  In this test we wish to test the algorithm on lassos from ordinary models to meassure the lenght of the concretizations and the number of iterations of the while-loop, as the complexity of such models is assumed to be greater than parameterised models used in other tests. The first symbolic lasso is created by a model based on the "gossiping girls"-problem [2]. The second symbolic lasso is created by a model of a mutual exclusion algoritm by Lamport [17]. This algorithm allows starvation, and the query used to generate the model is "does requesting access to the critical section always lead to obtaining permission to enter it?"

All tests are run 10 times.

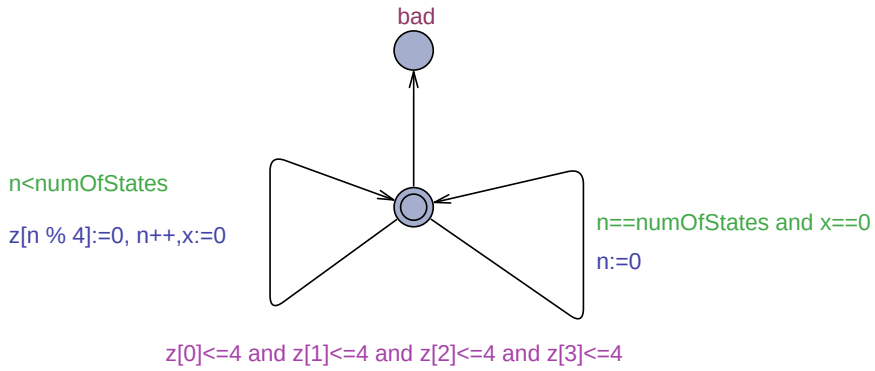


Figure 6.3: The UPPAAL model used for testing scalability wrt. the number of states. In the model, n is an integer, x is a clock and z is an array consisting of 4 clocks. The parameter $numOfClocks$ is adjusted to increase the number of states. The query asked is whether the *bad* location is always reached.

Test results The results of all tests are shown in Table 6.1. Note that all results are mean values of 10 runs.

- Scalability wrt. the number of states
  The results are in Table 6.0(a).

- Scalability wrt. the number of clocks
  The results are in Table 6.0(b).

- Performance on ordinary models
  The results are in Table 6.0(c).

(a) Result of scalability tests wrt. the number of states

States	L-RAM	D-RAM	L-Time	D-Time	While	Length
405	23000	23094	.41	.40	2	1
805	26448	26421	.74	.70	2	1
1205	30011	29730	1.09	1.00	2	1
1605	33406	33031	1.40	1.31	2	1
2405	40211	40227	2.09	2.00	2	1
2805	43646	43631	2.38	2.30	2	1
3205	46892	46876	2.71	2.60	2	1
4005	53956	53678	3.37	3.23	2	1

(b) Result of scalability tests wrt. the number of clocks

Clocks	States	L-RAM	D-RAM	L-Time	D-Time	While	Length
11	121	37048	37048	2.50	2.41	1	1
21	121	77376	77376	13.28	13.03	1	1
31	121	141332	141332	39.19	38.40	1	1
41	121	228884	228884	86.69	84.31	1	1

(c) Results for ordinary models

Model	States	Clocks	L-RAM	D-RAM	L-Time	D-Time	While	Length
gossip	15	2	18115	14426	.13	.11	1	1
lamport_mutex	18	2	8335	12311	.12	.10	1	1

Table 6.1: The L columns refer to results from the Algorithm 4 and the D columns refer to the results of the dummy solver. Memory usage is measured in kilobytes and time in seconds. The While column refers to the number of iterations the while loop has performed. The Length column refers to the length of the concretization.

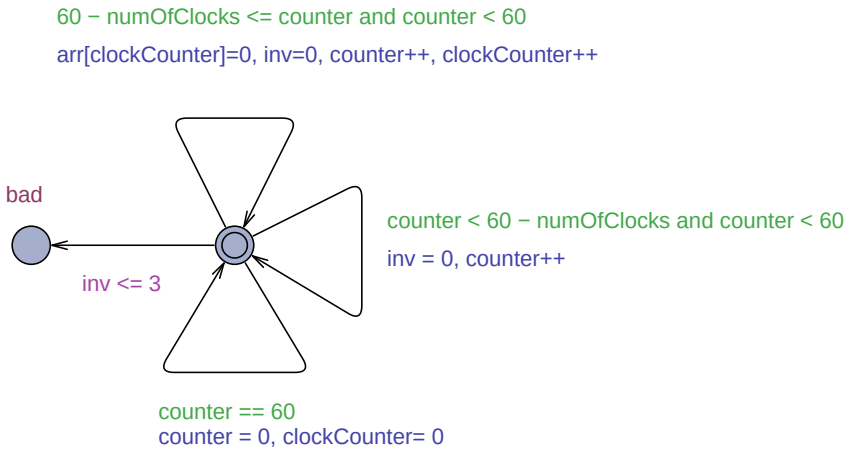


Figure 6.4: The UPPAAL model used for testing scalability wrt. the number of clocks. The model contains two counters - $clockCounter$ and $counter$ - that are used to count the number of iterations performed. The invariant clock inv is used to force an upper bound on each zone. The parameter $numOfClocks$ is used to adjust the number of clocks in the model. The array arr contains $numOfClocks$ clocks. The model increases the counter 60 times before resetting it to 0. In the last $numOfClocks$ iterations, these clocks in arr are reset. The query asked is whether the *bad* location is always reached.

Discussion We discuss each test.

- Scalability wrt. the number of states
  Both RAM-usage and time spent increases linearly in the number of states for both Algorithm 4 and the dummy solver. Note that the RAM-usage for both the dummy and Algorithm 4 is the same, hence the while-loop of Algorithm 4 require less RAM than loading the trace and computing BACKWARDS on the head of the symbolic lasso. The difference in the computation time of Algorithm 4 and the dummy solver is only a fraction of the time used by the dummy solver. This implies that the while-loop in Algorithm 4 is faster than loading the trace and performing BACKWARDS. Note though that only one iteration of the while loop was performed. If a symbolic lasso requires several iterations in the while loop, the time spend would increase.

- Scalability wrt. the number of clocks
  The RAM-usage increases non-linearly for both the dummy solver and Algorithm 4. This implies the RAM-usage is not caused by the while-loop in Algorithm 4, as the increase also happens in the dummy solver. The non-linear increase in both RAM-usage and CPU-time is the result of the increase in the number of clocks. Note that the number of iterations of the while-loop has doubled, hence the time spent on the while-loop has increased. Still, the time spent in the while-loop is only a fraction of the entire computation.

- Performance on lassos created by ordinary models
  Algorithm 4 is fast on both lassos and it seems that this approach is a viable solution for liveness traces. In both cases only one iteration was of the while-loop was performed. It seems that the models chosen do not create more complex lassos than the parameterised models used for the previous tests. The length of the concretizations is also one in both

cases. The results suggest that there exists a concretization of length one for all traces, but a more thorough test where more models are used might reveal that this is not the case.

Chapter 7

Implementation

The algorithms presented in Chapter 3, 5 and 6 have been implemented in the prototype tool Concrete Traces for UPPAAL (CTU). The tool has been designed to work in conjunction with UPPAAL and generates concretizations from symbolic traces and lassos output by UPPAAL.

In this chapter, we discuss its overall structure, how it is used and its limitations.

7.1 Structure of CTU

CTU is as a stand-alone tool, that has been designed with extendability in mind. The extendability has been achieved by splitting CTU into modules that each have a specific task. We first describe the general modules and their tasks.

rationalDBM The `rationalDBM` module provides a DBM library which allows rational numbers as bounds. The module provides closed form preserving implementations of the zone operations presented in Chapter 4.
The module also implements different versions of the FINDPOINT algorithm. The different versions are distinguished by the strategy they apply. In one implementation, the smallest value possible is selected for each clock and in another implementation, the maximal value is selected. Finally one implementation only minimises the value of one clock, with respect to some $\varepsilon \in \mathbb{Q}_{>0}$. This is typically used to minimise the elapsed clock.

symbTrace The `symbTrace` module contains the internal format of both symbolic traces and lassos. The representation is essentially a double linked list of transitions and symbolic states. Besides a location and a DBM, each symbolic state contains the invariant of the location. Each transition contains a guard and reset. Guards are represented as both DBMs and individual constraints. This design decision was made as the algorithms require different representations.

xmlloader The `xmlloader` module is responsible for loading a symbolic trace or lasso into the internal format. The xmlloader is currently very restrictive with respect to the syntax of guards and resets.
When loading the DBMs, this module must know which clock is the **0** clock, as this clock is treated differently than the other clocks. The `xmlloader` relies on the main module to pass it the id of the **0** clock.

Each approach to finding concretizations is contained in a so-called solver module. We describe the solver modules and their tasks.

entry The `entry` solver implements the entry time approach from Chapter 5. The implementation of the entry time approach has been divided into three distinct parts:

- Create last-reset-at look-up table.
  During this step a look-up table is created that is used to simulate the last-reset-at function. Each row in this table represents a symbolic state and each column a clock. The entry in row i, column x contains $lra(\Psi, i, x)$.
- Create entry time constraints.
  This part constructs entry time constraints from a forward reachability trace. It uses the last-reset-at look-up table. The constraints are then added to a DBM and it is closed. The DBM is represented using the UPPAAL DBM library, as the constraints contain only integers.
- Find Solution.
  The entry time constraints are solved using an implementation of FINDPOINT. The strategy used in FINDPOINT is as follows. First, the entry time of the last state is minimised with respect to ε. Afterwards the entry times of the remaining sates are minimised with respect to $\frac{\varepsilon}{n}$, where n is the number of states. These entry times are minimised in increasing order.

From a readability perspective, a concretization containing many integer delays is preferred over one with few integer delays. The `entry` solver allows choosing integer delays. The integer delays are selected using a greedy approach i.e. whenever possible an integer delay is chosen.

back This solver implements Algorithm 3 presented in Chapter 5. The following strategy for choosing valuations is used in this implementation.
oIn line 1 the value of the elapsed clock is minimised with respect to the supplied $\varepsilon \in \mathbb{Q}_{>0}$. In lines 2, 6 and 10 the elapsed clock is maximised with respect to $\frac{\varepsilon}{n}$, where n is the number of states in the trace. The elapsed clock is minimised in lines 5 and 9 with respect to $\frac{\varepsilon}{n}$, where n is the number of symbolic states in the trace.
The strategy used ensures that the delays in each symbolic state become as small as possible with respect to $\frac{\varepsilon}{n}$.

live The `live` solver implements the Algorithm 4 presented in Chapter 6. The strategy for the initial call to FINDINTEGERPOINT in line 3 is to maximise all clocks. In line 15, Algorithm 3 is called. In Algorithm 3, the strategy for the `back` solver is used, with the exception that the valuation given as an argument is chosen in line 1.

7.2 Using CTU

CTU is called from a command line interface. Calling CTU follows the structure
`ctu [options] trace [solver-options]`
where trace is a file containing a symbolic trace. The important command line arguments are explained below.

```
-e n      Set precision for fastest epsilon trace to 1/n.

-f        Output the result as floating points - computations
  are still performed using rationals.

-n        Output entry times instead of delays.

-g clock  Inform CTU that 'clock' is the elapsed clock.

-z zero   Inform CTU that 'zero' is the zero clock.

-s solver Set 'solver' as the solver to be used.
```

The option `-z` has been added to make CTU independent of the naming convention used in a symbolic trace or lasso. It allows CTU to identify the **0**-clock. If this value is not specified, CTU uses the default value `sys.t(0)`, which is the representation used in UPPAAL.
Similarly, the option `-g` has been added to help CTU identify the elapsed clock. The default value is `sys.#tau`, as in UPPAAL.

The `-s` option selects a specific solver to be used. The solver may be any of the following modules `entry`, `back` or `live`.

7.3 Limitations

Being a prototype, CTU has certain limitations. The input to CTU is generated by UPPAAL, which extends classic Timed Automata with a wide range of modelling features. Many of these are not supported by CTU.
Examples of these features include the C-like language that can be used to create small functions. CTU supports evaluating clocks against simple arithmetic expressions but not against return values from function calls.
The semantics of committed and urgent locations are not supported in CTU either. It is however possible to modify a model using these features such CTU can find concretizations for the symbolic traces and lassos created by it. The modification involves adding a clock that is reset upon entering a committed/urgent location forcing time to stop by adding an invariant to the committed/urgent location. Hereby the zones and constraints reflect the fact that delaying is not possible in these locations.

Chapter 8

Conclusion

Tools such as UPPAAL verify safety and liveness properties for Timed Automata. If a property is violated, UPPAAL returns a symbolic trace/lasso leading to the violation. A symbolic trace/lasso assists the user in understanding the behaviour of the system and in determining the cause of the violation. Unfortunately, symbolic traces/lassos are hard to comprehend, as they represent the timing aspects of a system using Difference Bound Matrices. Instead, a so-called concretization that contains concrete delays is wanted.
For symbolic traces representing finite behaviour, our contribution is providing two algorithms that create concretizations of these. We have proven that these algorithms are sound and complete and tested them against each other. The results showed that the backwards approach is fastest on symbolic traces with many states and few clocks, whereas the entry time approach is fastest on symbolic traces with many clocks and few states. We have also tested both approaches on a large industrial case and conclude both are viable.
For symbolic lassos representing infinite behaviour, we have considered the subclass of symbolic lassos where only non-strict timing constraints are present and upper bounds always exist. Our contribution is providing an algorithm that finds concretizations of such symbolic lassos. The algorithm finds concretizations that only contain integer valuations and integer delays, which makes the concretizations easy to comprehend. We have tested this approach and found that the length of the concretizations in all our tests was one. Furthermore, the symbolic loop was never traversed more than two times. The approach has been tested on a few symbolic lassos created from ordinary models and we conclude that the approach is viable for such models.
We have used an algorithm to find valuations that satisfy a Difference Bound Matrix. Our contribution to this algorithm is proving its correctness.
A well-known result for Timed Automata is that the operations performed during a symbolic state space exploration can be performed using Difference Bound Matrices. We provide a proof of this result for the case where strict constraints are allowed.

8.1 Future Work

In the following we discuss possible future work.

Backwards approach - avoid costly zone operations The backwards approach relies on zone operations that require closed form. Computing the action and time predecessor require the use

of the intersection operator, which is used five times in each iteration. Computing the intersection is cubic in the number of clocks, and slow down the implementation. Future work involves reducing the number of times the intersection operator is used to improve the implementation.

Symbolic traces - find the maximal number of integer delays Integer delays make concretizations easier to comprehend for users. A future work would be to create an algorithm that finds concretizations of symbolic traces that contain the maximal number of integer delays possible or establish the complexity of the problem.

Integrate in UPPAAL Concretizations are currently generated by our tool prototype CTU. Integrating the algorithms into UPPAAL would provide multiple benefits. UPPAAL displays symbolic traces in a simulator and delays from a concretization could be added to this simulation. Furthermore, loading of the symbolic traces and lassos can be avoided, as UPPAAL has computed all the information. This would boost the performance a lot, since symbolic traces with many clocks often take a long time to load.

Adapt safety approaches to Linear Priced Timed Automata Linear Priced Timed Automata are Timed Automata extended with linear cost rates on locations and a fixed cost on edges. An interesting problem for Linear Priced Timed Automata is finding cost-optimal runs, and a future work is generalising the approach for safety traces to Linear Priced Timed Automata, such that the concretizations found describe cost-optimal runs.

Coping with state space reduction techniques In this thesis we have assumed the zones are calculated as prescribed by the symbolic semantics for Timed Automata. In reality, model checking tools uses state space reduction techniques. Unfortunately these techniques have created a gap between the output of UPPAAL and the input our algorithms expect. In the future one could attempt to close this gap.

Symbolic lassos - prove the existence of a concretization of length one Our experiments seem to indicate that there always exist a concretization of length one for symbolic lassos that only contain non-strict timing constraints and upper bounds always exist. A future work is to prove or disprove that this is the case.

Liveness - delay strategy There exist symbolic lassos for which there does not exist a concretization. A future work involves describing the concrete delays that these symbolic lassos can perform and creating an algorithm that computes them.

Bibliography

[1] *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS '97), December 3-5, 1997, San Francisco, CA, USA*, 1997. IEEE Computer Society.

[2] Luca Aceto, Anna Ingólfsdóttir, Kim Guldstrand Larsen, and Jiri Srba. *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, New York, NY, USA, 2007. ISBN 0521875463.

[3] Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In Mike Paterson, editor, *ICALP*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990. ISBN 3-540-52826-1.

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

[5] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008. ISBN 026202649X, 9780262026499.

[6] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[7] Johan Bengtsson. *Clocks, DBMs and States in Timed Systems*. PhD thesis, Uppsala University, Department of Information Technology, 2002.

[8] Johan Bengtsson. memtime. `http://www.update.uu.se/~johanb/memtime`.

[9] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2003. ISBN 3-540-22261-8.

[10] Johann Bengtsson and Fredrik Larsson. UPPAAL A Tool for Automatic Verification of Real-Time Systems. Master's thesis, Uppsala University, Department of Information Technology, 1996.

[11] A. Bouajjani, S. Tripakis, and S. Yovine. On-the-fly symbolic model checking for real-time systems. In *RTSS '97: Proceedings of the 18th IEEE Real-Time Systems Symposium* DBL [1], page 25. ISBN 0-8186-8268-X.

[12] David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In Joseph Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1989. ISBN 3-540-52148-8.

[13] Uli Fahrenberg, Kim G. Larsen, and Claus R. Thrane. Verification, performance analysis and controller synthesis for real-time systems. In Manfred Broy, Wassiou Sitou, and Tony Hoare, editors, *Engineering Methods and Tools for Software Safety and Security*, volume 22 of *NATO Science for Peace and Security Series - D: Information and Communication Security*, pages 203–229. IOS Press, 2009. ISBN 978-1-58603-976-9.

[14] Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345, 1962.

[15] Klaus Havelund, Arne Skou, Kim Guldstrand Larsen, and K. Lund. Formal modeling and analysis of an audio/video protocol: an industrial case study using uppaal. In *IEEE Real-Time Systems Symposium* DBL [1], pages 2–13.

[16] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Inf. Comput.*, 111(2):193–244, 1994.

[17] Leslie Lamport. A fast mutual exclusion algorithm. *ACM Trans. Comput. Syst.*, 5(1):1–11, 1987.

[18] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *STTT*, 1(1-2): 134–152, 1997.

[19] Danny Bøgsted Poulsen and Jonas van Vliet. Concrete Traces for UPPAAL. DAT5 Project Report, 2009. URL `http://www.cs.aau.dk/~dannybp/ConcreteTracesForUPPAAL.pdf`.

[20] Tomas Gerhard Rokicki. *Representing and modeling digital circuits*. PhD thesis, Stanford, CA, USA, 1993.

[21] UPPAAL team. UPPAAL DBM library. `http://www.cs.aau.dk/~adavid/UDBM/index.html`.

[22] Sergio Yovine. Kronos: A verification tool for real-time systems. *STTT*, 1(1-2):123–133, 1997.

Thesis Summary

In recent years, model checking has attained a prominent position in the formal verification of reactive systems. Model checking aims at verifying that a system adheres to its specification. Hereby model checking attempts to solve one of the main concerns within software development - software verification.
Model checking tools have been developed to verify whether a model of a system adheres to its specification. One such tool is UPPAAL, which verifies properties for Timed Automata. The verification is based on a state space exploration, but the state space is often infinite. To allow effective and efficient verification, abstractions over the state space have been introduced. UPPAAL uses symbolic semantics, a coarse abstraction over the state space in which time is described through the notion of zones.
A zone is a set of possible time valuations. Difference Bound Matrices contain difference constraints and are often used to describe zones in model checkers for time-formalisms. We have proven an algorithm for finding valuations in a Difference Bound Matrix correct.
If a property is violated, UPPAAL often provides a diagnostic trace as proof of the violation. Unfortunately, these diagnostic traces are subject to the same abstraction as the state space, hence they are hard for users to understand. From a user perspective, a concrete diagnostic trace that proves the violation is wanted. A concrete diagnostic trace contains concrete time values in the form of delays and valuations.
In this thesis, we divide the diagnostric traces into two disjoint settings: Those that violate safety properties and those that violate a liveness properties.
For diagnostic traces violating safety properties, we have provided two approaches for finding concrete diagnostic traces. The first approach analyses the edges and invariants of the Timed Automaton and derives entry time constraints. A solution to these constraints is then found and used to generate a concrete diagnostic trace. The second approach considers the zones in the diagnostic trace and generates a concrete diagnostic trace in a backwards manner. Both approaches were adapted to solve the fastest trace problem, which states how to reach the violation in shortest possible time. Furthermore, the approaches were tested against each other.
A diagnostic trace violating a liveness property allows infinite behaviour and contains a loop. We restrict ourselves to considering the traces where all zones are based on non-strict integer constraints. A concrete diagnostic trace violating a liveness property then contains a valuation that is visited infinitely often. This valuation is found by performing maximal delays in each zone. We have shown that these concrete diagnostic traces only contain integer delays, which is useful, as this is easy to comprehend for a user.