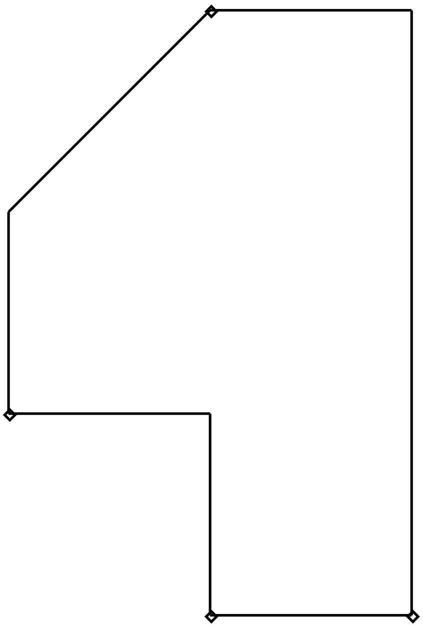
Reachability Analysis for Timed Automata using Max-Plus Algebra



DAT5, Fall 2010
Group d504a
7th semester
Department of Computer Science
Aalborg University
January 17th 2011

Title:

Reachability Analysis for Timed Automata using Max-Plus Algebra

Theme:

Model checking

Period:
DAT5, Fall 2010

Group:
d504a

Group members:
Qi Lu

Michael Madsen

Martin Milata

Søren Ravn

Supervisors:
Uli Fahrenberg
Kim Guldstrand Larsen

Circulation: 7

Pages: 34

Ended: January 17th 2011

Abstract:

In this report, we elaborate on the idea of using max-plus polyhedra as a data structure for the clock space in reachability analysis of timed automata. Drawing inspiration from the extensive work that has been done on difference bound matrices, as well as previous work on max-plus polyhedra in other areas, we have developed the remainder of the algorithms needed to perform forward and backward reachability analysis using max-plus polyhedra. Furthermore, we give proofs that the algorithms are correct according to the definition of the operations. Finally, to show that the approach works in practice and theory alike, we create a proof-of-concept implementation on top of the model checker opaal.

PREFACE

This project began September 7th 2010 and ended January 17th 2011. The report documents the process of performing reachability analysis on timed automata using max-plus algebra.

Inlined code or words that refer to some implementation aspect will be shown in a `typewriter` font. Concepts and words of particular interest are *emphasized* when first used in a chapter.

References to figures, tables and listings will contain a given number in the form of the chapter number and counter, for example, the first figure in Chapter 4 will be referenced as Figure 4.1.

Citations will appear in Harvard style in round parentheses, using the last name of the author and the year of publication, e.g. (Author et al., 2009). Inline citations will appear as e.g. Author et al. (2009).

The report is divided into the following chapters:

Chapter 1 contains an introduction to the problem we deal with, and how we propose to solve it.

Chapter 2 provides background theory supporting the topics dealt with in the report.

Chapter 3 contains algorithms for operations on max-plus polyhedra and their proofs.

Chapter 4 contains an overview of our implementation of this approach.

Chapter 5 concludes the report and discusses future work and possible improvements.

Lastly, we would like to thank Mads Chr. Olesen and Kenneth Yrke Jørgensen for their work on the opaal model checker, and their willingness to assist us in implementing this approach, Alexandre David for his insight into UPPAAL, and Jesper Dyhrberg for his previous collaboration on this topic.

CONTENTS

CHAPTER 1

INTRODUCTION

A real-time system is a system where total completion of a task depends not only on whether the logic is correct, but also the time at which it is performed. Examples of real-time systems include airbags, pacemakers, live video streaming, video game systems, and production lines.

A key problem when developing a real-time system is to ensure correctness of the system. For that purpose, it is useful to construct a model of the system and verify certain properties directly on the model. This is known as *model checking*, and when applied to real-time systems, it is called *real-time model checking*. Zone-based reachability analysis of timed automata is a well-established technique for performing real-time model checking, and several tools exist for this purpose, e.g. Uppaal (Uppsala University and Aalborg University, 2010) and Kronos (Yovine, 1997).

Zones are an abstraction of a set of states, which allows reduction of the number of states that need to be considered during analysis. A common representation of zones are Difference Bound Matrices, or DBMs for short (Alur et al., 1992) , which have the same expressive power as zones. However, they are not perfect: for good performance, one must be able to determine if a given zone has already been handled, to minimize the risk of state-space explosion. This can normally be done by simply taking the union of all states visited and performing inclusion checking to check if a zone has been visited completely, but DBMs are not capable of representing exact unions, as zones are not closed under union. Instead, an overapproximation must be performed, and then followed up by an exact analysis if necessary.

One way to handle this issue is to use a different data structure. We propose to use *max-plus polyhedra*, which have been shown capable of providing large performance improvements for selected problems in static analysis (Allamigeon et al., 2009, 2010). Max-plus polyhedra are more expressive than DBMs, and capable of creating better overapproximations than DBMs, which would minimize the need to perform re-analysis.

CHAPTER 2

PRELIMINARIES

This chapter explains the preliminary notions that are used in the rest of the report. The first section introduces timed automata, whereas the second section is concerned with the definition of max-plus algebra and max-plus polyhedra. Finally, the third section describes difference bound matrices, and provides an example to illustrate the difference between max-plus polyhedra and DBMs.

2.1 Timed automata

Timed automata can be thought of as finite automata that can interact with a number of real-valued clocks. They have proven to be useful in modelling and verification of real-time systems. Timed automata were introduced by Alur and Dill (1994); a more thorough overview of relevant notions can be found for example in (Aceto et al., 2007).

2.1.1 Syntax

First, let X be a finite set of real-valued non-negative variables referred to as *clocks*. Define $\mathcal{B}(X)$ to be set of all *clock constraints* g generated by following grammar:

$$g ::= x_1 \sim n \mid x_1 - x_2 \sim n \mid g_1 \wedge g_2,$$

where $x_1, x_2 \in X$, $n \in \mathbb{N}$ is a natural number and $\sim \in \{<, \leq, =, \geq, >\}$.

A *timed automaton* is a quintuple (L, X, l_0, E, I), where

- L is a finite set of *locations*,
- X is a finite set of clocks,
- $l_0 \in L$ is the *initial location*,
- $E \subseteq L \times \mathcal{B}(X) \times 2^X \times L$ is the set of *edges* and
- $I : L \to \mathcal{B}(X)$ a function which assigns to every location an *invariant*.

Instead of (l_1, g, r, l_2), we will write $l_1 \xrightarrow{g,r} l_2$. Here, l_1 is the source location of the edge, l_2 the destination, g is the guard of the edge and r the set of clocks to be reset after transition.

Strict constraints We are currently only able to represent conjunctions of non-strict constraints using max-plus polyhedra. While the external representation of polyhedra allows simple extension to strict inequalities, it is not the case with the internal representation which we use. This means that in the rest of the report (excluding the preliminaries), we use the definition of $\mathcal{B}(X)$ such that $\sim \in \{\leq, =, \geq\}$, which results in weaker model of timed automata that is still interesting. See section 5.2 on page 30 for further discussion.

2.1.2 Semantics

We will first try to give a very informal description of the semantics of timed automata. The state of a timed automaton is composed of its control location and values of each clock. In the initial state, the location is set to l_0 and the value of all clocks is zero. Whenever the automaton is in some control location l, it has two choices.

1. Similarly to finite state automata, it can do a transition over one of the edges (l, g, r, l') that lead from it. However, this is only possible when the current clock values satisfy the guard g of the edge. After the transition, all the clocks in the set r of the edge are reset to zero. Additionally, the invariant of the destination state must be satisfied by the clock valuation after resetting.

2. It can "stay" in location l for some period of time. This means that all the clocks increase by the same amount of time. The values of the clocks must satisfy the location invariant $I(l)$ during this whole period.

To formally define the semantics of a timed automaton, we must first define *clock valuations*. A clock valuation v is a function $v : X \to \mathbb{R}_{\geq 0}$, that assigns a non-negative value to each clock. Let $\delta \in \mathbb{R}_{\geq 0}$ and $r \subseteq X$. We will define $v + \delta$ to be the valuation such that $(v + \delta)(c) = v(c) + \delta$ and $v[r]$ to be the valuation such that $v[r](c) = 0$ whenever $c \in r$ and $v[r](c) = v(c)$ otherwise.

Formally, the semantics of a timed automaton (L, X, l_0, E, I) is given in terms of a transition system[1] $(S, s_0, \to)$:

- $S = \{(l, v) \mid l \in L, v \in \mathbb{R}^X_{\geq 0}, v \vDash I(l)\}$ is the set of states,
- $s_0 = (l_0, v_0)$, where v_0 is the clock valuation that assigns 0 to all clocks, is the initial state,
- $\to \subseteq S \times S$ are transitions such that:
  - $(l, v) \to (l', v')$ if $l \xrightarrow{g,r} l'$, $v \vDash g$, $v' = v[r]$,
  - $(l, v) \to (l, v + \delta)$ for all $\delta \in \mathbb{R}_{\geq 0}$ such that $v + t \vDash I(l)$ for any $0 \leq t \leq \delta$.

Such a transition system is in most cases infinite and even uncountable, which means it cannot be directly used in an algorithm. Fortunately, we can also construct transition systems which are finite – this is achieved by replacing individual states with *symbolic states*, where each such state consists of a control location and a set of clock valuations. We naturally require those sets to have a finite description and the clock valuations contained in them to be in some way equivalent, which usually means that they have to be *untimed bisimilar*, see Aceto et al. (2007).

An example of such symbolic semantics is the *region graph*, where the sets of clock valuations are divided into equivalence classes based on integer parts, orderings of fractional parts of clock

[1] If we label the edges of the timed automaton, this becomes a labelled transition system. Labeling of edges is, however, not useful for our purpose.

values and whether or not the value is greater than some fixed constant. The number of such classes grows very quickly with the number of clocks, hence the region graph is not really suitable for algorithmic use. Nevertheless, region graphs have been shown to be useful when proving decidability of some properties of timed automata.

There is, however, another variant of symbolic semantics, which is suitable for practical use.

2.1.3 Zones and zone graphs

Zones are sets of clock valuations that satisfy a conjunction of clock constraints. Formally, $Z \subseteq \mathbb{R}_{\geq 0}^{X}$ is a zone if there is a $g \in \mathcal{B}(X)$ such that $Z = \{v \mid v \vDash g\}$. Zones can also be thought of as convex subsets of $|X|$-dimensional Euclidean space.

In order to define transitions on symbolic states of the form (l, Z), where l is a location and Z is a zone, we again need to define some operations first. Let Z be a zone, $g \in \mathcal{B}(X)$ and $r \subseteq X$.

- $Z \wedge g = \{v \in Z \mid v \vDash g\}$,
- $Z^{\uparrow} = \{v + \delta \mid v \in Z, \delta \in \mathbb{R}_{\geq 0}\}$,
- $Z[r] = \{v[r] \mid v \in Z\}$.

Lemma 2.1. *Let Z be a zone, $g \in \mathcal{B}(X)$ and $r \subseteq X$. Then $Z \wedge g$, $Z^{\uparrow}$ and $Z[r]$ are also zones (Aceto et al., 2007).*

We can now define the symbolic successor relation $\rightsquigarrow$ as follows:

- $(l, Z) \rightsquigarrow (l, Z^{\uparrow} \wedge I(l))$ – delay successor,
- $(l, Z) \rightsquigarrow (l', (Z \wedge g)[r] \wedge I(l'))$ if $l \xrightarrow{g,r} l'$ – discrete successor.

Let Z_0 be a zone containing just the one valuation which assigns zero to all clocks, and (l_0, Z_0) our initial symbolic state. All this together gives us a transition system on symbolic states, called the *zone graph.* We are usually only interested in the part that is reachable from (l_0, Z_0), but it still may be the case that even this part is infinite.

The zone graph can be made finite by the process of *normalization* (sometimes also referred to as *extrapolation*), which exploits the fact that once a clock value exceeds the maximal constant the clock is compared to in the constraints of the automaton, its precise value becomes irrelevant. There exist several such operations (Behrmann et al., 2006; Bouyer et al., 2005), which will make the state space finite while preserving its properties (i.e. reachability of a state in our case).

From a practical point of view, the representation of zones is very important. We obviously cannot represent a zone as a list of the clock valuations it contains. A logical approach is to represent zones by the constraints that define them, which is the basic underlying principle of the DBM data structure, which is nowadays most commonly used in tools for timed automata analysis. Section 2.3 on page 10 is devoted to describing this data structure.

2.1.4 Deciding reachability in zone graphs

A zone graph can be directly used to decide whether a particular state is reachable in a timed automaton, i.e. whether there is a run of the automaton that reaches the state. Although the algorithm is basically a depth-first search on the zone graph, it might be useful to show it here so that we can refer to some details later.

The input of an algorithm is a timed automaton together with a description of a state to check for reachability, i.e. a location s and a constraint $\varphi \in \mathcal{B}(X)$.

The algorithm keeps two sets of symbolic states – *Passed* for already processed states and *Waiting*, containing the initial state at the beginning, for states yet to be processed. The body of the main loop picks a state from the *Waiting* set, checks whether it satisfies φ, and terminates with positive answer if it does. Otherwise it checks if the state is already covered by the passed states, i.e. whether it is a subset of an already visited state with the same control location. If this is not the case, the state is added to the *Passed* set and its successors to the *Waiting* set. This is repeated as long as the *Waiting* set is nonempty.

Algorithm 1: Reachability

```
$Waiting := \{(l_0, Z_0)\}$
$Passed := \emptyset$
while $Waiting \neq \emptyset$ do
  choose and remove $(l, Z)$ from $Waiting$
  if $l = s \wedge Z \cap \varphi \neq \emptyset$ then
    return TRUE
  end if
  if $Z \nsubseteq Z'$ for all $(l, Z') \in Passed$ then
    $Passed := Passed \cup \{(l, Z)\}$
    $Waiting := Waiting \cup \{(l', Z') \mid (l, Z) \rightsquigarrow (l', Z') \wedge Z' \neq \emptyset\}$
  end if
  remove $(l, Z)$ from $Waiting$
end while
return FALSE
```

We can see from the algorithm that in order to develop a data structure that can be used to represent zones, we need it to support following operations:

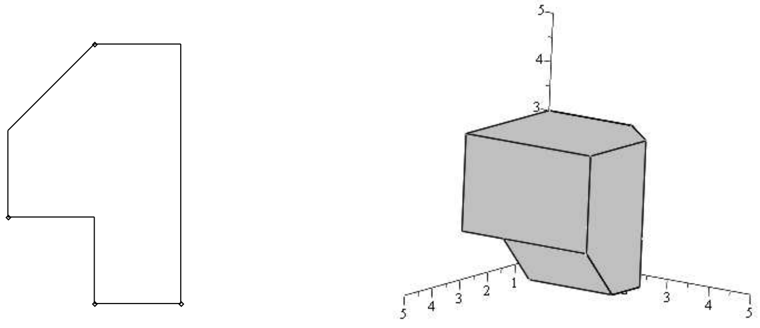
- decide whether some parts of the zone satisfies a constraint φ,
- decide whether it is a subset of another zone,
- compute successors of a state, which can be achieved by the three operations listed in Section 2.1.3 on the facing page and additional check, whether the zone is empty.

2.2 Max-plus algebra

Let $\mathbb{R}_{max}$ denote the set $\mathbb{R} \cup \{-\infty\}$, and let $a \oplus b = \max(a, b)$ and $a \otimes b = a + b$. The *max-plus algebra* is the semiring $(\mathbb{R}_{max}, \oplus, \otimes)$, that is, the set of real numbers equipped with zero element $-\infty$ with the maximum operation as addition and ordinary addition as multiplication. To further conform to the usual semiring notation, we denote $-\infty$ as $\mathbb{0}$ and 0, the neutral element with respect to max-plus multiplication, as $\mathbb{1}$. As with ordinary multiplication, we will also use the convention that $ab = a \otimes b$.

The definitions of addition and multiplication in max-plus algebra can be extended to vectors and matrices in the usual way – addition: $(\mathbf{v}_1, \dots, \mathbf{v}_n) \oplus (\mathbf{w}_1, \dots, \mathbf{w}_n) = (\mathbf{v}_1 \oplus \mathbf{w}_1, \dots, \mathbf{v}_n \oplus \mathbf{w}_n)$, multiplication by scalar: $\alpha \otimes (\mathbf{v}_1, \dots, \mathbf{v}_n) = (\alpha \otimes \mathbf{v}_1, \dots, \alpha \otimes \mathbf{v}_n)$, matrix multiplication: $(AB)_{ij} = \bigoplus_{k=1}^{n} A_{ik} \otimes B_{kj}$.

Note that the max-plus semiring is idempotent, because $a \oplus a = a$ for any element a, and that it is not a ring, because for every a except $a = \mathbb{0}$, there is no b such that $a \oplus b = \mathbb{0}$.



(a) 2D polyhedron.

(b) 3D polyhedron.

Figure 2.1: Examples of polyhedra in 2D and 3D.

The dual notion to max-plus algebra, used in some literature, is the *min-plus algebra*, also called *tropical semiring*,[2] in which the maximum operation is replaced by the minimum operation and positive infinity is used as zero element. This has also lead some authors to call max-plus algebra *the arctic semiring.*

Notation In the rest of the report we will *mainly* use the letters $a, b, c \ldots$ to denote elements of $\mathbb{R}_{max}$. Greek letters $\alpha, \beta, \ldots$ will be used for elements of $\mathbb{R}_{max}$ in context of scalar multiplication. Whenever speaking of dimension makes sense, we will denote it n. Vectors, i.e. elements of $\mathbb{R}^n_{max}$, will be denoted $\mathbf{v}, \mathbf{w}, \ldots$; we will use $\mathbf{v}_i$ to mean i-th component of vector $\mathbf{v}$, as is usual. Whenever we work with an indexed family of vectors, the indices of individual elements will be written in superscript, i.e. $\mathbf{v}^1, \ldots, \mathbf{v}^m$ to avoid confusion with selecting the element of vector.

2.2.1 Polyhedra over max-plus algebra

Convex max-plus polyhedra are the max-plus analogues of classical convex polyhedra.

Definition 2.2. A convex max-plus polyhedron is a subset of $\mathbb{R}^n_{max}$ that satisfies a finite set of (max-plus) linear inequalities.

The word *convex* refers to the property that any max-plus line segment between two points of the set is contained in the set. As shown in Figure 2.1, which displays convex max-plus polyhedra in 2D and 3D, we can see that this is different from the classical understanding of convexity.

Whenever we mention polyhedra in the report, we refer to closed convex max-plus polyhedra, unless stated otherwise. Max-plus convex sets were introduced by Zimmermann (1977), a general introduction can be found for example in Gaubert and Katz (2007).

[2] They are called tropical in honour of Imre Simon, who pioneered the field, apparently because he was from Brazil.

2.2.2 Representation of max-plus polyhedra

Because polyhedra are usually infinite sets of points, we need to represent them in some finite way, which we furthermore would like to be able to manipulate efficiently. In the following, we consider three representations of max-plus polyhedra: systems of constraints, sets of generators and max-plus cones of higher dimension; the third being a slight variation of the second. Note that the conversion between the first and the second representations is computationally rather expensive, while conversion between the second and the third is trivial.

All three representations are described in Allamigeon et al. (2008), the conversion algorithm is given in Allamigeon et al. (2009) and Allamigeon et al. (2010).

Systems of constraints

One possible representation of max-plus polyhedra is by a finite set of linear inequalities, i.e. inequalities of the form $\mathbf{ax} \oplus b \geq \mathbf{cx} \oplus d$, where $\mathbf{a}, \mathbf{c} \in \mathbb{R}_{max}^{1\times n}$ and $b, d \in \mathbb{R}_{max}$. Such a system of s constraints can be described by two matrices $A, C \in \mathbb{R}_{max}^{s\times n}$ and two vectors $\mathbf{b}, \mathbf{d} \in \mathbb{R}_{max}^{s}$, with the polyhedron $P = \{\mathbf{x} \mid A\mathbf{x} \oplus \mathbf{b} \geq C\mathbf{x} \oplus \mathbf{d}\}$. This representation is also called *external representation.*

A slightly surprising difference from ordinary linear algebra is that a system of inequalities can be represented as a system of equalities (and vice versa). This follows from the fact that $a \geq b \Leftrightarrow a = a \oplus b$. We can therefore also choose to represent max-plus polyhedra as systems of equalities.

Sets of generators

Let $A, B \subseteq \mathbb{R}_{max}^{n}$. The Minkowski sum is defined as $A \oplus B = \{\mathbf{a} \oplus \mathbf{b} \mid \mathbf{a} \in A, \mathbf{b} \in B\}$. Vector $\mathbf{a}$ is a convex combination of vectors $\mathbf{b}^1, \ldots, \mathbf{b}^m$ if $\mathbf{a} = \bigoplus_{i=1}^{m} \alpha_i \mathbf{b}^i$ for some scalars α_i such that $\bigoplus_{i=1}^{m} \alpha_i = \mathbb{1}$. Let $\mathrm{co}(A)$ denote the set of all convex combinations of points from A, i.e. its convex hull.

Vector $\mathbf{a}$ is a linear combination of vectors $\mathbf{b}^1, \ldots, \mathbf{b}^m$ if $\mathbf{a} = \bigoplus_{i=1}^{m} \alpha_i \mathbf{b}^i$ for some scalars α_i. A max-plus *cone* is a set that is closed under linear combination. The max-plus cone generated by A, denoted $\mathrm{cone}(A)$, is the set of all linear combinations of members of A. Max-plus cones are the analogues of vector (sub)spaces of vectors over a field, and they are studied in different contexts under the name *semimodules* (Gaubert and Katz, 2007).

The following theorem is an analogue of a similar theorem for classical polyhedra.

Theorem 2.3. *Any max-plus polyhedron can be represented as Minkowski sum of a bounded convex set and a cone, both of which are finitely generated. Additionally, the Minkowski sum of some bounded convex set and some cone, both of which are finitely generated, represents some max-plus polyhedron. (Gaubert and Katz, 2007)*

In other words, max-plus polyhedra can be represented as $\mathrm{co}(V) \oplus \mathrm{cone}(W)$, where V and W are finite sets of points. This can be written explicitly as

$$\alpha_1 \mathbf{v}^1 \oplus \cdots \oplus \alpha_p \mathbf{v}^p \oplus \beta_1 \mathbf{w}^1 \oplus \cdots \oplus \beta_q \mathbf{w}^q,$$

where $\bigoplus_{i=1}^{p} \alpha_i = \mathbb{1}$ and $\beta_1, \ldots, \beta_q \in \mathbb{R}_{max}$. The representation is also referred to as *internal.*

Furthermore, every polyhedron has a unique minimal V, the elements of which are called *extreme points.* Extreme points have the property that they cannot be written as a convex combination of any other points in V. There is no unique minimal W, because any generator of the minimal set can be replaced by a scalar multiple. The set of all scalar multiples of a vector is called a *ray*, and every cone has a unique minimal set of *extreme rays* that generate it. Therefore the minimal W is only unique up to the choice of representative for each ray.

Homogeneous coordinates for max-plus polyhedra

Max-plus polyhedra in n dimensions can be also represented as max-plus cones in n+1 dimensions, which can be thought of as using homogeneous coordinates. Let $P = \text{co}(V) \oplus \text{cone}(W)$ be the max-plus polyhedron generated by the sets $V, W \subseteq \mathbb{R}^n_{max}$. Now, let $Z \subseteq \mathbb{R}^{n+1}_{max}$ be defined as $Z = \{(\mathbf{v}, \mathbb{1}) \mid \mathbf{v} \in V\} \cup \{(\mathbf{w}, \mathbb{0}) \mid \mathbf{w} \in W\}$. It can be seen that $P = \{\mathbf{x} \mid (\mathbf{x}, \mathbb{1}) \in \text{cone}(Z)\}$. The requirement that the last component of the vector must be equal to $\mathbb{1}$ enforces that the scalars used to multiply elements that were in V sum to $\mathbb{1}$, while no restriction is placed on elements of W. The advantage of representing polyhedra as cones is that we don't have to distinguish between two kinds of generators, which allows the algorithms to be considerably simpler.

This representation can be easily converted back to the representation by two sets of generators. The elements with nonzero last coordinate are multiplied by a scalar such that their last coordinate becomes $\mathbb{1}$. We can then drop the last coordinate, which gives us the set V. Dropping the last coordinates of the vectors where it is $\mathbb{0}$ gives us the set W.

2.3 Difference Bound Matrices

Difference Bound Matrices (Dill, 1989), or DBMs for short, is currently one of the most efficient data structures for representing zones (Bengtsson, 2002). A DBM is, as the name suggests, a matrix with entries representing the difference between clocks. To be able to do this in a uniform way for both regular difference constraints as well as comparing just one clock to a constant, a zero clock, $\mathbf{0}$, with the constant value 0, is introduced. This approach benefits from the fact that, as mentioned in Section 2.1.3 on page 6, zones are defined by conjunctions of constraints. With a little rewriting these zone constraints can be converted into difference constraints on the form $x - y \preceq n$, where $x, y \in C \cup \{\mathbf{0}\}$, $\preceq \in \{\leq, <\}$ and $n \in \mathbb{Z}$ where C is the set of clocks. This is exactly what a DBM represents, hence one DBM encodes exactly one zone.

Since we are only interested in the tightest constraints and every constraint is concerned with two clocks, every zone is defined by at most $|C_0| \cdot (|C_0| - 1)$ constraints, where $C_0 = C \cup \{\mathbf{0}\}$. By defining the upper bound on the difference between two clocks as $x - y \preceq n$ and the lower bound as $y - x \preceq -n$, zones in systems with $|C|$ clocks can be stored as $|C_0| \times |C_0|$ matrices.

To compute the DBM for a zone, every clock in C_0 is numbered, assigning one column and one row to the clock. Every entry in the matrix, D, now represents the bound $x_i - x_j \preceq n$ where x_i, x_j is clocks, i is the row index of the matrix, j the column index. This means that rows and columns encode lower and upper bounds respectively.

To be able to handle strictness, an entry in the DBM is not just a value, but rather the tuple $(n, \preceq)$ where $n \in \mathbb{Z}$ and $\preceq \in \{\leq, <\}$, representing the bound $x_i - x_j \preceq n$. When no bound is present for the given clock difference, ∞ is used, since everything is less than or equal to infinity. Additionally, since all clocks are positive, the implicit constraints $\mathbf{0} - x_i \leq 0$ are added, and since the difference between a clock and itself should always be 0, $x_i - x_i \leq 0$ is added as well.

Finally, to be able to manipulate DBMs, comparison and addition of bounds must be defined. Bound comparison is a logical extension of comparison of integers where everything is smaller than infinity, and for equal values, strict constraints is smaller than non-strict. This provides a logical ordering of constraints $(n_1, <) < (n_2, <) < (n_2, \leq) < (n_3, \leq) < (n_4, <) < (\infty, <)$ where $n_1 < n_2 < n_3 < n_4 < \infty$. Similarly, addition is a simple extension of integer addition, where adding infinity to something is infinity, and the tightest strictness is always carried to the result. I.e. $(n_1, \preceq_1) + (n_2, \preceq_1) = (n_1 + n_2, \preceq_1 + \preceq_2)$ where $(< + <=<)$, $(\leq + \leq=\leq)$ and $(< + \leq=<)$.

Example An arbitrary zone D could be represented by the constraints, $x < 10 \wedge \mathbf{0} - x \leq -20 \wedge y - x \leq 10 \wedge \mathbf{0} - z \leq -5$, which in turn would be represented by the DBM D below.

$$D = \begin{bmatrix} (0, \leq) & (-20, \leq) & (0, \leq) & (-5, \leq) \\ (10, <) & (0, \leq) & \infty & \infty \\ \infty & (10, \leq) & (0, \leq) & \infty \\ \infty & \infty & \infty & (0, \leq) \end{bmatrix}$$

Since there can be infinitely many different conjunctions of constraints representing the same solution set and thereby the same actual zone, a canonical representation is necessary. The canonical representation for DBMs is the one representing the tightest constraints, without altering the solution set. This canonical representation can be computed by converting the DBM into a directed graph, where clocks are represented by nodes and difference constraints are labelled edges between the appropriate nodes. Now all there is to do is compute the shortest path between nodes, e.g. by using the Floyd-Warshall algorithm (Floyd, 1962), and then converting back to matrix form (Bengtsson, 2002).

2.3.1 DBM operations

As mentioned above, DBMs provide efficient algorithms for reachability-analysis operations. This includes linear time algorithms for the basic operations of delaying and resetting, as well as the operations of freeing a clock and the special reset functions for shifting and copying.

For computing the relationship between two DBMs and intersecting with one clock quadratic algorithms are provided. This is also the case for the backward delay, used in backward reachability. DBMs also provide algorithms for normalisation, the commonly used algorithm being quadratic as well.

For comparison of the DBM algorithms to our proposed algorithms on max-plus polyhedra we refer the reader to Chapter 3 on page 15.

2.3.2 Example

Since DBMs are the current industry standard for performing real-time model checking, it is useful to provide a more direct comparison between max-plus polyhedra and DBMs.

Figure 2.2 on the following page shows an example of a small TA of two clocks, that benefits from using max-plus polyhedra rather than DBMs. Imagine that this is just the tiny initial part with start location l_0 of a much bigger TA connected by the transition going out of location l_3. Running a reachability algorithm on this example will give us the following results in locations l_1 and l_2, shown as a zone Z, a max-plus polyhedron P and as a DBM D(We only consider non-strict constraints in this example. For simplicity, the elements in the DBM for the example are values, not tuple). A visual view of the two zones can be seen in Figure 2.3 on the next page. In both locations the status is that we are ready to take the transition out.

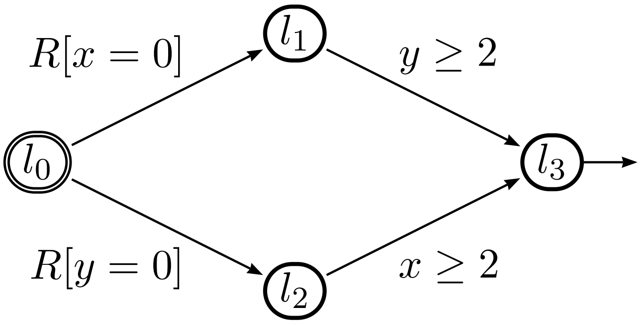
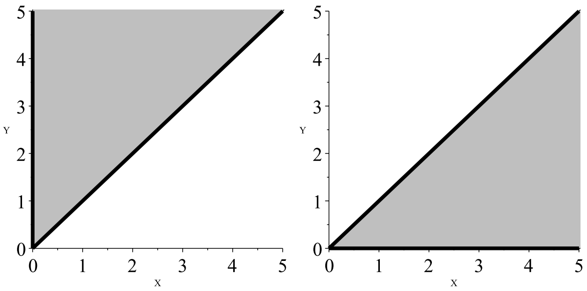


Figure 2.2: Timed automaton used for this example.



(a) l_1

(b) l_2

Figure 2.3: Location l_1 and l_2 after delay.

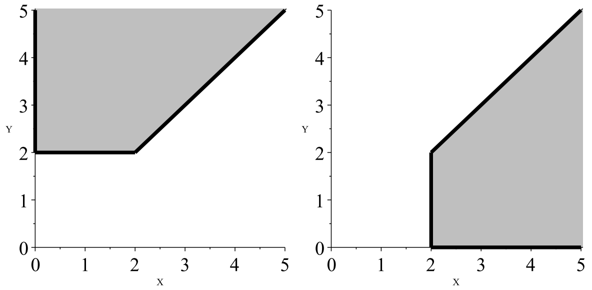
l_1**:**

$$\begin{aligned} Z &= \llbracket \mathbf{0} - x \leq 0 \wedge \mathbf{0} - y \leq 0 \wedge x - y \leq 0 \rrbracket \\ P &= co(\{\left(\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right)\}) \oplus cone(\{\left(\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right), \left(\begin{smallmatrix} -\infty \\ 0 \end{smallmatrix}\right)\}) \\ D &= \begin{bmatrix} 0 & 0 & 0 \\ \infty & 0 & 0 \\ \infty & \infty & 0 \end{bmatrix} \end{aligned}$$

l_2**:**

$$\begin{aligned} Z &= \llbracket \mathbf{0} - x \leq 0 \wedge \mathbf{0} - y \leq 0 \wedge y - x \leq 0 \rrbracket \\ P &= co(\{\left(\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right)\}) \oplus cone(\{\left(\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right), \left(\begin{smallmatrix} 0 \\ -\infty \end{smallmatrix}\right)\}) \\ D &= \begin{bmatrix} 0 & 0 & 0 \\ \infty & 0 & \infty \\ \infty & 0 & 0 \end{bmatrix} \end{aligned}$$

Continuing the reachability algorithm we calculate the state space going from l_1 and l_2 to



(a) l_3 via l_1 (b) l_3 via l_2

Figure 2.4: The two different possible states in location l_3.

l_3. At first these are stored as two separate states, containing the zones seen in Figure 2.4 on the facing page.

l_3 **via** l_1**:**

$$\begin{array}{rcl} Z & = & [\![\mathbf{0} - x \leq 0 \wedge \mathbf{0} - y \leq -2 \wedge x - y \leq 0]\!] \\ P & = & co(\{(\begin{smallmatrix}0\\2\end{smallmatrix})\} \oplus cone(\{(\begin{smallmatrix}0\\0\end{smallmatrix}),(\begin{smallmatrix}0\\-\infty\end{smallmatrix})\}) \\ D & = & \begin{bmatrix} 0 & 0 & -2 \\ \infty & 0 & 0 \\ \infty & \infty & 0 \end{bmatrix} \end{array}$$

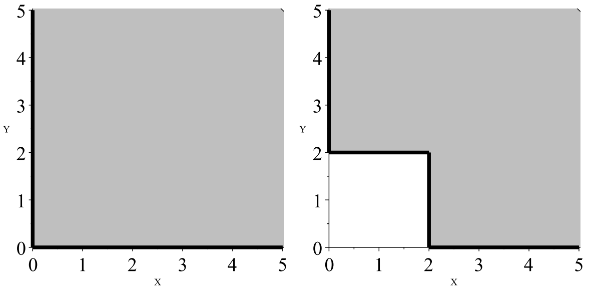
l_3 **via** l_2**:**

$$\begin{array}{rcl} Z & = & [\![\mathbf{0} - x \leq -2 \wedge \mathbf{0} - y \leq 0 \wedge y - x \leq 0]\!] \\ P & = & co(\{(\begin{smallmatrix}2\\0\end{smallmatrix})\}) \oplus cone(\{(\begin{smallmatrix}0\\0\end{smallmatrix}),(\begin{smallmatrix}-\infty\\0\end{smallmatrix})\}) \\ D & = & \begin{bmatrix} 0 & -2 & 0 \\ \infty & 0 & \infty \\ \infty & 0 & 0 \end{bmatrix} \end{array}$$

Notice that P and D are unchanged by a delay operation, hence we are ready to take a transition out of l_3. This allows us to see the difference between the two approaches. We have two different states for the location, l_3. Moving on from here, because neither max-plus polyhedra nor DBMs can do exact union, we must decide whether to do all subsequent operations on both instances of the state space, risking state space explosion, or we make an overapproximating union. Opting for the overapproximation, we get the following state for l_3.

l_3 **union:**

$$\begin{array}{rcl} Z & = & [\![\mathbf{0} - x \leq 0 \wedge \mathbf{0} - y \leq 0]\!] \\ P & = & co(\{(\begin{smallmatrix}2\\0\end{smallmatrix}),(\begin{smallmatrix}0\\2\end{smallmatrix})\}) \oplus cone(\{(\begin{smallmatrix}-\infty\\0\end{smallmatrix}),(\begin{smallmatrix}0\\-\infty\end{smallmatrix})\}) \\ D & = & \begin{bmatrix} 0 & 0 & 0 \\ \infty & 0 & \infty \\ \infty & \infty & 0 \end{bmatrix} \end{array}$$



(a) l_3 DBM union (b) l_3 MPP and exact union

Figure 2.5: Union of zones from the two l_3 states.

Figure 2.5 on the next page shows the overapproximating union both for DBM and max-plus polyhedra. As we can see the DBM overapproximation includes the entire state space, which is not exact, and therefore introduces the risk of false positives, throughout the remainder of the analysis. However, for the max-plus polyhedron, the union in this case is actually exact, thus there is no risk of false positives in this case. Finally must note that it is not necessarily the case always that exact union and max-plus union are the same.

CHAPTER 3

ALGORITHMS ON MAX-PLUS POLYHEDRA

With inspiration from Bengtsson's PhD thesis (Bengtsson, 2002) this chapter contains suggestions and ideas for different algorithms needed for forward and backward reachability checking. The algorithms accommodate the checking of properties of zones and doing different transforming operations.

Whereas Bengtsson et al. consider DBMs, we will work with and suggest equivalent algorithms for max-plus polyhedra represented as the Minkowski sum of generators. A polyhedron P always consists of a convex and a linear set of generators. The set of convex generators will be denoted V and the set of linear generators W. In the algorithms in this Chapter we use the convention of writing $\vee$ rather than $\oplus$, $+$ rather than $\otimes$, 0 instead of $\mathbb{1}$ and $-\infty$ instead of $\mathbb{0}$ for the ease of reading the pseudo code. Additionally note that n always denotes the number of clocks for a system, and p the number generators for a max-plus polyhedra.

3.1 Conversion algorithms

Some of the algorithms described in this chapter use two predefined algorithms described in Section 2.2.2 on page 10. These are concerned with the conversion from polyhedra to cones (Algorithm 2) and back (Algorithm 3 on the following page).

Algorithm 2: `poly-to-cone`(P)

```
$G := \emptyset$
for all $\mathbf{v} \in V$ do
  $G := G \cup \{(\mathbf{v}, 0)\}$
end for
for all $\mathbf{w} \in W$ do
  $G := G \cup \{(\mathbf{w}, -\infty)\}$
end for
return $G$
```

Algorithm 3: `cone-to-poly`(G)

```
$V, W := \emptyset$
for all $\mathbf{g} \in G$ do
  if $\mathbf{g}_n = -\infty$ then
    $W := W \cup \{\mathbf{g}\}$
  else
    for $i := 1$ to $n$ do
      $\mathbf{g}_i := \mathbf{g}_i + -\mathbf{g}_n$
    end for
    $V := V \cup \{\mathbf{g}\}$
  end if
end for
return $V, W$
```

3.2 Property checking

The algorithms in this section do not alter a polyhedron, but are used in reachability analysis to determine whether a given state is reached, as well as determining if some state has previously been visited.

3.2.1 Emptiness test – `consistent`(P)

To check for consistency of a polyhedron is to check whether it contains a legal clock valuation. By the definition of max-plus polyhedra, a consistent polyhedron is one that at least contains one generator in the convex set. This is because the set of scalars for the convex set must sum to max-plus one, however if there are no generators, there will be no scalars and the sum of nothing is max-plus zero (Allamigeon et al., 2008).

Additionally, clocks will always have a value in positive Euclidean space. Hence, for the polyhedron to be consistent, there must exist at least one combination of scalars such that all dimensions are positive or 0.

However, if we start with a polyhedron in positive space and only use the operations described here (for `reset`, we additionally require it to be used with positive constants), this will preserve the polyhedron in positive space. This makes the test for emptiness $O(1)$ as we only need to check if we have a non-empty set of convex generators according to the definition.

3.2.2 Membership test – `contains-point`$(\mathbf{x}, P)$

Testing whether a point $\mathbf{x}$ is contained in a polyhedron P is not directly used in reachability analysis, but is used as a subroutine in `contains` and `cleanup` algorithms.

Checking whether one point $\mathbf{x}$ can be generated by a polyhedron P is done by converting P to a cone G of $\mathbb{R}^{n+1}_{max}$ as shown in Algorithm 2 on the previous page. Similarly, $\mathbf{x}$ is converted to $\mathbf{x}'$ of $\mathbb{R}^{n+1}_{max}$ with the last coordinate set appropriate according to whether it is an actual point or a ray representative.

Then the only thing left to do is to see if $\mathcal{G}\mathbf{y} = \mathbf{x}'$ admits a solution, where $\mathcal{G}$ is a matrix containing all generators of G as columns. The equation may not have a solution, but the inequality $\mathcal{G}\mathbf{y} \leq \mathbf{x}'$ always does. We can compute the maximal solution $\widehat{\mathbf{y}}$ of this inequality according to the formula $\widehat{\mathbf{y}}_i = \min_{1 \leq j \leq n+1}(\mathbf{x}'_j - \mathcal{G}_{ji})$. The equation $\mathcal{G}\mathbf{y} = \mathbf{x}'$ then has a solution if and only if $\mathcal{G}\widehat{\mathbf{y}} = \mathbf{x}'$. If so $\mathbf{x}'$ is in P otherwise it is not. The algorithm runs in $O(pn)$ time. More details are provided by Allamigeon et al. (2008).

Algorithm 4: `contains-point`$(\mathbf{x}, G)$

```
for all $\mathbf{g}^i \in G$ do
  $\mathbf{y}_i := \min_{1 \le j \le n+1}(\mathbf{x}_j - \mathbf{g}^i_j)$
end for
for all $1 \le j \le n+1$ do
  $\mathbf{z}_j := \max_{\mathbf{g}^i \in G}(\mathbf{y}_i + \mathbf{g}^i_j)$
end for
if $\mathbf{x} = \mathbf{z}$ then
  return true
end if
return false
```

3.2.3 Subset test – `contains`(P, P')

Inclusion checking is a crucial operation when doing state space exploration. It is needed to determine whether a given state has already been visited, and hence does not need to be traversed again. Without the ability to do inclusion checking, reachability analysis of TAs containing cycles would be infeasible. To do this for max-plus polyhedra is simple: given two polyhedra P and P', to determine if P contains P', we just need to check whether all generators of P' can be generated by P.

The complexity of the contains algorithm is $O(pp'n)$, where n is the number of clocks, p is the number of generators for P and p' the number of generators for P', as every call to `contains-point` takes $O(pn)$ and there is p' of them. For comparison, the DBM algorithm for inclusion checking is quadratic in the number of clocks, $O(n^2)$.

Algorithm 5: `contains`(P, P')

```
$G :=$ poly-to-cone$(P)$
$G' :=$ poly-to-cone$(P')$
for all $\mathbf{g}' \in G'$ do
  if $\neg$contains-point$(\mathbf{g}', G)$ then
    return false
  end if
end for
return true
```

3.2.4 Constraint satisfaction – `satisfied`(P, φ)

It is important to be able to check, non-destructively, whether a polyhedron partially satisfies a given constraint. This is used to determine whether a state satisfies some timing constraint given in the initial query.

For constraints on the form $\varphi = x_i - x_j \sim c$ where $\sim \in \{\le, =, \ge\}$ and $x_i \in C$, $x_j \in C \cup \{0\}$, a simple `satisfied` algorithm is to intersect the polyhedron with the constraint and then check for emptiness. The correctness of this is trivial and will not need more discussion. Complexity-wise, however, the algorithm is $O(p^2n)$ due to the use of constraint intersection, described in

Section 3.3.1. We believe this is not the fastest approach, but improving this is left to future work.

The DBM algorithm for constraint satisfaction uses the same approach, namely adding the constraint to the zone and checking for consistency, which is $O(n^2)$.

3.3 Transformations

The transformations are the algorithms which modify a polyhedron in order to determine the reachable state space for a TA. This includes the basic algorithms of *delay* and *reset* among others.

3.3.1 Constraint intersection – $\texttt{and}(P, x_i - x_j \sim c)$

Adding a constraint is done by intersecting the polyhedron P with the difference constraint $x_i - x_j \sim c$. An algorithm (Algorithm 7) for computing the intersection of a max-plus cone with the half-space satisfying a set of max-plus inequalities is given by Allamigeon et al. (2009). As any difference constraint can be expressed as max-plus inequality, the only thing that remains is to convert the constraint of the form $x_i - x_j \sim c$ to two vectors **a** and **b** that represent the max-plus half-space $\{\mathbf{x} \mid \mathbf{a} \otimes \mathbf{x} \leq \mathbf{b} \otimes \mathbf{x}\}$, and use the aforementioned algorithm on the cone representation of the polyhedron.

Without loss of generality, we can assume that the constraint is $x_i - x_j \leq c$, because $x_i - x_j \geq c$ is equivalent to $x_j - x_i \leq -c$ and intersection with $x_i - x_j = c$ can be computed by simply computing the intersection with $x_i - x_j \leq c$ and $x_i - x_j \geq c$. This constraint can be rewritten as $x_i \leq x_j + c$, or in the max-plus notation, $0 \otimes x_i \leq c \otimes x_j$. This means that **a** will be a vector the components of which will be -∞, except for the i'th value, which will be 0. Similarly, **b** will consist of negative infinities except at the j'th place, which will be c.

Algorithm 6: $\texttt{and}(P, x_i - x_j \leq c)$

```
$\mathbf{a} := (-\infty, \ldots, -\infty)$
$\mathbf{b} := (-\infty, \ldots, -\infty)$
$\mathbf{a}_i := 0$
$\mathbf{b}_j := c$
return cone-to-poly(intersect-halfspace(poly-to-cone($P$), $\mathbf{a}, \mathbf{b}$))
```

Algorithm 7: $\texttt{intersect-halfspace}(G, \mathbf{a}, \mathbf{b})$

```
$G^{\leq} := \{\mathbf{g} \in G \mid \mathbf{a} \otimes \mathbf{x} \leq \mathbf{b} \otimes \mathbf{x}\}$
$G^{>} := \{\mathbf{g} \in G \mid \mathbf{a} \otimes \mathbf{x} > \mathbf{b} \otimes \mathbf{x}\}$
$H := G^{\leq}$
for all $(g, h) \in G^{\leq} \times G^{>}$ do
  $H := H \cup \{((\mathbf{a} \otimes \mathbf{h}) \otimes \mathbf{g}) \oplus ((\mathbf{b} \otimes \mathbf{g}) \otimes \mathbf{h})\}$
end for
return $H$
```

The complexity of `intersect-halfspace` is $O(p^2 n)$ – the evaluation of the expression inside the loop takes $O(n)$ and may be performed $O(p^2)$ times, which is also the upper bound on number of computed generators. As the conversions to and from the cone representation can be naively implemented in $O(pn)$, the overall time complexity of intersection with a constraint is in $O(p^2 n)$. The complexity of this operation for DBMs is $O(n^2)$.

3.3.2 Delay – up(P)

The delay operation is one of the basic algorithms used for forward exploration. Delay is easily done on a max-plus polyhedron, simply by copying all points in the convex combination to the linear combination, as shown by Dyhrberg et al. (2010). The complexity is $O(pn)$, as we need to loop though all points and for each point we need to copy its value in every dimension. For DBMs the delay algorithm is linear in the number of clocks, $O(n)$.

```
Algorithm 8: up(P)
    $W := W \cup V$
```

Proof of correctness Recall that the definition of delayed set P is $P^{\uparrow} = \{\mathbf{v} + d \mid \mathbf{v} \in P, d \in \mathbb{R}_{\geq 0}\}$, i.e. that any point in $P^{\uparrow}$ is obtained by translating some point from P along all axes by some non-negative distance d. Let $P = \text{co}(V) \oplus \text{cone}(W)$ be a max-plus polyhedron. Our algorithm copies all points from V to W – let $\widehat{P} = \text{co}(V) \oplus \text{cone}(V \cup W)$ be the result of running our algorithm on P. We have to prove that $\widehat{P} = P^{\uparrow}$. This is the case if $\widehat{P} \subseteq P^{\uparrow}$ and $P^{\uparrow} \subseteq \widehat{P}$.

- $P^{\uparrow} \subseteq \widehat{P}$: Let $\mathbf{p} \in P^{\uparrow}$. From the definition of delay operation, we have that $\mathbf{p} = \mathbf{q} + d$ for $\mathbf{q} \in P$ and some $d \in \mathbb{R}_{\geq 0}$. Hence,

$$\mathbf{p} = \mathbf{q} + d = \left(\bigoplus_{i=1}^{p} \alpha_i \mathbf{v}^i \oplus \bigoplus_{i=1}^{q} \beta_i \mathbf{w}^i \right) \otimes d = \bigoplus_{i=1}^{p} \alpha_i d \mathbf{v}^i \oplus \bigoplus_{i=1}^{q} \beta_i d \mathbf{w}^i,$$

  where $\alpha_i, \beta_i \in \mathbb{R}_{max}$, $\mathbf{v}^i \in V$, $\mathbf{w}^i \in W$ and $\bigoplus_{i=1}^{p} \alpha_i = 0$. The rightmost side of the equality now expresses $\mathbf{p}$ as a linear combination of generators from $V \cup W$. To satisfy the definition of max-plus polyhedra, we also need to have a convex combination of at least one point in the expression. Because d is non-negative, we know that $\alpha_i d \mathbf{v}^i \oplus \alpha_i \mathbf{v}^i = \alpha_i d \mathbf{v}^i$ for all $1 \leq i \leq p$. This means that we can add $\bigoplus_{i=1}^{p} \alpha_i \mathbf{v}^i$ to the sum without changing its value:

$$\mathbf{q} + d = \bigoplus_{i=1}^{p} \alpha_i \mathbf{v}^i \oplus \bigoplus_{i=1}^{p} \alpha_i d \mathbf{v}^i \oplus \bigoplus_{i=1}^{q} \beta_i d \mathbf{w}^i,$$

  which means that $\mathbf{q} + d \in \text{co}(V) \oplus \text{cone}(V \cup W)$ and therefore $\mathbf{p} \in \widehat{P}$.

- $\widehat{P} \subseteq P^{\uparrow}$: Let $\mathbf{p}$ be a point in $\widehat{P}$. From the definition of $\widehat{P}$, we can assume that

$$\mathbf{p} = \bigoplus_{i=1}^{p} \alpha_i \mathbf{v}^i \oplus \bigoplus_{i=1}^{q} \beta_i \mathbf{w}^i \oplus \bigoplus_{i=1}^{r} \gamma_i \mathbf{v}^i,$$

  for $\alpha_i, \beta_i, \gamma_i \in \mathbb{R}_{max}$, $\mathbf{v}^i \in V$, $\mathbf{w}^i \in W$ and $\bigoplus_{i=1}^{p} \alpha_i = 0$. We have to show that this point can be expressed as some point $\mathbf{q} \in P$ translated by positive distance d diagonally along all axes. In order to find $\mathbf{q}$, we need all the points from V in the sum to form convex combination. We can accomplish this by letting $d = \max_{i=1}^{r} \gamma_i$, which allows us to rewrite the equation above as:

$$\mathbf{p} = \bigoplus_{i=1}^{p} (\alpha_i - d) d \mathbf{v}^i \oplus \bigoplus_{i=1}^{q} (\beta_i - d) d \mathbf{w}^i \oplus \bigoplus_{i=1}^{r} (\gamma_i - d) d \mathbf{v}^i,$$

and then factoring it out,

$$\mathbf{p} = \left(\bigoplus_{i=1}^{p} (\alpha_i - d)\mathbf{v}^i \oplus \bigoplus_{i=1}^{q} (\beta_i - d)\mathbf{w}^i \oplus \bigoplus_{i=1}^{r} (\gamma_i - d)\mathbf{v}^i \right) \otimes d = \mathbf{q} + d.$$

As we can see that $\bigoplus_{i=1}^{r}(\gamma_i - d) = 0$ and subsequently $\bigoplus_{i=1}^{p}(\alpha_i - d) \oplus \bigoplus_{i=1}^{r}(\gamma_i - d) = 0$, it is the case that $\mathbf{q} \in \mathrm{co}(V) \oplus \mathrm{cone}(W) = P$ and therefore $\mathbf{q} + d = \mathbf{p} \in P^{\uparrow}$. □

3.3.3 Backward delay – `down`(P)

Backward delay is the algorithm for determining all the states that could have brought us into a given state by delay. It is used when doing backward state-space exploration rather than forward exploration. To do backward delay on a polyhedron we need to add the generator $(-1, \ldots, -1)$ to the convex set and then intersect with the polyhedron for positive Euclidean space. Since `and` is complexity $O(p^2 n)$ the entire complexity of down is $O(p^2 n^2)$. Backward delay for DBMs is $O(n^2)$.

Algorithm 9: `down`(P)

```
$V := V \cup \{(-1, \ldots, -1)\}$
for $i := 1$ to $n$ do
  and$(P, x_i \geq 0)$
end for
```

Proof of correctness Let $P = (\mathrm{co}(V) \oplus \mathrm{cone}(W)) \subseteq \mathbb{R}^n_{\geq 0}$ be a polyhedron. The polyhedron that we would like to get after applying this algorithm is $P^{\downarrow} = \{\mathbf{v} \mid \mathbf{v} \in \mathbb{R}^n_{\geq 0}, \mathbf{v} + d \in P, d \in \mathbb{R}_{\geq 0}\}$. Let $\widehat{P} = (\mathrm{co}(V \cup \{\mathbf{f}\}) \oplus \mathrm{cone}(W)) \cap \mathbb{R}^n_{\geq 0}$, where $\mathbf{f}$ is the vector which has -1 as all components. We need to prove that $P^{\downarrow} = \widehat{P}$.

- $P^{\downarrow} \subseteq \widehat{P}$: Let $\mathbf{p}$ be a point in $P^{\downarrow}$. This means that there is a point $\mathbf{p} + d \in P$ for some $d \in \mathbb{R}_{\geq 0}$, from which we can derive possible representation of $\mathbf{p}$, if we consider the scalars of $\mathbf{p} + d$ as max-plus multiples of d:

$$\mathbf{p} + d = \bigoplus_{i=1}^{p} \alpha_i d \mathbf{v}^i \oplus \bigoplus_{i=1}^{q} \beta_i d \mathbf{w}^i = \left(\bigoplus_{i=1}^{p} \alpha_i \mathbf{v}^i \oplus \bigoplus_{i=1}^{q} \beta_i \mathbf{w}^i \right) \otimes d,$$

where $\alpha_i, \beta_i \in \mathbb{R}_{max}$, $\mathbf{v}^i \in V$, $\mathbf{w}^i \in W$ and $\bigoplus_{i=1}^{p} \alpha_i d = 0$. We can now show that $\mathbf{p} \in \widehat{P}$ – indeed:

$$\mathbf{p} = \bigoplus_{i=1}^{p} \alpha_i \mathbf{v}^i \oplus \bigoplus_{i=1}^{q} \beta_i \mathbf{w}^i = \bigoplus_{i=1}^{p} \alpha_i \mathbf{v}^i \oplus 0\mathbf{f} \oplus \bigoplus_{i=1}^{q} \beta_i \mathbf{w}^i.$$

In the last expression, we have $(\bigoplus_{i=1}^{p} \alpha_i) \oplus 0 = 0$ and as we know that $\mathbf{p} \in \mathbb{R}^n_{\geq 0}$, the addition of $0 \otimes \mathbf{f}$ will not change the resulting point. Therefore $\mathbf{p} \in \widehat{P}$.

- $\widehat{P} \subseteq P^{\downarrow}$: Let $\mathbf{p} \in \widehat{P}$:

$$\mathbf{p} = \bigoplus_{i=1}^{p} \alpha_i \mathbf{v}^i \oplus \gamma \mathbf{f} \oplus \bigoplus_{i=1}^{q} \beta_i \mathbf{w}^i,$$

where $\alpha_i, \beta_i, \gamma \in \mathbb{R}_{max}$, $\mathbf{v}^i \in V$, $\mathbf{w}^i \in W$ and $(\bigoplus_{i=1}^{p} \alpha_i) \oplus \gamma = 0$. Let $d = -\max_{i=1}^{p} \alpha_i$. Now we can show that $\mathbf{p} + d \in P$:

$$\mathbf{p} + d = \left(\bigoplus_{i=1}^{p} \alpha_i \mathbf{v}^i \oplus \bigoplus_{i=1}^{q} \beta_i \mathbf{w}^i \right) \otimes d = \bigoplus_{i=1}^{p} \alpha_i d \mathbf{v}^i \oplus \bigoplus_{i=1}^{q} \beta_i d \mathbf{w}^i,$$

since $\bigoplus_{i=1}^{p} \alpha_i d = 0$. According to the definition, this also means that $\mathbf{p} \in P^{\downarrow}$. □

3.3.4 Resetting clocks – `reset`$(P, x_i = c)$

Reset is the operation of resetting one clock to a given value. In effect it is an affine projection to the hyperplane equivalent to a given constant for the given dimension. It is done by iterating through all generators, setting the given dimension to the given reset value for the convex generators, and setting the given dimension to $-\infty$ for the linear generators (Dyhrberg et al., 2010). This operation can be done in linear time in the number of generators $O(p)$. The reset operation on DBMs is done in linear time in the number of clocks $O(n)$.

Algorithm 10: `reset`$(P, x_i = c)$

```
for all $\mathbf{v} \in V$ do
  $\mathbf{v}_i := c$
end for
for all $\mathbf{w} \in W$ do
  $\mathbf{w}_i := -\infty$
end for
```

Proof of correctness Recall the definition of reset is $P_{R[x_i=c]} = \{\mathbf{v} \mid \exists \mathbf{v}' \in P.\ \forall 1 \leq j \leq n, j \neq i.\ \mathbf{v}_j = \mathbf{v}'_j \wedge \mathbf{v}_i = c\}$. Expressed in another way, this means that the value of i'th component of each point of polyhedron must equal to c. Hereby follows directly `reset`$(P, x_i = c) \rightarrow \mathrm{co}(V_i) \oplus \mathrm{cone}(W_i) = c$ where V_i and W_i are the sets $\{\mathbf{v}_i \mid \mathbf{v} \in V\}$ and $\{\mathbf{w}_i \mid \mathbf{w} \in W\}$ respectively.

This can be expressed as the fact that either $(\texttt{co}(V_i) = c \wedge \texttt{cone}(W_i) \leq c)$ or $(\texttt{co}(V_i) \leq c \wedge \texttt{cone}(W_i) = c)$ must always hold. Looking at the proposed algorithm the first of the two is most likely to always hold.

For the linear set we know:

(1) $\forall \mathbf{w} \in W,\ \mathbf{w}_i = -\infty$

(2) $\forall k \in \mathbb{R}_{max},\ -\infty \otimes k = -\infty$

(3) $-\infty \oplus -\infty = -\infty$

From (1), which is given by the algorithm, and (2), which is part of the definition of the max-plus semiring, we can deduce that $\forall \mathbf{w} \in W,\ \mathbf{w}_i \otimes \beta = -\infty$. Combining this with (3), also from the definition of max-plus, leads us to $\texttt{cone}(W_i) = -\infty \Rightarrow \texttt{cone}(W_i) \leq c$ which is exactly what we are looking for.

For the convex part we know:

(1) $\forall \mathbf{v} \in V,\ \mathbf{v}_i = c$

(2) $\bigoplus_1^n \alpha = 0$

(3) $0 \otimes c = c$

(4) $\forall(k \leq 0) \in \mathbb{R}_{max},\ \forall c \geq 0,\ k \otimes c \leq c$

(5) $\forall(k \leq c) \in \mathbb{R}_{max},\ k \oplus c = c$

Statement (1) is derived directly from the algorithm, whereas statements (2-5) is given by the definition of the max-plus semiring. Combining (1) and (2) with (3) gives us that $\exists \mathbf{v} \in V,\ \mathbf{v}_i \otimes \alpha = c$. Furthermore we can conclude from (1), (2) and (4) that $\nexists \mathbf{v} \in V,\ \mathbf{v}_i \otimes \alpha > c$. Finally these two deductions combined with (5) leads to $\texttt{co}(V_i) = c$ as there exists one generator resulting in c and the rest of the generators are less than or equal to c.

This wraps up the part ensuring the i'th dimension is always returning the correct value after a reset.

For dimensions other than the i'th, nothing is changed. This can be verified by converting P and $P_{R[x_i=c]}$ polyhedra of $\mathbb{R}^{n-1}$ removing the i'th dimension. It is easily seen that these two polyhedra are equivalent as they should be from the definition of reset.

3.3.5 Shifting a polyhedron – shift$(P, x_i = x_i + c)$

Shifting is another special reset operation that translates the entire polyhedron in one dimension. For all generators in both the convex and the linear combination, this can be done by adding c to the value of the chosen dimension. For the linear combination, the convention $-\infty + c = -\infty$ is used. Finally, the polyhedron is intersected with positive space for the shifted clock. Complexity of shift is $O(p^2n)$ for max-plus polyhedra and $O(n)$ for DBMs. Note that the intersection with positive space is only necessary when $c < 0$, which makes the complexity $O(p)$ in the case that $c \geq 0$.

Algorithm 11: shift$(P, x_i = x_i + c)$

```
for all $\mathbf{v} \in V$ do
  $\mathbf{v}_i := \mathbf{v}_i + c$
end for
for all $\mathbf{w} \in W$ do
  $\mathbf{w}_i := \mathbf{w}_i + c$
end for
and$(P, x_i \geq 0)$
```

Proof of correctness In order to prove this algorithm correct we first need to introduce the definition of a max-plus linear map.

Definition 3.1. A max-plus linear map is a function $f : \mathbb{R}^n_{max} \to \mathbb{R}^n_{max}$ such that

1. $\alpha \otimes f(\mathbf{v}) = f(\alpha \otimes \mathbf{v})$, and

2. $f(\mathbf{v}) \oplus f(\mathbf{w}) = f(\mathbf{v} \oplus \mathbf{w})$,

for any $\alpha \in \mathbb{R}_{max}$ and $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n_{max}$. Let $f(V)$ denote the set $\{f(\mathbf{v}) \mid \mathbf{v} \in V\}$ for $V \subseteq \mathbb{R}^n_{max}$.

Lemma 3.2. *Let $f : \mathbb{R}^n_{max} \to \mathbb{R}^n_{max}$ be a linear map, $P = \mathrm{co}(V) \oplus \mathrm{cone}(W)$ a max-plus polyhedron, and $C = \mathrm{cone}(U)$ a max-plus cone, where U, V and W are finite. Then*

1. $f(P) = \mathrm{co}(f(V)) \oplus \mathrm{cone}(f(W))$, *and*
2. $f(C) = \mathrm{cone}(f(U))$.

What this lemma says is basically that in order to apply a max-plus linear function to the polyhedron, we can just apply it to the generators and get the right result.

Proof. We will prove only the first equation, as proving the other works on a similar basis. Let $\mathbf{x} \in \mathbb{R}^n_{max}$ and $P = \mathrm{co}(V) \oplus \mathrm{cone}(W)$ be a polyhedron and f a linear map. Then

$$\begin{aligned}
\mathbf{x} \in f(P) &\Leftrightarrow \mathbf{x} \in \{f(\mathbf{y}) \mid \mathbf{y} \in P\} \\
&\Leftrightarrow \mathbf{x} \in \{f(\mathbf{y}) \mid \mathbf{y} = \textstyle\bigoplus_{i=1}^{p} \alpha_i \mathbf{v}^i \oplus \bigoplus_{i=1}^{q} \beta_i \mathbf{w}^i\} && \text{(definition of polyhedron)} \\
&\Leftrightarrow \mathbf{x} = f(\mathbf{y}) = f\left(\textstyle\bigoplus_{i=1}^{p} \alpha_i \mathbf{v}^i \oplus \bigoplus_{i=1}^{q} \beta_i \mathbf{w}^i\right) \\
&\Leftrightarrow \mathbf{x} = f\left(\textstyle\bigoplus_{i=1}^{p} \alpha_i \mathbf{v}^i\right) \oplus f\left(\textstyle\bigoplus_{i=1}^{q} \beta_i \mathbf{w}^i\right) && \text{(linear map 1.)} \\
&\Leftrightarrow \mathbf{x} = \textstyle\bigoplus_{i=1}^{p} f\left(\alpha_i \mathbf{v}^i\right) \oplus \bigoplus_{i=1}^{q} f\left(\beta_i \mathbf{w}^i\right) && \text{(linear map 1.)} \\
&\Leftrightarrow \mathbf{x} = \textstyle\bigoplus_{i=1}^{p} \alpha_i f\left(\mathbf{v}^i\right) \oplus \bigoplus_{i=1}^{q} \beta_i f\left(\mathbf{w}^i\right) && \text{(linear map 2.)} \\
&\Leftrightarrow \mathbf{x} \in \mathrm{co}(f(V)) \oplus \mathrm{cone}(f(W)),
\end{aligned}$$

where $p, q \in \mathbb{N}$, $\alpha_i, \beta_i \in \mathbb{R}_{max}$, $\mathbf{v}^i \in V$, $\mathbf{w}^i \in W$ and $\bigoplus_{i=1}^{p} \alpha_i = 0$. □

The algorithm $\mathtt{shift}_{x_i = x_i + c}(x_1, \ldots, x_i, \ldots, x_n) = (x_1, \ldots, x_i + c, \ldots, x_n)$ consists of two parts. The first part is a max-plus linear map for adding the value c to each clock x_i in each generator which belongs to either V or W. The other part is the algorithm $\mathtt{and}(P, x_i \geq 0)$. Our algorithm applies the linear function to all generators, and by Lemma 3.2, the first part of `shift` algorithm can be proved. Given that the correctness of the `and` algorithm has already been proven, the algorithm is therefore correct.

3.3.6 Copying clock values – $\mathtt{copy}(P, x_i = x_j)$

A special reset operation where the value of one clock is copied into another clock. This is done by, for each generator copying the value x_j to x_i in both the convex and linear combination, an operation which as with regular `reset` can be done in $O(p)$. Also as for regular `reset`, `copy` is $O(n)$ for DBMs.

Algorithm 12: $\mathtt{copy}(P, x_i = x_j)$

```
for all $\mathbf{v} \in V$ do
    $\mathbf{v}_i := \mathbf{v}_j$
end for
for all $\mathbf{w} \in W$ do
    $\mathbf{w}_i := \mathbf{w}_j$
end for
```

Proof of correctness According to the definition of copy algorithm, we know the function $\mathrm{copy}_{x_i = x_j}(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n) = (x_1, \ldots, x_j, \ldots, x_j, \ldots, x_n)$ is linear. The correctness of the copy algorithm can be proved by lemma 3.2. The copy algorithm sets the valuations of clock x_i equal to the valuations of clock x_j in each generator.

3.3.7 Removing constraints – free(P, x_i)

Freeing a clock on a polyhedron is done by removing all constraints on that particular clock. It is used in combination with constraint intersection to handle resets when exploring the state-space backward. Performing the free operation on a polyhedron can be done by resetting the polyhedron with respect to the clock being freed, and then adding the generator containing $-\infty$ for all clocks except x_i, where the value should be 0, to the set W. Complexity-wise, free is $O(n+p)$ since reset is linear in the number of points, and creating a vector is linear in the number of clocks. Free on DBMs is $O(n)$.

Algorithm 13: free(P, x_i)

```
    reset$(P, x_i)$
    $\mathbf{g} := (-\infty, \ldots, -\infty)$
    $\mathbf{g}_i := 0$
    $W := W \cup \{\mathbf{g}\}$
```

Proof of correctness Let P be a polyhedron. Removing a constraint on clock $\mathbf{x}_d$ in P should result in the set $P_{*d} = \{\mathbf{v} \mid \exists \mathbf{v}' \in P. \, \forall 1 \leq i \leq n, d \neq i. \; \mathbf{v}_i = \mathbf{v}'_i\}$. Let $\widehat{P}$ be the polyhedron obtained by running our algorithm on P, i.e. by resetting the clock $\mathbf{x}_d$ to zero and adding a generator with zero at the d'th place and negative infinities at all others to the set of linear generators. Let $\mathbb{0}_d$ denote this generator. We shall now prove that $P_{*d} = \widehat{P}$.

- $P_{*d} \subseteq \widehat{P}$: Let $\mathbf{p}$ be a point in P_{*d}. By the definition of P_{*d}, there must be a point $\mathbf{q} \in P_{R[\mathbf{x}_d=0]}$ such that it has the same values of all components except the d'th. From the definition of reset operation, we also know that $\mathbf{q}_d = 0$. We can see that $\mathbf{q} \oplus (\mathbf{p}_d \otimes \mathbb{0}_d)$ is equal to $\mathbf{p}$ and is still contained in $\widehat{P}$, because of the added generator. In other words, $\mathbf{q} \oplus (\mathbf{p}_d \otimes \mathbb{0}_d) = \mathbf{p} \in \widehat{P}$.

- $\widehat{P} \subseteq P_{*d}$: Let $\mathbf{p} \in \widehat{P}$. Because the reset and addition of $\mathbb{0}_d$ only affected the d'th coordinate, we know that there is a point $\mathbf{p}' \in P$ that is equal to $\mathbf{p}$ except for the d'th coordinate. Because the d'th coordinate was first set to zero and then possibly incremented by some scalar multiple of $\mathbb{0}_d$, it cannot be negative and therefore satisfies the second condition of membership in P_{*d}. Thus, $\mathbf{p} \in P_{*d}$. □

3.3.8 Union overapproximation – convex-union(P_1, P_2)

This operation returns the smallest polyhedron that contains both P_1 and P_2. Such an operation is useful when performing reachability analysis with *convex hull overapproximation* of the state space. We have shown in our previous work (Dyhrberg et al., 2010) that the overapproximated symbolic state obtained with max-plus polyhedra is at most as large as the one obtained with similar operation on equivalent DBMs.

The algorithm for max-plus polyhedra simply creates union of generators of both polyhedra, which can be considered an $O(pn)$ operation. For DBMs, the overapproximation algorithm is $O(n^2)$ as it simply involves searching through the matrices and picking the higher value from them.

3.4 Cleaning up

All transformations on max-plus polyhedra, except for `shift`, may introduce redundant generators, i.e. generators that are not extreme points and can therefore be expressed as a combination of the extreme points. Because the time complexity of most of the algorithms depends on the number of generators, it is desirable to remove such redundant generators to store only the minimal number of generators.

3.4.1 Removing redundant generators – `cleanup`(P)

Cleaning up results in combination of a unique and minimal representation for max-plus polyhedra. The uniqueness part is to be understood as a minimal unique convex part, and a minimal but not necessarily unique linear part. Making the linear part unique would require a slight alteration of *cleanup* which normalizes the linear vectors in some way. However, we do not believe this will give us any significant advantage, so we have decided not to do this.

Algorithm 14: `cleanup`(P)

```
$G :=$ poly-to-cone($P$)
for all $\mathbf{g} \in G$ do
  if contains-point($\mathbf{g}, C \setminus \{\mathbf{g}\}$) then
    $G := G \setminus \{\mathbf{g}\}$
  end if
end for
return cone-to-poly($C$)
```

This is done by checking for each point whether it can be generated by the other points. If it does, it is removed. Every such check costs $O(pn)$, which makes the time complexity of this algorithm $O(p^2n)$ (Gaubert and Katz, 2007; Allamigeon et al., 2008; Butkovič et al., 2007).

A cleaned up polyhedron can be considered to be canonical in the sense that a cleaned up polyhedron is the optimal input for the different algorithms in a similar way as a canonical DBM is. Therefore in some way it can be compared to the canonicalization algorithm for DBMs, which is the standard Floyd-Warshall algorithm and therefore $O(n^3)$.

Additionally, a cleaned up polyhedron is the minimal form, whereas a canonical DBM is not.

3.5 Summary

In Table 3.1 on the next page we see a comparison of the complexity of the algorithms for max-plus polyhedra as well as DBMs. The number of clocks is denoted n, while p denotes the number of generators of the polyhedron. Complexity-wise, the two data structures seem to be pretty equal, although the DBM approach appears to have a slight advantage. However, max-plus polyhedra allow more complex shapes to be represented by a single instance, i.e. some non-convex unions of zones.

Table 3.1 notes

a. This does require the zone to always be canonical and that every other operation checks whether the upper bound is lower than the corresponding lower bound whenever a bound is changed, setting D_{00} to a negative value if this is the case.

	Max-plus polyhedra	DBM
Emptiness test	$O(1)$	$O(1)^a$
Subset test	$O(p^2n)$	$O(n^2)$
Constraint satisfaction	$O(p^2n)^b$	$O(n^2)$
Constraint intersection	$O(p^2n)$	$O(n^2)$
Delay	$O(pn)$	$O(n)$
Backward delay	$O(p^2n^2)$	$O(n^2)$
Resetting clocks	$O(p)$	$O(n)$
Shifting	$O(p^2n)$	$O(n)$
Copying clock values	$O(p)$	$O(n)$
Removing constraints	$O(p+n)$	$O(n)$
Union overapproximation	$O(pn)$	$O(n^2)$
Removing redundant generators	$O(p^2n)$	$O(n^3)^c$

Table 3.1: Complexity of algorithms for max-plus polyhedra and DBMs.

b. We believe that this can be done in $O(p)$. However, the verification of this is left as future work, see Section 5.1 on page 29.

c. As mentioned in 3.4.1 on the previous page, we have decided to compare this to DBM canonicalization. Note that some of the DBM algorithms accommodate the possibility of a slight altering which preserves canonicity, allowing the canonicalization to be skipped.

CHAPTER 4

IMPLEMENTATION

In order to try out max-plus polyhedra as a data structure for reachability analysis in practice, we have decided to use the open source Python project opaal (Olesen and Jørgensen, 2010) as the basic model-checking tool. The goal of opaal is to provide a simple base for rapid prototyping for model-checking concepts. Through its use of MPI, opaal is implemented with support for distributed and parallel model-checking alike.

The design architecture of opaal consists of four parts, a user interface, a passed-waiting list, a model parser and the model checking algorithms. With the design criteria of loose coupling, it should be easy to swap any component for a different implementation in order to try out.

Though opaal is a work in progress, enough components are implemented to make it useful for prototyping. The current version includes a passed-waiting list, a reachability algorithm and functionality to parse Uppaal models exported as XML-files. Additionally, opaal allows us to call Python code from transitions and invariants.

In order to let opaal use our data structure, we have first implemented it in Python with all of the functions and operations we require. This type can then be exported for use as a variable in models. Afterwards, we rewrite the model to call the appropriate methods on our data structure, instead of using regular guards and invariants. Additionally, since we do not have an algorithm for performing normalization of max-plus polyhedra, we require the timed automaton given as an input to be bounded, i.e. to have upper invariant in all locations, which guarantees that the state space of the automaton will be finite.

Currently, opaal cannot create actual clock variables like in Uppaal. Instead, we modify the model to create an n-dimensional max-plus polyhedron. Each of the original clocks then corresponds to a numerical index, and this index is then used whenever referring to that clock in guards or invariants. This is currently being worked on within opaal, so in the future we will be able to refer to clocks by names defined similarly to regular clocks in a Uppaal model.

For any model that has been modified to use max-plus polyhedra, we are able to perform reachability queries. It is also possible to specify clock constraints in the query, but for simplicity, the prototype only supports checking a single clock constraint per query.

By itself, the prototype cannot be used to determine whether or not max-plus polyhedra will lead to any actual performance gain, since opaal is intended as a platform for prototyping, rather than a competitor to highly optimized tools like Uppaal. However, we are still able to get a basic idea about the performance of our data structure, and although we have only been able to test against a small number of models, our observations so far suggest that the idea is still viable.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

The objective of this project was to expand upon our previous work (Dyhrberg et al., 2010) and test our approach in practice. For this purpose, we have designed algorithms for forward reachability analysis of timed automata, and created a proof-of-concept implementation, integrated into the opaal model checker. Preliminary testing confirms that the idea is not completely infeasible, although the exact performance potential remains unknown.

Several issues are still unresolved, which prevents us from using max-plus polyhedra to their full effect. Future work would benefit from considering these issues, described below.

5.1 Performance

So far, we have shown that real-time model checking using max-plus polyhedra is possible. It remains to be shown that it can compete with the traditional zone-based approach in terms of performance. Available real-time model checkers (Uppsala University and Aalborg University, 2010; Yovine, 1997) are highly optimized as a result of the immense amount of work that has been put into them. This is not the case with our implementation which was intended as a proof of concept, which makes it direct comparison with other model checkers difficult.

Some of the algorithms we have suggested for operations on max-plus polyhedra use a rather naive approach. This leads us to believe that some optimization of the algorithms can be achieved. For example, we have a suggestion for a complexity wise better algorithm for `satisfied`: instead of naively using constraint intersection, it seems possible that we can instead compare the constraint against the generators, specifically the elements of these that relate to the constraint.

The idea is to divide the check into four different cases according to the type of constraint.

The first and second are the cases where one clock is compared to a constant. For $\leq$ constraints, there must exist at least one generator in the convex set that satisfies the condition for the clock. For $\geq$, either there must exist a generator in the convex set that satisfies the constraint or a linear generator which does not contain $-\infty$ for the given dimension.

The third and fourth cases are concerned with difference constraints, and are essentially identical as $a - b \leq c \Leftrightarrow b - a \geq -c$. Again, there must either be a convex generator satisfying the constraint, or a linear generator satisfying the constraint, where the value for neither of the related dimensions must be $-\infty$.

As one can see this is a suggestion for a linear time algorithm $O(p)$, which is a significant improvement over the naive $O(p^2n)$ suggested in Section 3.2.4 on page 17. The proof of correctness for this suggestion, however, is left as future work.

5.2 Strict constraints

Max-plus polyhedra can only represent sets of non-strict inequalities. Standard definition of timed automata, however, allows the guards and invariants to contain strict difference constraints. We are currently limited to automata which don't use them, but to make our approach usable in practice, we need to support strict constraints as well.

While it is straightforward to extend the external representation to allow strict inequalities, the same does not hold true for the internal representation, which we use for our algorithms. There are two approaches to solve the same problem–i.e. to represent a polyhedron of which some faces may not be closed–with *classical* polyhedra; both of them are described in (Bagnara et al., 2002). The first approach uses a closed polyhedron of dimension $n+1$ to represent an n-dimensional, not-necessarily-closed polyhedron. The polyhedron represented in this manner is preserved under the usual operations such as intersection, and can be tested for emptiness.

The second approach uses a direct characterization of not-necessarily-closed polyhedra in terms of generating sets. This characterization uses a third set of generators, called *closure points*, to allow for faces that are not closed.

Unfortunately, none of these extensions of classical polyhedra carry directly over to the max-plus realm. Either one of them would have to be changed significantly, or a completely different technique would have to be used.

5.3 Federation data structure

The forward reachability algorithm described in Section 2.1.4 on page 6 needs to check whether newly discovered symbolic states have already been processed – *passed*. The set of passed states can be implemented as a list of such states, but this is very inefficient. It has been shown that a more efficient data structure, such as clock decision diagrams (Larsen et al., 1999) in the zone-based case, may provide a considerable reduction of the space needed to store passed states.

This data structure (also referred to as *federation*) needs to represent finite union of symbolic states, i.e. max-plus polyhedra in our case, and support following two operations efficiently:

1. Checking whether a polyhedron P in internal representation is a subset of the federation.
2. Adding a new polyhedron P in internal representation to the federation.

Note that we currently cannot use a data structure derived from binary decision diagrams, or any other decision structure on which it is possible to compute set complement. The reason is that finite unions of max-plus polyhedra are not closed under set complement. This somehow relates the problem of finding a suitable federation data structure with the problem of representing polyhedra with open faces.

5.4 Normalization operation

In general, the symbolic state space explored during reachability checking may be infinite. One solution is to apply a *normalization* operation to each computed successor, which will make the state space finite while keeping the properties of the original one. Several such operations exist for zones (Behrmann et al., 2006; Bouyer et al., 2005), which mainly differ in the following aspects:

- Whether they are correct for automata with diagonal constraints in guards and invariants.
- To what extent they keep properties of the original system, e.g. whether the normalized system is bisimilar or just in a simulation relation (which is sufficient for reachability) with the original system.
- The coarseness of the abstraction provided by the normalized system. Coarser systems result in smaller state spaces (which may significantly improve performance), but tend to behave differently in the sense of previous point.

While it may be possible that one of the algorithms can be more or less directly adapted to max-plus polyhedra, the problem is that all of them work on the inequalities defining the zone and we would like to avoid converting from generators to inequalities and back. These conversions are costly and we hope it is possible to perform normalization directly on the generators.

BIBLIOGRAPHY

Johan Bengtsson. *Clocks, DBMs, and States in Timed Systems*. PhD thesis, Uppsala University, June 2002.

Roberto Bagnara, Elisa Ricci, Enea Zaffanella, and Patricia M. Hill. Possibly not closed convex polyhedra and the parma polyhedra library. In *Proceedings of the 9th International Symposium on Static Analysis*, volume 2477 of *Lecture Notes in Computer Science*, pages 213–229. Springer-Verlag, 2002.

Mads Chr. Olesen and Kenneth Yrke Jørgensen. opaal. `http://www.opaal-modelchecker.com/`, 2010. Retrieved November 30, 2010.

Xavier Allamigeon, Stéphane Gaubert, and Éric Goubault. Inferring min and max invariants using max-plus polyhedra. In María Alpuente and Germán Vidal, editors, *SAS*, volume 5079 of *Lecture Notes in Computer Science*, pages 189–204. Springer, 2008. ISBN 978-3-540-69163-1.

Peter Butkovič, Hans Schneider, and Sergeĭ Sergeev. Generators, extremals and bases of max cones. *Linear Algebra and its Applications*, 421(2-3):394 – 406, 2007. ISSN 0024-3795. doi: 10.1016/j.laa.2006.10.004. URL `http://www.sciencedirect.com/science/article/B6V0R-4MD95FY-2/2/5196aae1a1b65714f29a59649f9b9301`.

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.

Karel Zimmermann. A general separation theorem in extremal algebras. *Ehkon.-Mat. Obz.*, 13: 179–201, 1977.

Jesper Dyhrberg, Qi Lu, Michael Madsen, and Søren Ravn. Computations on zones using max-plus algebra. Bachelor's project, Aalborg University, 2010. `https://services.cs.aau.dk/cs/tools/library/details.php?id=1274952619`.

Gerd Behrmann, Patricia Bouyer, Kim G. Larsen, and Radek Pelánek. Lower and upper bounds in zone-based abstractions of timed automata. *Int. J. Softw. Tools Technol. Transf.*, 8:204–215, June 2006. ISSN 1433-2779. doi: 10.1007/s10009-005-0190-0.

Patricia Bouyer, François Laroussinie, and Pierre-Alain Reynier. Diagonal constraints in timed automata: Forward analysis of timed systems. In *In Proc. FORMATS'05, vol. 3829 of LNCS*, pages 112–126. Springer, 2005.

Luca Aceto, Anna Ingólfsdóttir, Kim Guldstrand Larsen, and Jiří Srba. *Reactive Systems*. Cambridge University Press, 2007. ISBN 9780521875462.

Xavier Allamigeon, Stéphane Gaubert, and Éric Goubault. Computing the extreme points of tropical polyhedra, 2009. URL http://www.citebase.org/abstract?id=oai:arXiv.org:0904.3436. arXiv:0904.3436.

Kim G. Larsen, Justin Pearson, Carsten Weise, and Wang Yi. Clock difference diagrams. *Nordic J. of Computing*, 6:271–298, September 1999. ISSN 1236-6064. URL http://portal.acm.org/citation.cfm?id=774455.774459.

Stéphane Gaubert and Ricardo D. Katz. The minkowski theorem for max-plus convex sets. *Linear Algebra and its Applications*, 421(2-3):356 – 369, 2007. ISSN 0024-3795. doi: 10.1016/j.laa.2006.09.019. URL http://www.sciencedirect.com/science/article/B6V0R-4MD95FY-1/2/c170c6864b9ed2fccff0caa56e2226eb.

Xavier Allamigeon, Stéphane Gaubert, and Éric Goubault. The tropical double description method. In Jean-Yves Marion and Thomas Schwentick, editors, *STACS*, volume 5 of *LIPIcs*, pages 47–58. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010. ISBN 978-3-939897-16-3.

Sergio Yovine. Kronos: A Verification Tool for Real-Time Systems. *International Journal on Software Tools for Technology Transfer*, 1(1–2):123–133, 1997.

Rajeev Alur, Costas Courcoubetis, David L. Dill, Nicolas Halbwachs, and Howard Wong-Toi. An implementation of three algorithms for timing verification based on automata emptiness. In *IEEE Real-Time Systems Symposium*, pages 157–166, 1992.

Uppsala University and Aalborg University. UPPAAL. http://www.uppaal.com/, 2010. Retrieved May 25, 2010.

Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5:345–, June 1962. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/367766.368168. URL http://doi.acm.org/10.1145/367766.368168.

Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183 – 235, 1994. ISSN 0304-3975. doi: 10.1016/0304-3975(94)90010-8. URL http://www.sciencedirect.com/science/article/B6V1G-45D9T5J-X/2/67d800a59735fe5a3c21e8ac23488693.