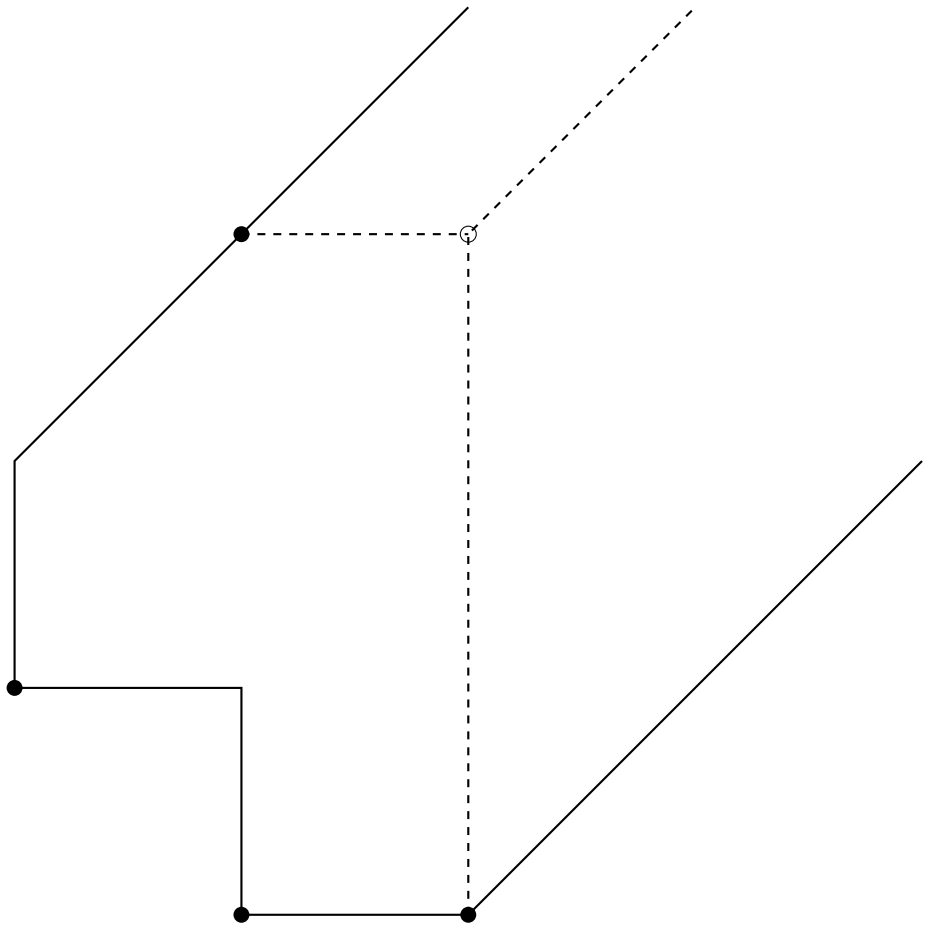
Reachability Analysis for Timed Models using Max-Plus Algebra



DAT6, Spring 2011
Group f11d604a
8th semester
Department of Computer Science
Aalborg University
June 9th 2011

Title:

Reachability Analysis for Timed Models using Max-Plus Algebra

Theme:

Model checking

Period:

DAT6, Spring 2011

Group:

f11d604a

Group members:

Qi Lu

Michael Madsen

Søren Ravn

Supervisors:

Uli Fahrenberg

Kim Guldstrand Larsen

Circulation: 6

Pages: 49

Ended: June 9th 2011

Abstract:

In this report, we elaborate on the work done on showing max-plus polyhedra capable of representing the clock space for reachability checking of timed models. We show that there exist models where max-plus overaproximation is exact, but DBM overapproximation is not. We prove an extrapolation algorithm for single-dimensional max-plus polyhedra, and describe principles for an extrapolation algorithm for general max-plus polyhedra. Finally, we implement our max-plus polyhedra representation on top of a model checker for timed-arc Petri nets, and show that for some models where DBM overapproximation fails to provide correct results, max-plus overapproximation is faster than exact DBM analysis.

PREFACE

This project began February 1st 2011 and ended June 9th 2011. The report documents the process and performance 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.

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.

The project is a continuation of our project from previous semesters, and contains some content from the associated project reports. The following parts are mostly unchanged from the previous report:

- Chapter 1 on page 3
- Section 2.1.1 on page 4
- Section 2.2 on page 12
- Section 2.3 on page 15
- Chapter 3 on page 19 excluding Sections 3.2.4 on page 21, 3.3.2 on page 25 and 3.4 on page 29

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.

Chapter 4 describes elements which may be used in an extrapolation algorithm to ensure termination of reachability analysis.

Chapter 5 contains an overview of our implementation of this approach.

Chapter 6 concludes the report and discusses future work and possible improvements.

Lastly, we would like to thank Morten and Lasse Jacobsen for their work on the verifytapn model checker, and their willingness to assist us in implementing this approach, Alexandre David for his insight into UPPAAL, Jiří Srba for his insight into timed-arc Petri nets, Xavier Allamigeon, Stéphane Gaubert and Éric Goubault for inspiring the project, and Jesper Dyhrberg and Martin Milata for their 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 the logical order of events, but also the time at which events are 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 is a well-established technique for performing real-time model checking, and several tools exist for this purpose, e.g. UPPAAL [30], TAPAAL [28] and KRONOS [32]. Common models for real-time model checking include *timed automata* [5] and *timed-arc Petri nets* [12, 20].

A zone is 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 is as a Difference Bound Matrix, or DBM for short [8, 16], 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 [3, 4]. 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 and Petri nets, 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 models

The concept of real-time model checking is not specific to one particular type of model, but can be applied to any class of models containing a notion of real time.

Two of the more well-known classes are timed automata and timed-arc Petri nets. For this report, we have used the theory of timed automata when developing our algorithms, but the implementation performs model checking on timed-arc Petri nets. Despite the significant differences in behavior, networks of timed automata and timed-arc Petri nets are equivalent [14], and the same underlying data structure for model checking can be used in either case, simply by adapting the reachability algorithm itself to the different data structure. TAPAAL [28], a tool similar to UPPAAL [30], but for timed-arc Petri nets, currently performs model checking by converting the Petri net into a timed automaton, and performing model checking on it using UPPAAL.

2.1.1 Timed automata

Timed automata, or TA for short, 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 [7]; a more thorough overview of relevant notions can be found for example in [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 is the destination, g is the guard of the edge and r is the set of clocks to be reset after transition.

Generally, transitions may also include input and output actions; these are used to synchronize two timed automata running in parallel, by requiring that an input action and its corresponding output action in a different automaton is taken at the same time. This allows for a network of timed automata, where all the automata in the network are run in parallel [1]. In the interest of simplicity, however, we ignore the actions in this report, as they do not make any practical difference for our purpose.

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 restrict $\mathcal{B}(X)$ to $\sim \in \{\leq, =, \geq\}$, which results in weaker model of timed automata that is still interesting. See section 6.2 on page 44 for further discussion.

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] $(S, s_0, \rightarrow)$:

- $S = \{(l, v) \mid l \in L, v \in \mathbb{R}^X_{\geq 0}, v \vDash I(l)\}$ 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,
- $\rightarrow \subseteq S \times S$ are transitions such that:
  - $(l, v) \rightarrow (l', v')$ if $l \xrightarrow{g,r} l'$, $v \vDash g$, $v' = v[r]$,
  - $(l, v) \rightarrow (l, v + \delta)$ 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. [1].

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 values and whether or not the value is greater than some fixed constant.

Definition 2.1 (Region Equivalence)**.** Given two clock valuations $\mathbf{v}, \mathbf{v}'$ and upper bounds on all clocks $\{k_x \mid x \in C\}$, where C is the set of clocks, for the constraints on the form $x \leq c$ and $x \geq c$ it holds $c \leq k_x$.
The integer part of a clock valuation $\mathbf{v}_x$ is denoted $\lfloor \mathbf{v}_x \rfloor$ and the fractional part is denoted $fract(\mathbf{v}_x)$. The equivalence relation $\sim$ between two clock valuations is then defined as $\mathbf{v} \sim \mathbf{v}'$ iff all of the following conditions hold:

1. $\forall x \in C. \lfloor \mathbf{v}_x \rfloor = \lfloor \mathbf{v}'_x \rfloor \vee (\mathbf{v}_x > k_x, \mathbf{v}'_x > k_x)$,
2. $\forall x, y \in C, \mathbf{v}_x \leq k_x, \mathbf{v}_y \leq k_y. fract(\mathbf{v}_x) \leq fract(\mathbf{v}_y)$ iff $fract(\mathbf{v}'_x) \leq fract(\mathbf{v}'_y)$,
3. $\forall x \in C, \mathbf{v}_x \leq k_x. fract(\mathbf{v}_x) = 0$ iff $fract(\mathbf{v}'_x) = 0$.

A clock region is the set of clock valuations induced by the equivalence relation $\sim$ [6, Definition 4.3].

Definition 2.2 (Region Closure)**.** The region closure of a set of clock valuations, S, is the minimal set S' s.t. for all $\mathbf{p} \in S, \mathbf{p}' \sim \mathbf{p}$, $\mathbf{p}' \in S'$.

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.

[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.

Zones and zone graphs

Zones are sets of clock valuations that satisfy a conjunction of clock constraints. Formally, $Z \subseteq \mathbb{R}^X_{\geq 0}$ 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 2.3. *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 [1].*

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 *extrapolation* (sometimes also referred to as *normalization*), 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 [9, 13], 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 15 is devoted to describing this data structure.

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: Forward reachability

```
1:  Waiting := {(l_0, Z_0)}
2:  Passed := ∅
3:  while Waiting ≠ ∅ do
4:     choose and remove (l, Z) from Waiting
5:     if l = sandZ ∧ φ ≠ ∅ then
6:        return TRUE
7:     end if
8:     if Z ⊈ Z' for all (l, Z') ∈ Passed then
9:        Passed := Passed ∪ {(l, Z)}
10:       Waiting := Waiting ∪ {(l', Z') | (l, Z) ⇝ (l', Z') ∧ Z' ≠ ∅}
11:    end if
12: end while
13: return FALSE
```

We can see from the algorithm that in order to develop a data structure that supports forward reachability analysis, 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 2.1.1 on the preceding page and an additional check whether the zone is empty.

2.1.2 Timed-arc Petri nets

The concept of timed-arc Petri nets, denoted TAPN, derives from *Petri nets* [24], PN. It is a timed extension of PN where *ages* are associated with tokens, allowing tokens to be considered similar to clocks in timed automata. *Arcs* going from a place to a transition are labeled with a *time interval*, which a token age has to satisfy in order for the transition to be enabled.

Basic definitions

The set of well-formed time intervals $\mathcal{I}$ is defined by the following abstract syntax containing a lower bound $l \in \mathbb{N} \cup \{0\}$ and an upper bound $u \in \mathbb{N}$, s.t. $l < u$.

$$I ::= [l, u] \mid [l, l] \mid (l, u] \mid [l, u) \mid (l, u) \mid [l, \infty) \mid (l, \infty)$$

Given an age a, these intervals represent, respectively, the constraints $l \leq a \leq u, a = l, l < a \leq u, l \leq a < u, l < a < u, l \leq a < \infty$, and $l < a < \infty$. We can see the intersection of any finite number of time intervals $I_1, I_2, \ldots, I_n \in \mathcal{I}$ is either empty or belongs to $\mathcal{I}$.

Additionally, we can specify *invariants* in TAPNs. The abstract syntax of invariants $\mathcal{I}_{Inv} \subset \mathcal{I}$ is given by:

$$I_{Inv} ::= [0, 0] \mid [0, u] \mid [0, u) \mid [0, \infty)$$

A *timed-arc Petri net* is a tuple $N = (P, T, F, c, Inv)$ where

- P is a finite set of *places*,

- T is a finite set of *transitions* such that $P \cap T = \emptyset$
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of normal arcs, and is called *flow relation*, and
- $c : F|_{P \times T} \to I$ is a *time constraint* assigning time intervals to arcs from places to transitions.
- $Inv : P \to I_{Inv}$ is a function which assigns to every place an *invariant*.

Due to the definition of F, Petri nets can be considered as directed bipartite graphs, since it is not possible to have an arc directly between two places or two transitions.

A *marking* M on a TAPN N is a function $M : P \to \mathcal{C}(\mathbb{R}^+)$ where $\mathcal{C}(\mathbb{R}^+)$ denotes the set of finite multisets on $\mathbb{R}^+$, representing the distribution of tokens. $\mathcal{M}(N)$ denotes the set of all markings over N, and $\mathcal{M}(p)$ denotes the number of tokens in place p for marking M. A *marked* TAPN is a pair (N, M_0) where N is a TAPN and M_0 is the initial marking on N. Similar to the initial state of a timed automaton, we only allow initial markings where all tokens have age 0.

We define the *preset* or *input places* of a given transition $t \in T$ as $\bullet t = \{p \in P \mid (p, t) \in F\}$ and the *postset* or *output places* of t as $t\bullet = \{p \in P \mid (t, p) \in F\}$.

Semantics

A TAPN has two basic transition behaviors: *firing* and *time-elapsing*. When a transition is fired, one token in each place of its preset is consumed and a new token of age 0 is created in each place of its postset. The transition relation of the firing transition t from marking M to marking M' is written as $M \xrightarrow{t} M'$ and the time-elapsing transition is written as $M \xrightarrow{d} M'$, where $d \in \mathbb{R}_{\geq 0}$ is the time elapsing.

In order for a transition to be enabled, some conditions must be satisfied:

Definition 2.4 (Enabledness)**.** Given a TAPN $N = (P, T, F, c)$, a transition $t \in T$ is enabled in a marking M, iff for all $p \in \bullet t$, there exists a $x \in M(p)$, s.t. $x \in c(p, t)$.

In other words, a token in a place p may participate in enabling a transition t if the age of the token lies within the time interval of the arc from p to t. In order for the transition to be enabled, this must be satisfied by all places in the preset.

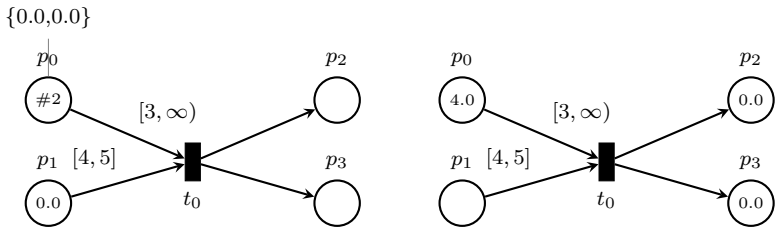
Definition 2.5 (Firing rule)**.** Given a TAPN $N = (P, T, F, c)$ and a marking M enabling a transition $t \in T$, t can be fired to create a new marking M' defined by $\forall p \in P. M'(p) = (M(p) \setminus C_t^-(p, t)) \cap C_t^+(t, p)$, where $C_t^-(p, t)$ and $C_t^+(t, p)$ denote the ages of tokens in place p, as given by the following equations:

$$C_t^-(p,t) = \begin{cases} \{x\}, & \text{if } p \in \bullet t \text{ s.t. } x \in M(p) \wedge x \in c(p,t) \\ \emptyset, & \text{otherwise} \end{cases}$$

$$C_t^+(t,p) = \begin{cases} \{0\}, & \text{if } p \in t\bullet \\ \emptyset & \text{otherwise.} \end{cases}$$

Definition 2.6 (Time-elapsing)**.** Let $N = (P, T, F, c)$ be a TAPN. A time-elapsing transition, where $r \in \mathbb{R}^+$ is the time elapsing, is defined as $M \xrightarrow{r} M'$ iff for all $p \in P$, $M'(p) = M(p) + r$.

Figure 2.1 on the next page contains an example of a small TAPN N before, Figure 2.1a, and after, Figure 2.1b, a time-elapse of 4 and firing transition t_0. We see tokens in places p_0 and p_1 are consumed and new tokens of age 0 are created in places p_2 and p_3. A formal definition of the net N is:



(a) TAPN N before firing transition t_0.

(b) N after firing t_0.

Figure 2.1: A timed-arc Petri net N.

- $P = \{p_0, p_1, p_2, p_3\}$,
- $T = \{t_0\}$,
- $F = \{(p_0, t_0), (p_1, t_0), (t_0, p_2), (t_0, p_3)\}$,
- $c = \{(p_0, t_0) \to [3, \infty), (p_1, t_0) \to [4, 5]\}$,
- $M_0 = \{p_0 \to \{0, 0\}, p_1 \to \{0\}\}$

Special arcs

Some literature, such as Bouyer et al. [14], as well as tools like TAPAAL [28], use arcs beyond the simple arcs described so far. Here, we will describe the meaning of these additional arc types.

Transport arcs A transport arc goes from one place to another via a transition. When the corresponding transition is fired, the token is moved to the new place, rather than being consumed and re-created [29].

In TAPAAL, transport arcs are drawn using a filled diamond, and a numeric identifier on the arcs to disambiguate when a transition involves multiple transport arcs.

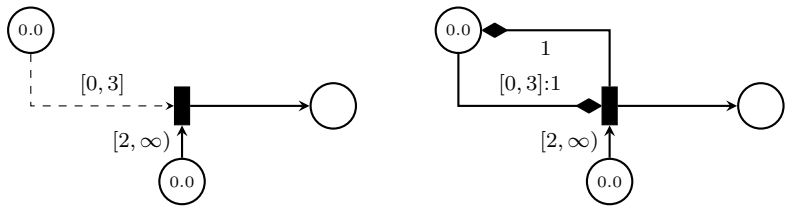
Testing arcs/read arcs A testing arc works like a normal arc, but the token used is not consumed when firing the transition. Instead, the token is left as-is, continuing to age without interruption. The addition of testing arcs is sufficient to make networks of timed automata and timed-arc Petri nets equally expressive [14], allowing us to convert a network of timed automata to a Petri net.

TAPAAL does not implement testing arcs directly, but their semantics can be emulated using a transport arc going to the transition and back to the original place; see Figure 2.2 on the facing page for a visual example.

Symbolic semantics

Conceptually, the age of a token functions like a clock. Consequently, we can consider a token to actually be a clock, thereby allowing us to directly apply the concept of zones as described in Section 2.1.1 on page 7.

A key difference is that unlike normal zones, Petri nets do not allow us to express constraints on differences between clocks, so effectively, each token merely has a range of valid ages. However,



(a) TAPN using a testing arc.

(b) Transport arc simulating a testing arc.

Figure 2.2: Special arcs in a TAPN.

we can still apply all of the other properties of zones themselves, as neither delay nor reset can introduce new difference constraints, and intersection will only introduce a difference constraint when actually intersecting with a difference constraint.

Since Petri nets do not have locations, we define a new symbolic successor relation $\rightsquigarrow$:

- $(place, age) \rightsquigarrow (place, age^{\uparrow} \wedge Inv(place))$ – delay successor,
- $(place, age) \rightsquigarrow (place', age' \wedge Inv(place'))$ if there exists $t \in T$ s.t. $(place, age) \xrightarrow{t} (place', age')$ – discrete transition successor.

Reachability for TAPN

Like for timed automata, we are interested in determining whether a given marking is reachable.

Definition 2.7 (Reachability)**.** Let (N, M_0) be a marked TAPN. A marking $M \in \mathcal{M}(N)$ is reachable if $M_0 \xrightarrow{*} M$, where the transition $\xrightarrow{*}$ is an arbitrary combination of *firing* and *time-elapsing* transitions.

For an arbitrary TAPN, the reachability problem is undecidable [26], as there is no limit on the number of tokens in the TAPN. However, by imposing a bound k on the number of tokens, reachability becomes decidable [21]. We call such Petri nets *k-bounded*.

Definition 2.8 (k-boundedness)**.** A marked TAPN (N, M_0) is *k-bounded* if the total number of token $|M| \leq k$ for any reachable marking M.

The reachability algorithm for k-bounded TAPNs is given by Algorithm 2 on the following page, and works on the same basic principle as the reachability algorithm for timed automata, but using token placements instead of locations. The reachability property ψ is a subset of CTL as defined in Jacobsen and Jacobsen [21]. Additionally, we define $(place_0, age_0)$ as the initial marking.

Algorithm 2: `Reachability of` k`-bounded TAPN`

$Waiting := \{(place_0, age_0)\}$
$Passed := \emptyset$
while $Waiting \neq \emptyset$ **do**
  choose and remove $(place, age)$ from $Waiting$
  if $(place, age) \vDash \psi$ **then**
    return TRUE
  end if
  if $age \nsubseteq age'$ for all $(place, age') \in Passed$ **then**
    $Passed := Passed \cup \{(place, age)\}$
    $Waiting := Waiting \cup \{(place', age') \mid (place, age) \rightsquigarrow (place', age') \wedge age' \neq \emptyset\}$
  end if
end while
return FALSE

By comparing Algorithm 2 to Algorithm 1 on page 8, it can easily be seen that the only difference is the substitution of locations and zones with placements and token ages.

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 $ab = a \otimes b$.

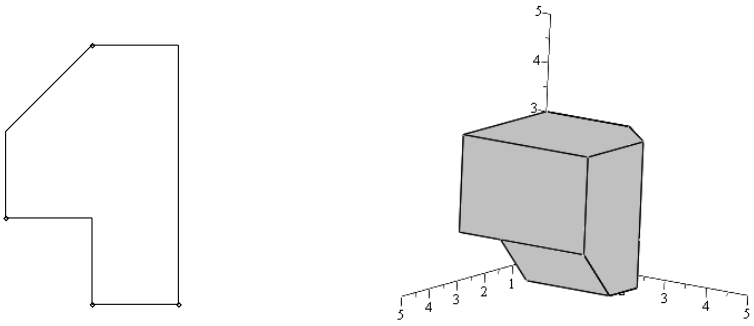
The definitions of addition and multiplication in max-plus algebra can be extended to vectors and matrices in the usual way – addition: $(\mathbf{v}_1, \ldots, \mathbf{v}_n) \oplus (\mathbf{w}_1, \ldots, \mathbf{w}_n) = (\mathbf{v}_1 \oplus \mathbf{w}_1, \ldots, \mathbf{v}_n \oplus \mathbf{w}_n)$, multiplication by scalar: $\alpha \otimes (\mathbf{v}_1, \ldots, \mathbf{v}_n) = (\alpha \otimes \mathbf{v}_1, \ldots, \alpha \otimes \mathbf{v}_n)$, 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 $a \oplus b = \mathbb{0}$.

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 $\mathbb{R}_{max}^n$, 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] They are called tropical in honour of Imre Simon, who pioneered the field, apparently because he was from Brazil.



(a) 2D polyhedron. (b) 3D polyhedron.

Figure 2.3: Examples of polyhedra in 2D and 3D.

2.2.1 Polyhedra over max-plus algebra

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

Definition 2.9. A convex max-plus polyhedron is a subset of $\mathbb{R}_{max}^n$ 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 2.3, 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 [33], a general introduction can be found for example in Gaubert and Katz [19].

2.2.2 Representations 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 and algorithms for conversion between these are described by Allamigeon et al. [2, 3, 4].

Systems of constraints

One possible representation of max-plus polyhedra is by a finite set of linear inequalities, i.e. inequalities of the form $\mathbf{a}\mathbf{x} \oplus b \geq \mathbf{c}\mathbf{x} \oplus d$, 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*.

This representation is very similar to a DBM, and it is indeed possibly to easily convert a DBM to a polyhedron in external representation through simple rewriting of clock constraints. As an example, a constraint $x_i - x_j \geq n$ can be written as the max-plus inequality $\mathbf{a}\mathbf{x} \oplus b \geq \mathbf{c}\mathbf{x} \oplus d$ with $\mathbf{a}_k = \mathbb{0}$ for $k \neq i$ and $\mathbf{a}_i = \mathbb{1}$, $\mathbf{c}_k = \mathbb{0}$ for $k \neq j$ and $\mathbf{c}_j = n$, and $b = d = \mathbb{0}$; see Dyhrberg et al. [17] for further details.

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 $A, B \subseteq \mathbb{R}^n_{max}$. 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* [19].

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

Theorem 2.10. *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. [19]*

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* [19, 2, 15]. 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.

The internal representation is very different from the external representation, and the conversion between these representations involves solving max-plus matrix equations, which is computationally rather expensive [2]. As a result, this report will only deal with the internal representation.

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 $P = \mathrm{co}(V) \oplus \mathrm{cone}(W)$ 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 [16], or DBMs for short, is currently one of the most efficient data structures for representing zones [10]. 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 2.1.1 on page 7, 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. However, as we will not deal with strictness in this report, we will simplify the notation where appropriate by using the values directly.

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 are smaller than non-strict. This provides a logical ordering of constraints $(n_1, \preceq) < (n_2, \preceq)$ for $n_1 < n_2$ or $n_1 = n_2 \wedge (n_1, <) < (n_2, \leq)$. 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. In other words, $(n_1, \preceq_1) + (n_2, \preceq_1) = (n_1 + n_2, \preceq_1 + \preceq_2)$ where $(< + \leq = <)$.

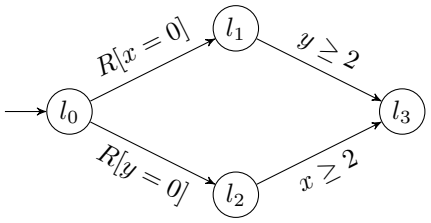


Figure 2.4: Timed automaton used for this example.

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 [18], and then converting back to matrix form [10].

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 freeing a clock.

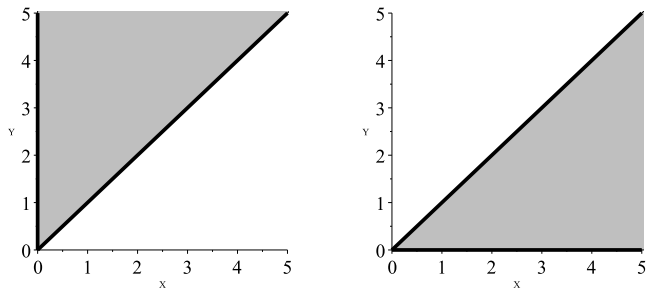
For computing the relationship between two DBMs and intersecting with one clock, quadratic algorithms are provided. This is also the case for the backward delay, used in backward reachability. DBMs also provide algorithms for normalization, the commonly used algorithm being quadratic as well.

For comparison of the DBM algorithms to our proposed algorithms on max-plus polyhedra we refer the reader to Chapter 3 on page 19.

2.3.2 Comparison with max-plus polyhedra

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 2.4 shows an example of a small TA of two clocks which 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 the forward 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. A visual indication of the two zones can be seen in Figure 2.5 on the next page. In both locations, we are ready to take the transition out.



(a) l_1 (b) l_2

Figure 2.5: Location l_1 and l_2 after delay.

l_1**:**

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

l_2**:**

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

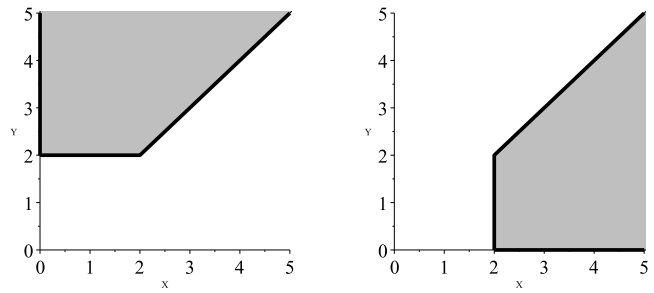
Continuing the reachability algorithm we calculate the state space going from l_1 and l_2 to l_3. At first these are stored as two separate states, containing the zones seen in Figure 2.6 on the following page.

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

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

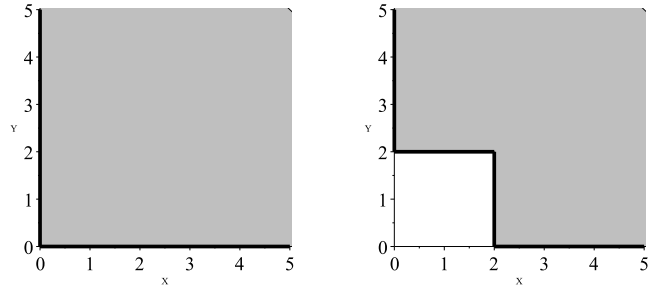
l_3 **via** l_2**:**

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



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

Figure 2.6: The two different possible states in location l_3.



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

Figure 2.7: Union of zones from the two l_3 states.

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 DBM and max-plus 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 whether we make an overapproximating union. Opting for the overapproximation, we get the following state for l_3.

l_3 union:

$$\begin{aligned} P &= co(\{\binom{2}{0}, \binom{0}{2}\}) \oplus cone(\{\binom{-\infty}{0}, \binom{0}{-\infty}\}) \\ D &= \begin{bmatrix} 0 & 0 & 0 \\ \infty & 0 & \infty \\ \infty & \infty & 0 \end{bmatrix} \end{aligned}$$

Figure 2.7 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.

Note, however, that max-plus union and exact union will not always coincide.

CHAPTER 3

ALGORITHMS ON MAX-PLUS POLYHEDRA

With inspiration from Bengtsson's PhD thesis [10] 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. Most of the algorithms are performed on polyhedra P always consiststing of a convex, V, and a linear set of generators, W. A few of the algorithms are performed on polyhedra represented as cones, denoted G. Additionally, note that n always denotes the number of clocks for a system, and p the number of 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 14. These are concerned with the conversion from polyhedra to cones (Algorithm 3) and back (Algorithm 4 on the following page); both having a complexity of $O(pn)$.

Algorithm 3: `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 4: `cone-to-poly`(G)

$V, W := \emptyset$
for all $\mathbf{g} \in G$ **do**
  if $\mathbf{g}_{n+1} = -\infty$ **then**
    $W := W \cup \{\mathbf{g}\}$
  else
    $g' = (g_1, \ldots g_n)$
    for $i := 1$ **to** n **do**
      $\mathbf{g}_i := \mathbf{g}_i + -\mathbf{g}_{n+1}$
    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 [2].

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`$(G, \mathbf{x})$

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 3 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, using the algorithm provided by Allamigeon et al. [2]. 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.

Algorithm 5: `contains-point`$(G, \mathbf{x})$

```
for all g^i ∈ G do
  y_i := min_{1≤j≤n+1}(x_j − g^i_j)
end for
for all 1 ≤ j ≤ n + 1 do
  z_j := max_{g^i∈G}(y_i + g^i_j)
end for
if x = z then
  return true
end if
return false
```

3.2.3 Inclusion test – `contains`(P, P')

Inclusion checking is a crucial operation when doing state space exploration. This is needed to determine whether a given state has already been visited, and hence does not need to be traversed again.

Inclusion checking is a logical extension of set inclusion, as checking if P' is included in P is the same checking if the set of all points generated by P' is included in the set of all points generated by P. 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 6: `contains`(P, P')

```
G := poly-to-cone(P)
G' := poly-to-cone(P')
for all g' ∈ G' do
  if ¬contains-point(G, g') then
    return false
  end if
end for
return true
```

3.2.4 Constraint satisfaction – `satisfied`$(P, x_i - x_j \sim c)$

Determining if a conjunction of constraints are satisfied requires expensive intersections, so it can be desirable to first check each constraint individually. For example, Petri nets can use this to determine enabledness of a transition. This algorithm has a complexity of $O(p)$, in contrast with the DBM algorithm which is trivially $O(1)$.

For Petri nets, x_j will always correspond to x_0, as Petri nets are incapable of directly expressing the difference between the age of two tokens.

Algorithm 7: satisfied($P, x_i - x_j \sim c$)

```
if ∼ is = then
  return satisfied(P, x_i − x_j ≤ c) and satisfied(P, x_i − x_j ≥ c)
if x_j = x_0 then {x_j is not a clock}
  return satisfied-single(P, x_i ∼ c)
else
  return satisfied-diff(P, x_i − x_j ∼ c)
```

Algorithm 8: satisfied-single($P, x_i \sim c$)

```
for all v ∈ V do
  if v_i ∼ c then
    return true
if ∼ is ≥ then
  for all w ∈ W do
    if w_i ≠ −∞ then
      return true
return false
```

Algorithm 9: satisfied-diff($P, x_i - x_j \sim c$)

```
if ∼ is ≥ then
  for all v ∈ V do
    if v_i ≥ v_j + c then
      return true
  for all w ∈ W do
    if w_i = −∞ and w_j = −∞ then
      continue
    if w_i ≥ w_j + c then
      return true
else
  return satisfied-diff(P, x_j − x_i ≥ −c)
return false
```

Proof. To prove the algorithm, we need to prove every possible branch. Essentially, there are three branches the algorithm can take as an equality can be expressed by $\leq \wedge \geq$ and the differece constraint $x_i - x_j \geq c$ is the same as $x_j - x_i \leq -c$. Let $P = \text{co}(V) \oplus \text{cone}(W)$ be a polyhedron, and $\mathbf{p}$ a point in P, given by:

$$\mathbf{p} = \bigoplus_{l=1}^{p} \alpha^l \mathbf{v}^l \oplus \bigoplus_{m=0}^{q} \beta^m \mathbf{w}^m$$

where $\alpha, \beta \in \mathbb{R}_{max}$ and $\bigoplus_{l=1}^{p} \alpha^l = 0$.

$x_i \leq c$: For a single clock $\leq$ constraint to be satisfied, the algorithm requires at least one convex generator satisfying the constraint, i.e. there exists a $\mathbf{v} \in V$ s.t. $\mathbf{v}_i \leq c$. To show this is sufficient, we can chose the corresponding $\alpha = 0$ and the rest of the scalars $\alpha, \beta = -\infty$. Let $\mathbf{u}$ denote the generator satisfying the constraint. Since we are scaling all other generators by $-\infty$ we get:

$$\mathbf{p} = 0 \otimes \mathbf{u} \oplus \mathbb{0}$$

where $\mathbb{0}_j = -\infty, 1 \leq j \leq n$ and therefore for this combinations of scalars, $\mathbf{p} = \mathbf{u}$. Since we know $\mathbf{u}_i \leq c$, there trivially exists a $\mathbf{p}$ s.t. $\mathbf{p}_i \leq c$.

In the opposite case, when there does not exist a $\mathbf{v} \in V$ s.t. $\mathbf{v}_i \leq c$, then it must be true that there does not exist a $\mathbf{p}.\mathbf{p}_i \leq c$. This follows trivially from the fact that $\min_{\mathbf{p}\in P}(\mathbf{p}_i) = \min_{\mathbf{v}\in V}(\mathbf{v}_i)$ since $\bigoplus_{l=1}^{p} \alpha^l = 0$.

$x_i \geq c$: For a single clock $\geq$ constraint to be satisfied, the algorithm states that if there exists a $\mathbf{v} \in V$ s.t. $\mathbf{v}_i \geq c$, or a $\mathbf{w} \in W$ s.t. $\mathbf{w}_i \neq -\infty$), then there exists a $\mathbf{p}$ where $\mathbf{p}_i \geq c$. The convex satisfiability works in exactly the same way as for the less than constraint, and follows exactly the same proof strategy.

The linear part only needs to be considered if there are no convex generators satisfying the constraint. This means we know $\bigoplus_{j=1}^{p} \alpha^j \mathbf{v}_i^j < c$. Thus, if we have a linear generator $\mathbf{w}$ where $\mathbf{w}_i \neq -\infty$, we can choose the corresponding scalar $\beta = |\mathbf{w}_i| + c$ and the remaining linear scalers as $-\infty$. This means that $\bigoplus_{j=1}^{p} \beta^j \mathbf{w}_i^j \geq c$, which in turn means:

$$\mathbf{p} = \bigoplus_{j=1}^{p} \alpha^j \mathbf{v}^j \oplus \bigoplus_{j=1}^{p} \beta^j \mathbf{w}^j \implies \mathbf{p}_i \geq c$$

If there does not exist a $\mathbf{w} \in W$ where $\mathbf{w}_i \neq -\infty$), it follows trivially that our linear set cannot satisfy the constraint.

$x_i - x_j \geq c$: For the difference constraint, the algorithm states that if there exists a $\mathbf{v} \in C$ s.t. $\mathbf{v}_i - \mathbf{v}_j \geq c$ or a $\mathbf{w} \in W$ s.t. $\mathbf{w}_i \neq -\infty, \mathbf{w}_j \neq -\infty, \mathbf{w}_i - \mathbf{w}_j \geq c$) then there exists a $\mathbf{p} \in P$ where $\mathbf{p}_i - \mathbf{p}_j \geq c$. Again, the convex part of the disjunction can be shown in the same way as for single-clock constraints.

For the linear part, standard math is used when dealing with negative infinity, so $-\infty - x = -\infty$ and $x - (-\infty) = \infty$. The remaining case, $-\infty - (-\infty)$, is already handled, since this means that no value can be added by this generator in the given dimensions, as $\beta \otimes -\infty = -\infty$. The calculations to see that is sufficient are straightforward:

$$\mathbf{w}_i - \mathbf{w}_j \geq c \implies \mathbf{p}_i - \mathbf{p}_j \geq c$$

since we can scale $\mathbf{w}$ with any value. If $\mathbf{w}_i = -\infty$ and $\mathbf{w}_j \neq -\infty$ then:

$$-\infty - \mathbf{w}_j = -\infty < c \implies \mathbf{p}_i - \mathbf{p}_j \ngeq c$$

Conversely, if $\mathbf{w}_i \neq -\infty$ and $\mathbf{w}_j = -\infty$ then:

$$\mathbf{w}_i - (-\infty) = \infty \geq c \implies \mathbf{p}_i - \mathbf{p}_j \geq c$$

The opposite case, where $\mathbf{w}_i - \mathbf{w}_j \leq c$, requires no additional explanation, since it is equivalent to $\mathbf{w}_j - \mathbf{w}_i \geq -c$.

□

3.3 Transformations

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

3.3.1 Constraint intersection – `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 11) 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. [3]. 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 $\mathbf{a}$ and $\mathbf{b}$ 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 $\mathbf{a}$ will be a vector the components of which will be -∞, except for the i'th value, which will be 0. Similarly, $\mathbf{b}$ will consist of negative infinities except at the j'th place, which will be c.

Max-plus polyhedra do not explicitly represent an x_0 clock, but the additional element introduced by converting to a cone is used instead.

Like for constraint satisfaction, Petri nets do not allow to directly express a difference constraint, so in $x_i - x_j \sim c$, x_j will always be x_0.

Algorithm 10: `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 11: `intersect-halfspace`$(G, \mathbf{a}, \mathbf{b})$

$G^{\leq} := \{\mathbf{g} \in G \mid \mathbf{a} \otimes \mathbf{g} \leq \mathbf{b} \otimes \mathbf{g}\}$
$G^{>} := \{\mathbf{g} \in G \mid \mathbf{a} \otimes \mathbf{g} > \mathbf{b} \otimes \mathbf{g}\}$
$H := G^{\leq}$
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 the number of computed generators. As the conversions from and to 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 [17]. 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 12: `up`(P)

$W := W \cup V$

The definition of a delayed zone shown in section 2.1.1 on page 7 is directly transferable to max-plus polyhedra: $P^{\uparrow} = \{\mathbf{v} + d \mid \mathbf{v} \in P, d \in \mathbb{R}_{\geq 0}\}$, that is, any point in $P^{\uparrow}$ can be obtained by adding a non-negative distance d to some point from P. The details of the proof are described in our previous project report [23].

Alternative delay algorithm

As with all the transforming algorithms, some redundancy is potentially introduced. Cleanup of the generators, as will be explained in Section 3.5.1 on page 29, is subsequently needed. When delaying, we can calculate exactly which generators from the convex set are needed in the linear set to express the delay, thereby reducing the amount of redundancy introduced. With this reduction, the amount of work for the subsequent cleanup algorithm is also reduced. Essentially, a delayed polyhedon consists of a convex set, imposing a lower bound, and a linear set giving the extreme rays. Thus, if a generator in the linear set can be generated by a combination of the remaining linear generators, it is redundant. From Algorithm 12, we see that the set of potential linear generators is the combination of the cone and convex set. Therefore, we need to check that any generator we want to add cannot be generated by the existing generators in W or by the remaining generators in V. This leads us to an algorithm for calculating the exact set of extra generators needed in the cone set to represent the delay. The set of generators to copy can be expressed as:

$$\{\mathbf{x} \mid \mathbf{x} \neq \bigoplus_{i=1}^{p} \gamma_i \mathbf{v}^i \oplus \bigoplus_{i=1}^{q} \beta_i \mathbf{w}^i\}$$

where $\mathbf{x} \in V, \mathbf{v}^i \in V \setminus \{\mathbf{x}\}$, $\mathbf{w}^i \in W$, $p, q \in \mathbb{N}$ and $\gamma, \beta \in \mathbb{R}_{max}$.

Algorithm 13: `up-alt`(P)

$V', W' = \emptyset$
for all $\mathbf{x} \in V$ **do**
  $V = V \setminus \mathbf{x}$
  if $\mathbf{x} \neq \bigoplus_{i=1}^{p} \gamma_i \mathbf{v}^i \oplus \bigoplus_{i=1}^{q} \beta_i \mathbf{w}^i$ **then**
    $W' = W' \cup \{\mathbf{x}\}$
    $V = V \cup \{\mathbf{x}\}$
  else
    $V' = V' \cup \{\mathbf{x}\}$
  end if
end for
$V = V \cup V'$
$W = W \cup W'$

To give an example, figure 3.1 on the following page shows a polyhedron and the result of running `up-alt`. We see that the algorithm only copies $\mathbf{v}^3$ of the convex generators to the cone set, copy denoted $\mathbf{w}^3$, since the other generators, $\mathbf{v}^1$ and $\mathbf{v}^2$, can be generated linearly from the rest. However, it will neither remove $\mathbf{w}^2$ nor $\mathbf{v}^3$ which becomes redundant generators from the modification; thus, cleanup is still required to get the polyhedron in minimal form.

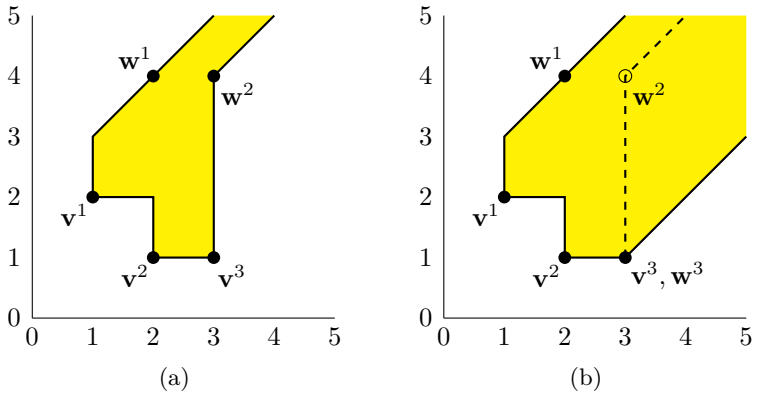


Figure 3.1: The generators of a max-plus polyhedron before and after running `up-alt`.

Complexity analysis Initially, we see that the complexity of the alternative delay algorithm, $O(p^2n)$, is worse than that of the original delay algorithm, $O(pn)$. However, as mentioned above, both algorithms require a subsequent cleanup of $O(p^2n)$, effectively resulting in the same complexity for both algorithms. The amount of work done by the original delay algorithm is less than the alternative one, but as the new algorithm reduces the number of generators that need to be copied, the amout of work performed by cleanup is reduced correspondingly, allowing for a potential performance increase.

Essentially, this approach provides some hints which allow the cleanup algorithm to perform less work. We can use similar principles to adjust the other algorithms, and go even further to define specialized cleanup algorithms for each individual algorithm, based on the properties of the possible outputs of each algorithm, which may lead to further performance benefits.

Proof. Our proof follows a similar strategy as our proof for `up`, shown in our previous report [23].

The delay of P is defined as $P^{\uparrow} = \{\mathbf{v} + d \mid \mathbf{v} \in P, d \in \mathbb{R}_{\geq 0}\}$, i.e. 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, and $\widehat{P}$ be the polyhedra obtained from running `up-alt`(P) be $\widehat{P} = \text{co}(V) \oplus \text{cone}(V' \cup W)$, where $V' = \{\mathbf{x} \mid \mathbf{x} \neq \bigoplus_{i=1}^{p} \gamma_i \mathbf{v}^i \oplus \bigoplus_{i=1}^{q} \beta_i \mathbf{w}^i\}$. We have to prove that $\widehat{P} = P^{\uparrow}$. This is the case if $P^{\uparrow} \subseteq \widehat{P}$ and $\widehat{P} \subseteq P^{\uparrow}$.

- $P^{\uparrow} \subseteq \widehat{P}$: Let $\mathbf{p} \in P^{\uparrow}$. From the definition of the 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 and 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 $\alpha_i d\mathbf{v}^i = \alpha_i d\mathbf{v}^i \oplus \alpha_i \mathbf{v}^i$ for all $1 \leq i \leq p$. So, $\bigoplus_{i=1}^{p} \alpha_i \mathbf{v}^i$ can be added to the equality without changing its value to get:

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

This means that $\mathbf{q} + d \in \text{co}(V) \oplus \text{cone}(V \cup W)$. We know $V' \subseteq V$ is the set of generators from V that cannot be generated by the linear combination of the remaining generators of V and W. Trivially, the separation of V into V' and $V \setminus V'$ gives us two disjoint sets. Therefore the expression representing the linearly scaled V, $\bigoplus_{i=1}^{p} \alpha_i d\mathbf{v}^i$ can be written as:

$$\bigoplus_{i=1}^{p} \alpha_i d\mathbf{v}^i = \bigoplus_{i=1}^{g} \alpha_i d\mathbf{v}'^i \oplus \bigoplus_{i=g+1}^{p} \alpha_i d\mathbf{v}''^i,$$

where $\mathbf{v}'^i \in V', \mathbf{v}''^i \in V \setminus V'$. This gives us:

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

We know from the definition of V' that:

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

since $\mathbf{v}'^i \in V \setminus V'$ are the generators that could be represented by the others.

Ultimately this gives us:

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

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

- $\widehat{P} \subseteq P^{\uparrow}$: It can easily be seen that at the end of `up-alt`, $V_{out} = V_{in}$ and $W_{out} \subseteq W_{in} \cup V_{in}$, where V_{in}, W_{in} is the input, and V_{out}, W_{out} is the output. From our previous report [23], we know that our original `up` algorithm results in $\widehat{P}_{old} = \text{co}(V) \oplus \text{cone}(W \cup V) = P^{\uparrow}$. This trivially shows that $\widehat{P} \subseteq \widehat{P}_{old}$, which implies $\widehat{P} \subseteq P^{\uparrow}$.

□

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 14: `down`(P)

```
V := V ∪ {(−1, ..., −1)}
for i := 1 to n do
  and(P, x_i ≥ 0)
end for
```

The algorithm works as a result of the properties of Minkowski sums of max-plus generators; by using -1, we essentially draw a ray going backwards from each individual generator in the original polyhedron, and this can be shown to match the definition of backwards delay, $P \downarrow = \{p \mid p + d \in P, d \in \mathbb{R}_{\geq 0}\}$. For details of the proof, refer to our previous project report [23].

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. This is done by iterating through all generators, setting the given dimension to the given reset value for the convex generators, and to $-\infty$ for the linear generators [17]. 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 15: `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

Intuitively, the algorithm works by forcing all the convex generators to be the desired value for the specificed dimension, and eliminating the effect of the linear generators on this dimension; consequently, the only possible value for that dimension is the desired value. For the full proof, see our previous project report [23].

3.3.5 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. Since linear generators may be scaled with an arbitrary value, this generator allows clock x_i to be any value. The proof of correctness is also described in our previous project report [23]. 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 16: `free`$(P, x_i = 0)$

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

3.3.6 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 [17] 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 takes the union of both sets of generators from each polyhedron, which can be done in $O(pn)$. For DBMs, the overapproximation algorithm is

$O(n^2)$ as it simply involves searching through the matrices and picking the higher value from them.

Algorithm 17: `convex-union`(P_1, P_2)

$P = co(V_1 \cup V_2) \oplus cone(W_1 \cup W_2)$

3.4 Termination of reachability

Because the clock space is infinite, the reachability algorithm cannot be guaranteed to terminate. To avoid this issue, DBMs use extrapolation as stated in Section 2.1.1 on page 7. However, due to the substantial differences in representation, the DBM algorithm cannot be applied directly to max-plus polyhedra.

Although we were able to show a number of principles which can be applied in an extrapolation algorithm for max-plus polyhedra (see Chapter 4 on page 31 for details), we were not able to construct a full algorithm which could accomplish this. Consequently, we cannot currently guarantee termination unless models are modified to ensure a finite state space, e.g. by not having infinite paths in the system.

3.5 Cleaning up

All of our transformations on max-plus polyhedra 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 redundancy and only store the minimal number of generators.

3.5.1 Removing redundant generators – `cleanup`(P)

Cleaning up results in a combination of a unique and minimal representation for max-plus polyhedra, in the sense that the convex part is minimal and unique, while the linear part is only minimal. 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 18: `cleanup`(P)

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

This is done by checking each point to see whether it can be generated by the other points; if so, the point is removed. Every such check is $O(pn)$, which makes the time complexity of this algorithm $O(p^2 n)$ [19, 2, 15].

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

	Max-plus polyhedra	DBM
Emptiness test	$O(1)$	$O(1)^*$
Inclusion test	$O(p^2n)$	$O(n^2)$
Constraint satisfaction	$O(p)$	$O(1)$
Constraint intersection	$O(p^2n)$	$O(n^2)$
Delay	$O(pn)$	$O(n)$
Backward delay	$O(p^2n^2)$	$O(n^2)$
Resetting clocks	$O(p)$	$O(n)$
Removing constraints	$O(p+n)$	$O(n)$
Union overapproximation	$O(pn)$	$O(n^2)$
Removing redundant generators	$O(p^2n)$	$O(n^3)^\dagger$

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

DBM is for DBM algorithms. Therefore, it can be compared to the canonicalization algorithm for DBMs, which is the standard Floyd-Warshall algorithm and therefore $O(n^3)$.

3.6 Summary

Table 3.1 shows a comparison of the complexity of the algorithms for max-plus polyhedra and for DBMs. The number of clocks is denoted n, while p denotes the number of generators of the polyhedron. We can conclude from this comparison that the complexities of the polyhedra-based algorithms are comparable to the DBM algorithms, though with a slight advantage to DBMs.

The main argument for using max-plus polyhedra is therefore the more accurate overapproximations; max-plus polyhedra make it possible to represent more complex shapes than a DBM, which can provide a significant advantage for some models.

Table 3.1 notes

* This requires the zone to always be canonical and for every operation changing a bound to verify if the upper bound is lower than the corresponding lower bound, setting D_{00} to a negative value if this is the case.

† As mentioned in Section 3.5.1 on the preceding page, we have decided to compare this to DBM canonicalization. Note that some of the DBM algorithms accommodate the possibility of a slight alteration which preserves canonicity, allowing the canonicalization to be skipped.

CHAPTER 4

PRINCIPLES FOR EXTRAPOLATION

Extrapolation is the most commonly used way to ensure termination when representing the clock space as a DBMs [9, 11]. This, as described in Section 2.1.1 on page 7, takes advantage of the fact that when maximal bounds are exceeded, the clock valuations become region equivalent, thereby allowing the zones to include entire regions that are region equivalent to points already in the zone.

Although we do not have a complete extrapolation algorithm, we have been able to determine a number of principles which could be used as part of such an algorithm for max-plus polyhedra.

To prove the validity of each principle, we must prove that the modification inferred from the principle is sound. In other words, the modification of a polyhedron P, denoted P^{mod}, must be a subset of the region closure of P, called $P^{closure}$, as given in Definition 2.2 on page 6. This subset, $P^{mod} \subseteq P^{closure}$ means that for all $\mathbf{p}' \in P^{mod}$, there exists $\mathbf{p} \in P$ s.t. $\mathbf{p}' \sim \mathbf{p}$.

Furthermore, if we want to prove a complete extrapolation algorithm, we would also need to prove that the modification performed on each of the infinite number of possible inputs, leads to termination of the reachability algorithm, which can be done by either proving a finite set of outputs, or by proving the outputs to be well quasi-ordered (WQO).

All lemmas and theorems in this chapter are only guaranteed to hold for timed automata with no difference constraints. Per definition, such constraints cannot exist in a Petri net, therefore the principles hold for any such model.

Definition 4.1 (Well-Quasi Order)**.** A binary relation, $\preceq$, on the elements of a set Σ is a *well quasi-order* if and only if $\preceq$ is a preorder on Σ, and for any infinite sequence $s_1, s_2, s_3, \ldots$ of elements of Σ, there exist $i < j$ such that $s_i \preceq s_j$.

Lemma 4.2. *The containment relation, $\supseteq$, between two polyhedra is a preorder, i.e. it is reflexive and transitive.*

Proof. Since the containment between polyhedra defined in Section 3.2.3 on page 21 is a logical extension of set containment, it follows trivially that $\supseteq$ is a preorder over the set of all max-plus polyhedra. □

4.1 1-clock model

Let an arbitrary one-dimensional polyhedron $P = \text{co}(V) \oplus \text{cone}(W)$ and a maximal constant k.

Lemma 4.3. *There exist only finitely many 1-dimensional polyhedra P where there does not exist a $\mathbf{v} \in V$ where $\mathbf{v}_1 > k$.*

Proof. This is trivial as there are only finitely many integer values between k and 0. □

Lemma 4.4. *For a one-clock model, given $P = co(V) \oplus cone(W)$ where there exists a $\mathbf{v} \in V$ s.t. $\mathbf{v}_1 > k$, then $P^{mod} = co(V) \oplus cone(W \cup \{(0)\})$ leads to $P^{mod} \subseteq P^{closure}$.*

Proof. Let P be a polyhedron satisfying the condition, then $P^{mod} = \text{co}(V) \oplus \text{cone}(W \cup \{\mathbf{0}\})$ where $\mathbf{0}$ is the generator (0). This gives us:

$$\mathbf{p} = \bigoplus_{m=1}^{p} \alpha^m \mathbf{v}^m \oplus \bigoplus_{m=1}^{q} \beta^m \mathbf{w}^m$$

$$\mathbf{p}' = \bigoplus_{m=1}^{p} \alpha^m \mathbf{v}^m \oplus \bigoplus_{m=1}^{q} \beta^m \mathbf{w}^m \oplus \gamma \mathbf{0}$$

where $\mathbf{p} \in P$, $\mathbf{p}' \in P^{mod}$ and $\gamma \in \mathbb{R}_{max}$. We know there exists a $\mathbf{p}^* \in P$ s.t. $\mathbf{p}^*_1 > k$, since there exists a $\mathbf{u} \in V$ s.t. $\mathbf{u}_1 > k$. For any valid assignment of α, β, and γ scalars for a $\mathbf{p}'$, it is necessarily the case that $\bigoplus_{m=1}^{p} \alpha^m \mathbf{v}_1^m \oplus \bigoplus_{m=1}^{q} \beta^m \mathbf{w}_1^m \leq \bigoplus_{m=1}^{p} \alpha^m \mathbf{v}_1^m \oplus \bigoplus_{m=1}^{q} \beta^m \mathbf{w}_1^m \oplus \gamma \mathbf{0}_1$. If they are equal, $\gamma\mathbf{0}$ has no effect, and we can trivially reuse the same α and β scalars to create a $\mathbf{p} \in P$ s.t. $\mathbf{p} = \mathbf{p}' \implies \mathbf{p} \sim \mathbf{p}'$.

If $\gamma\mathbf{0}$ has an effect on the result, then it is either the case that $\mathbf{p}'_1 > k$, in which case $\mathbf{p}' \sim \mathbf{p}^*$, or that $\mathbf{p}'_1 \leq k$, in which case $\mathbf{p}' \in P$, since the presence of $\mathbf{u}$ guarantees that if such a point exists, there must also exist a way to obtain that point entirely using V, as V imposes a lower bound which $\gamma\mathbf{0}$ cannot violate. Thus we have shown that for all $\mathbf{p}'$ there exists a $\mathbf{p}$ where $\mathbf{p}' \sim \mathbf{p}$, which implies $P^{mod} \subseteq P^{closure}$. □

Lemma 4.5. *The set of polyhedra Σ^{ex} being $\{P \mid \nexists \mathbf{v} \in V.\mathbf{v} > k\} \cup \{P^{mod} = co(V) \oplus cone(W \cup \{(0)\}) \mid \exists \mathbf{v} \in V.\mathbf{v} > k\}$ with the containment relation, $\supseteq$, between polyhedra, $(\Sigma^{ex}, \supseteq)$ is a WQO.*

Proof. From Lemma 4.2 on the preceding page we know the superset relation between polyhedra is a preorder. The rest of the proof is simple conversion from the fact that $(\mathbb{N}, \leq)$ is a WQO. First we see that this well quasi-order can be interpreted as the set of intervals $([\mathbb{N}, \infty[)$ is a WQO over $\supseteq$. Finally we see that Σ^{ex} correspond to the intervals $([x, \infty[)$, $x \in \mathbb{N}_{>k}$ and the intervals $([y, z])$ where $0 \leq y \leq z \leq k < \infty$. It is trivially seen that the intervals between y, z are a finite set and $([\mathbb{N}_{>k}, \infty[) \subset ([\mathbb{N}, \infty[)$. We know a subset of a WQO is itself a WQO [25] and trivially, the union of a well quasi-ordered set and a finite set over the same preordered relation, is a well quasi-order as well; thus $(\Sigma^{ex}, \supseteq)$ is a WQO. □

Theorem 4.6. *For 1-clock models, $P^{extra} = co(V) \oplus cone(W \cup \{(0)\})$ if there exists a $\mathbf{v} \in V$ s.t.$\mathbf{v}_1 > k$, otherwise $P^{extra} = P$, P^{extra} is an extrapolation of P.*

Proof. From Lemma 4.5, we know that our given definition of P^{extra} is a WQO. Lemma 4.4 shows that the modified polyhedra are sound with rspect to region closure. The unmodified polyhedra are trivially sound. Since the two cases for this algorithm are mutually exclusive, combining them must retain the soundness. Finally, since the condition for one algorithm is the negation of the other, we trivially know P^{extra} covers the entire set of possible input polyhedra. □

4.2 n-clock models

For models with more than one clock, we have developed a number of sound subroutines which may be used as part of a complete extrapolation algorithm.

Lemma 4.7. *Given maximal constants $k_1, \ldots, k_n$ for clocks $x_1, \ldots, x_n$ there exists only finitely many different polyhedra P satisfying $\nexists \mathbf{p} \in P \forall 1 \leq (i,j) \leq n, i \neq j . \mathbf{p}_i - \mathbf{p}_j > k_i \vee (\mathbf{p}_i > k_i \wedge \mathbf{p}_j > k_j)$.*

Proof. The lemma states that below maximal bounds and between diagonals going out from the bottom of each maximal bound there is only finitely many possible polyhedra. This can be seen as a trivial extension of the fact that only finitely many positive integer points exist which satisfy the condition, and since all generators in a polyhedron represent positive integer points, there can only be finitely many distinct polyhedra satisfying the condition. In Figure 4.1 the highlighted clock space is showing the part defined by Lemma 4.7 for two clocks, with maximal bounds k_x and k_y. □

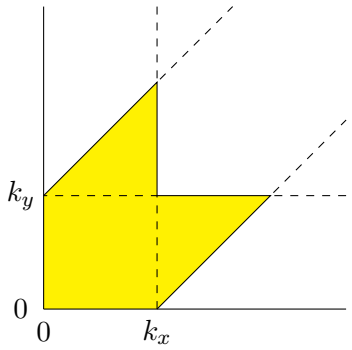


Figure 4.1: The two-dimensional finite part clock space below maximal bounds and their diagonals as stated by Lemma 4.7.

Lemma 4.8. *For all polyhedra P where there exists a $\mathbf{v} \in V$ s.t. $\mathbf{v}_i - \mathbf{v}_{j \neq i} > k_i)$, or a $\mathbf{w} \in W$ s.t. $\mathbf{w}_i - \mathbf{w}_{j \neq i} > k_i)$, then for all $\mathbf{p} \in P$, there exists a $\mathbf{p}^{\#} \in P$ such that $\mathbf{p}^{\#}_{j \neq i} = \mathbf{p}_j$ and $\mathbf{p}^{\#}_i > k_i$.*

Proof. Let the $\mathbf{v}$ or $\mathbf{w}$ which satisfied the initial condition be denoted by $\mathbf{f}$ and the corresponding scalar by ω. Then by choosing $\omega = -\max_{l \neq i}(\mathbf{f}_l) - s$, where $0 < s < 1$, we get:

$$\omega \mathbf{f}_i = -\max_{l \neq i}(\mathbf{f}_l) - s + \mathbf{f}_i$$

$$\omega \mathbf{f}_{j \neq i} = -\max_{l \neq i}(\mathbf{f}_l) - s + \mathbf{f}_j$$

We know from the initial condition that $\mathbf{f}_i - \mathbf{f}_{j \neq i} > k_i$ and since all generators have integer or $-\infty$ elements, $\mathbf{f}_i - \mathbf{f}_{j \neq i} - 1 \geq k_i$. Trivially, $\mathbf{f}_{j \neq i} \leq \max_{l \neq i}(\mathbf{f}_l)$, so $\omega \mathbf{f}$ can be generalised to:

$$\omega \mathbf{f}_i > k_i$$

$$\omega \mathbf{f}_{j \neq i} < 0$$

As $\omega < 0$ for all s, we know it is a legal scalar value in both cases that $\mathbf{f} \in V$ and $\mathbf{f} \in W$, except for the case where $\mathbf{f}$ represents the only convex generator. In this case $V = \{\mathbf{f}\}$ and thus we

know that $0 \otimes \mathbf{f}_i \oplus \text{cone}(W) \implies \mathbf{p}_i > k_i$ for all $\mathbf{p} \in P$. If $\mathbf{f}$ is not the only convex generator, or if it is a linear generator, we define $P^* = P \setminus \{\mathbf{f}\}$ and $\mathbf{p}^* \in P^*$, such that $\mathbf{p} = \mathbf{p}^* \oplus \omega\mathbf{f}$. Now we see that

$$\mathbf{p} = \mathbf{p}^* \oplus \omega\mathbf{f} \implies \begin{cases} \mathbf{p}_i > k_i & \text{as } \omega\mathbf{f}_i > k_i \implies \max(\mathbf{p}_i^*, \omega\mathbf{f}_i) > k_i \\ \mathbf{p}_{j\neq i} = \mathbf{p}_j^* & \text{as } \omega\mathbf{f}_j < 0 \implies \max(\mathbf{p}_j^*, \omega\mathbf{f}_j) = \mathbf{p}_j^* \end{cases} \tag{4.1}$$

Now we consider the case where $\omega < -\max_{l\neq i}(\mathbf{f}_l) - s$. Here we see that $\omega\mathbf{f}$ does not add anything to P in dimensions other than the i'th.

$$\mathbf{p}_{j\neq i}^* \oplus \omega\mathbf{f}_j = \mathbf{p}_j^*$$

This is fine as we have covered all $\mathbf{p}_j^*$ in Equation 4.1.

For the remaining case, where $\omega > -\max_{l\neq i}(\mathbf{f}_l) - s$, we see that regardless of what clock valuations $\omega\mathbf{f}$ adds for the rest of the dimensions, it will always be the case that

$$\mathbf{p} = \mathbf{p}^* \oplus \omega\mathbf{f} \implies \mathbf{p}_i > k_i$$

□

Theorem 4.9. *For all polyhedra P where there exists a $\mathbf{v} \in V$ s.t. $\mathbf{v}_i - \mathbf{v}_{j\neq i} > k_i$, or a $\mathbf{w} \in W$ s.t. $\mathbf{w}_i - \mathbf{w}_{j\neq i} > k_i$), then $P^{mod} = co(V) \oplus cone(W \cup \{\mathbb{0}^{i\mathbb{1}}\})$, where $\mathbb{0}_i^{i\mathbb{1}} = 0$, $\mathbb{0}_{j\neq i}^{i\mathbb{1}} = -\infty$, then $P^{mod} \subseteq P^{closure}$.*

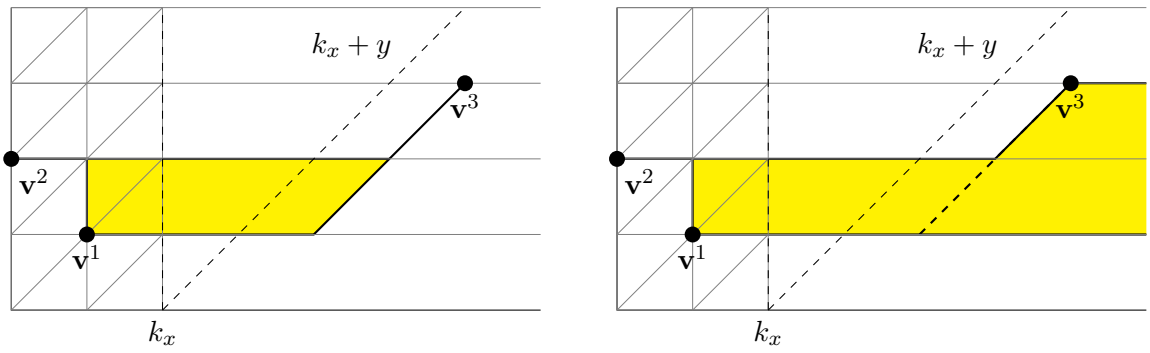


Figure 4.2: Theorem 4.9 applied to a polyhedron.

Proof. For a two-dimensional example of the modification, see Figure 4.2.

Let $\mathbf{p} \in P$ and $\mathbf{p}' \in P^{mod}$.

$$\mathbf{p} = \bigoplus_{m=1}^{p} \alpha^m \mathbf{v}^m \oplus \bigoplus_{m=1}^{q} \beta^m \mathbf{w}^m$$

$$\mathbf{p}' = \bigoplus_{m=1}^{p} \alpha^m \mathbf{v}^m \oplus \bigoplus_{m=1}^{q} \beta^m \mathbf{w}^m \oplus \gamma \mathbb{0}^{i\mathbb{1}}$$

where $\alpha, \beta, \gamma \in \mathbb{R}_{max}$, $m, p, q \in \mathbb{N}$ and $\bigoplus_{m=1}^{p} \alpha^m = 0$.

It is trivially seen that adding $\gamma\, \mathbb{0}^{i\mathbb{1}}$ to any point $\mathbf{p} \in P$, the resulting point $\mathbf{p}'$ must be identical to $\mathbf{p}$ except for the i'th dimension, where it must be greater than or equal to the value in $\mathbf{p}$. In combination with Lemma 4.8 on page 33, we can therefore safely discard the other dimensions, degenerating our polyhedron into one dimension. From here, we can directly apply the same proof as for 4.4 on page 32 to show $P^{mod} \subseteq P^{closure}$. □

Lemma 4.10. *For all polyhedra P where there exists a $1 \leq i \leq n$ s.t. $\mathbf{v}_i > k_i$ for all $\mathbf{v} \in V$, then for all $\mathbf{p} \in P$, $\mathbf{p}_i > k_i$.*

Proof. This is trivial as the convex generators impose a lower bound for all clocks, since $\bigoplus_{j=1}^{p} \alpha^j = 0$. □

Theorem 4.11. *For all polyhedra P where there exists a $1 \leq i \leq n$ s.t. for all $\mathbf{v} \in V$, $\mathbf{v}_i > k_i$, then $P^{mod} = P_{R[x_i=k_i+1]} \oplus \gamma \mathbb{0}^{i\mathbb{1}}$, where $\gamma \in \mathbb{R}_{max}$, $\mathbb{0}_i^{i\mathbb{1}} = 0$, $\mathbb{0}_{j\neq i}^{i\mathbb{1}} = -\infty$, and $P^{mod} \subseteq P^{closure}$.*

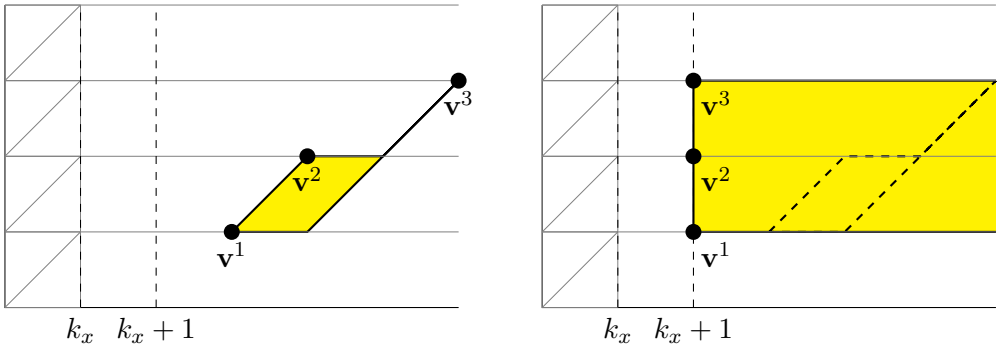


Figure 4.3: A polyhedron modified as described by Theorem 4.11.

Proof. Figure 4.3 shows an example of the modification done on a two-dimensional polyhedron.

Let $\mathbf{p} \in P$ and $\mathbf{p}' \in P^{mod}$. From the definition of reset given in Algorithm 15 on page 28, we know that for all $\mathbf{p}' \in P^{mod}$ there exists a $\mathbf{p} \in P$ s.t. $\mathbf{p}'_i > k_i$ and $\mathbf{p}'_{j\neq i} = \mathbf{p}_j$, as we have reset the i'th dimension to $k_i + 1$, and $\gamma \mathbb{0}^{i\mathbb{1}}$ neither can add values lower than the lower bound imposed by the convex generators nor add anything in any dimensions other than the i'th. This satisfies region closure from Definition 2.2 on page 6 for all dimensions $j \neq i$.

For the i'th dimension, we can use the knowledge from Lemma 4.10 that for all $\mathbf{p} \in P$, $\mathbf{p}_i > k_i$. Thus we know that for all combinations of legal values in dimensions other than the i'th, there exists a $\mathbf{p} \in P$ where $\mathbf{p}_i > k_i$. This can be combined with the fact that all dimensions apart from the i'th have been preserved to see that for all $\mathbf{p}' \in P^{mod}$ there exists a $\mathbf{p} \in P$ s.t. $\mathbf{p}'_i > k_i \wedge \mathbf{p} > k_i$. All dimensions are therefore covered, so for all $\mathbf{p}' \in P^{mod}$ there exists a $\mathbf{p} \in P$ s.t. $\mathbf{p} \sim \mathbf{p}' \implies P^{mod} \subseteq P^{closure}$. □

Corollary 4.12. *All polyhedra P, where $\mathbf{v}_i > k_i$ for all $1 \leq i \leq n$ and $\mathbf{v} \in V$, can be extrapolated to $P^{extra} = co(\{(k_1 + 1, \ldots, k_n + 1)\}) \oplus cone(\{\mathbb{0}^{1\mathbb{1}}, \ldots, \mathbb{0}^{n\mathbb{1}}\})$.*

Theorem 4.13. *For all polyhedra P where there exists a $\mathbf{v} \in V$ s.t. $\mathbf{v}_{1\leq i\leq n} > k_i$, then $P^{mod} = co(V) \oplus cone(W \cup \{\mathbf{v}\})$ and $P^{mod} \subseteq P^{closure}$.*

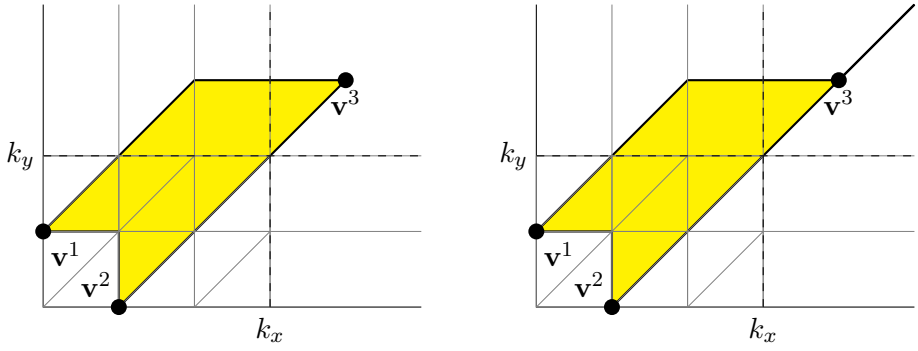


Figure 4.4: A polyhedron modified by the property of Theorem 4.13

.

Proof. In Figure 4.4 a two-dimensional example of the modification is shown.

Let the convex generator satisfying the condition be denoted $\mathbf{u}$ and $\mathbf{p}' \in P^{mod}$:

$$\mathbf{p}' = \bigoplus_{j=1}^{p} \alpha^j \mathbf{v}^j \oplus \bigoplus_{j=1}^{q} \beta^j \mathbf{w}^j \oplus \gamma\mathbf{u}$$

where $\alpha, \beta, \gamma \in \mathbb{R}_{max}$ and $\bigoplus_{j=1}^{p} \alpha^j = 0$.

For any $\gamma \leq \alpha_u$, where α_u is the convex scalar used for $\mathbf{u}$, $\gamma\mathbf{u} \oplus \alpha\mathbf{u} = \alpha\mathbf{u}$; in other words, $\gamma\mathbf{u}$ is overridden by $\alpha\mathbf{u}$.

For any $\gamma \leq 0, \alpha_u < \gamma$, $\gamma\mathbf{u}$ overrides $\alpha_u\mathbf{u}$, but since γ is a valid value for α_u, we could have used the value for γ as the value for α_u, so $\gamma\mathbf{u}$ doesn't add any new points.

For any $\gamma > 0$, $\gamma\mathbf{u}$ enforces $\mathbf{p}'_i > k_i$, since for all $1 \leq i \leq n$, $\gamma\mathbf{u} > k_i \implies \gamma\mathbf{u}_i \oplus \mathbf{x}_i > k_i$, where $\mathbf{x} \in \mathbb{R}^n_{max}$; thus any such point is region equivalent with $\mathbf{u}$.

Since this works on arbitrary polyhedra, then intuitively, the principle can be applied to multiple generators satisfying the condition. □

Claim. *Theorems 4.9, 4.11 and 4.13 are sufficient to perform extrapolation on 2-dimensional polyhedra.*

Proof sketch. Lemma 4.7 on page 33 states that the number of possible polyhedra below both of the k-diagonals or below both k_x and k_y is finite. This is exactly the part of the clock space highlighted in Figure 4.1 on page 33.

We believe that in two dimensions, the resetting modification described in Theorem 4.11 on the preceding page results in a finite set for polyhedra entirely below the opposite maximal bound; i.e. the number of polyhedra entirely below k_y and entirely above k_x is finite and vice versa. That still leaves the polyhedra that are partly above either of the k-diagonals and below the respective k's, and the polyhedra both above and below k_x and k_y. The latter modification is described in Theorem 4.13 on the previous page, while the former is described in Theorem 4.9 on page 34.

We believe the union of these two sets of modified polyhedra to be well quasi-ordered, since given a polyhedron partly above the k_x-diagonal and thus modified to extend infinitely for the x-clock, there are only finitely many supersets of this polyhedron, given we maintain the possible values for y. Conversely, if we do not maintain the value of y, there are only finitely many values y can be until either $y = 0$ or $y - x > k_y$, which will cause a modification by extending infinitely

in the direction of the y-clock. There is then only the possibility of modifying both x and y. In this case, there are only finitely many steps back to $(0, 0)$ or until $x > k_x$ and $y > k_y$, in which case the polyhedron may be extended infinitely on the diagonal. A similar strategy could have been used if the initial polyhedron was modified either on the y-clock or on the diagonal.

Since the lemma and the three theorems cover all possible input polyhedra in two dimensions, and since the union of finite and well quasi-ordered sets over the same preorder is still WQO, we can make use of these to ensure termination. Additionally, all of the theorems have been proven sound on arbitrary polyhedra, hence the combination of these is still sound, allowing extrapolation of 2-dimensional polyhedra. □

4.3 Summary

For models containing only one clock, we have shown that extrapolation can be done using the provided theorems. Additionally, we have sketched a proof strategy for our claim that the theorems are also sufficient for two-dimensional polyhedra.

The reason this is not sufficient for models where $n > 2$ is that Theorem 4.13 on page 35 only allows modification when all maximal bounds are exceeded. Thus there exists infinite sequences given by modifying two clocks, where the polyhedra are both below k_i and above k_i-diagonals. Therefore, the presented modification strategies are not sufficient for complete extrapolation of arbitrary polyhedra.

CHAPTER 5

IMPLEMENTATION

In order to test the efficiency of our approach, we have implemented it in VerifyTAPN [22], a project created to perform reachability checking directly on timed-arc Petri nets. This chapter will only discuss the changes made as part of our implementation; for a more detailed explanation of VerifyTAPN itself, refer to Jacobsen and Jacobsen [21].

5.1 Changes to VerifyTAPN

In addition to implementing max-plus polyhedra, we have added support for overapproximation to VerifyTAPN.

In a timed automaton, overapproximation is implemented by only storing a single zone for each location, and using convex hull union when adding more zones. This helps to avoid state-space explosion, but at the cost of losing accuracy: even if a point is found to be inside a zone when using overapproximation, that point may not actually be reachable. For a timed-arc Petri net, the same principle applies, such that one zone is stored for each combination of token placements.

Overapproximation allows us to find a case where we can exploit the properties of our convex union, and get a correct result in a situation where DBMs would provide an inconclusive result. In such a case, we can compare the running time of max-plus overapproximation with the running time of both overapproximation and exact analysis for DBMs.

5.2 Max-plus polyhedra

VerifyTAPN includes support for zone-based model checking, using the Uppaal DBM library [31]. The implementation supports a dynamic number of tokens by adding or removing clocks as necessary, and also supports symmetry reduction.

Our implementation of max-plus polyhedra also supports a dynamic number of tokens in a similar way to DBMs. Symmetry reduction is not currently implemented, so all testing on DBMs is performed without this option; however, it would certainly be possible to implement symmetry reduction for max-plus polyhedra as well, which should provide similar results as adding symmetry reduction for DBMs.

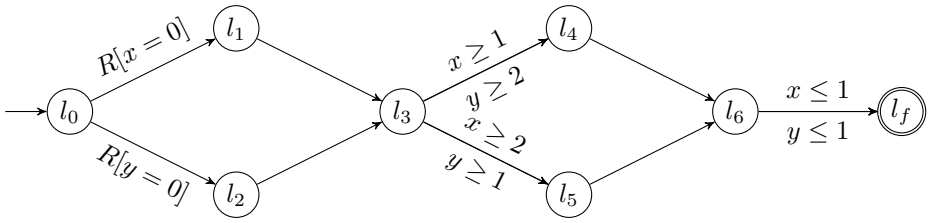


Figure 5.1: Timed automaton used as template for benchmarking.

To represent our generators, we have implemented a simple `MPVector` class which can handle the various operations we need to perform on the generators - other than max-plus addition and multiplication, this includes adding and removing dimensions, and comparing two instances of the class. To keep the code as simple as possible, the underlying storage is currently implemented using `std::vector<int>`, but can be substituted with e.g. a manually allocated array.

The two sets of generators in our max-plus polyhedra are represented using two doubly-linked lists (`std::list<MPVector>`). Although it may seem counter-intuitive to store sets using lists, it is much more difficult to alter elements in a set, since practical implementations of sets (e.g. `std::set`) require the elements to be ordered, necessitating that elements are removed, modified and subsequently re-added. Since we perform the `cleanup` operation after most other operations, these extra points will be removed quickly, so we consider this to be a reasonable trade-off. Another disadvantage of lists is that finding a specific element requires searching through the entire list - but this is not a problem in practice, since none of our algorithms look for specific points; when dealing with individual generators, we always want to work on all of them, so we have to iterate over the entire collection.

5.3 Benchmarking

The main advantage of using max-plus polyhedra is the possibility for more accurate overapproximations, so we have created a model specifically to highlight this difference. The basic model is shown in Figure 5.1, and it uses the same principle as the model in Section 2.3.2 on page 16.

If overapproximation with DBMs are used, the final transition from l_6 to l_f will be enabled, even though this is clearly not possible in practice – hence, the result will be inconclusive. In this case, max-plus polyhedra avoid this issue, since they are able to generate the exact result even with overapproximation.

Using a tool, we construct different models from this template, based on three parameters:

Number of branches Between l_3 and l_6, we can vary the number of branches n. Each branch goes to a different, intermediate state, with the i'th branch constraining on $x \geq i$ and $y \geq (n - i + 1)$.

Number of repetitions Since the exact automaton shown in Figure 5.1 is too small to show any significant differences in timing, we chain the template after itself a number of times. When doing so, the transition between l_6 and l_f is replaced with a transition from l_6 to l_0 in the next instance of the template, resetting both clocks involved with the upcoming instance.

Number of clocks We can use an arbitrary number of clocks when repeating our template. Essentially, the set of clocks C is put into a queue; every time the template is instantiated for a repetition, two clocks are removed from the queue and used in place of x and y. After generating the current instance, the clocks are put back in the order they were removed, and the next instance can be generated. For example, if we use 4 clocks (1-4), the first instance would use clocks 1 and 2, while the second instance would use 3 and 4, and the nth instance would use the same clocks as the $(n-2)$th instance.

The tool works by constructing the timed automaton, then converting it to a timed-arc Petri net using an algorithm presented by Srba [27] and ultimately serializing this Petri net as an XML model.

As stated in Section 3.4 on page 29, termination is guaranteed due to a finite number of paths through the system. Consequently, we do not add upper invariants on the age of tokens.

5.3.1 Results

Table 5.1 shows the absolute time required to perform reachability checking with breadth-first search on our model, using a variety of different parameter values. *MPP* refers to the time spent performing model checking with max-plus polyhedra and overapproximation (which is exact in this case), whereas *DBM* refers to the time spent performing using DBMs and exact analysis. To reduce the impact of unfortunate scheduling, each model has been checked 3 times, and the best result selected.

Since overapproximations with DBMs will result in incorrect results, this configuration is not included; similarly, because all of our max-plus algorithms are of equal or higher complexity than the corresponding DBM algorithms, DBMs will always outperform max-plus polyhedra when both are faced with exact analysis, so we also avoid exact analysis with max-plus polyhedra.

The computer used for these benchmarks is an Intel Core i7-920 @ 2.67 GHz, with 6 GB RAM and Windows 7 (64-bit). GCC 4.3.3 (32-bit) was used to compile the program using -O3, and DBM operations are performed using the Uppaal DBM library version 2.07 built in release mode.

Branches		2		3		5		10	
Reps	Clocks	DBM	MPP	DBM	MPP	DBM	MPP	DBM	MPP
100	2	0.26s	0.22s	0.44s	0.31s	0.85s	0.55s	2.35s	1.44s
200	2	1.03s	0.75s	1.78s	1.08s	3.44s	1.88s	9.78s	4.97s
400	2	4.61s	3.06s	7.74s	4.37s	15.0s	7.66s	42.6s	19.8s
100	3	0.28s	0.26s	0.47s	0.37s	0.89s	0.64s	2.44s	1.64s
200	3	1.09s	0.85s	1.85s	1.23s	3.54s	2.11s	9.81s	5.40s
400	3	4.36s	3.06s	7.26s	4.35s	14.1s	7.53s	40.0s	19.2s
100	4	0.44s	0.60s	1.27s	1.63s	3.06s	8.63s	9.88s	144s
200	4	1.72s	1.62s	5.06s	3.82s	12.5s	18.8s	41.2s	311s
400	4	7.14s	5.06s	21.3s	10.7s	51.4s	43.7s	174s	630s
100	5	0.45s	0.93s	1.84s	2.46s	4.41s	12.2s	14.3s	185s
200	5	1.89s	2.28s	7.13s	5.47s	17.8s	25.3s	57.8s	367s
400	5	7.56s	6.21s	28.5s	13.4s	71.4s	54.4s	239s	734s

Table 5.1: Timing results for model checking

To more easily compare the results, Table 5.2 shows the time spent for max-plus polyhedra as a percentage of the time spent for DBMs. The number of states discovered and explored are shown, respectively, in Table 5.3 and Table 5.4 on the next page.

Branches		2	3	5	10
Reps	Clocks				
100	2	83.3%	70.3%	64.6%	61.2%
200	2	72.7%	60.7%	54.5%	50.8%
400	2	66.4%	56.6%	51.0%	46.6%
100	3	93.8%	79.6%	71.8%	67.0%
200	3	78.6%	66.4%	59.5%	55.1%
400	3	70.1%	59.9%	53.6%	48.0%
100	4	137%	128%	282%	1453%
200	4	94.0%	75.4%	151%	753%
400	4	70.9%	50.1%	85.0%	361%
100	5	205%	133%	275%	1291%
200	5	120%	76.8%	142%	634%
400	5	82.2%	47.0%	76.2%	308%

Table 5.2: Time spent on max-plus as percentage of DBM time

Branches		2		3		5		10	
Reps	Clocks	DBM	MPP	DBM	MPP	DBM	MPP	DBM	MPP
100	2	1399	900	1997	1100	2995	1500	5391	2500
200	2	2799	1800	3997	2200	5995	3000	10791	5000
400	2	5599	3600	7997	4400	11995	6000	21591	10000
100	3	1399	900	1997	1100	2995	1500	5391	2500
200	3	2799	1800	3997	2200	5995	3000	10791	5000
400	3	5599	3600	7997	4400	11995	6000	21591	10000
100	4	2191	900	5755	1100	11897	1500	31511	2500
200	4	4391	1800	11555	2200	23897	3000	63311	5000
400	4	8791	3600	23155	4400	47897	6000	126911	10000
100	5	2191	900	7715	1100	17385	1500	50327	2500
200	5	4391	1800	15515	2200	34985	3000	101327	5000
400	5	8791	3600	31155	4400	70185	6000	203027	10000

Table 5.3: Number of discovered states during model checking

Branches		2		3		5		10	
Reps	Clocks	DBM	MPP	DBM	MPP	DBM	MPP	DBM	MPP
100	2	1100	700	1500	800	2100	1000	3500	1500
200	2	2200	1400	3000	1600	4200	2000	7000	3000
400	2	4400	2800	6000	3200	8400	4000	14000	6000
100	3	1100	700	1500	800	2100	1000	3500	1500
200	3	2200	1400	3000	1600	4200	2000	7000	3000
400	3	4400	2800	6000	3200	8400	4000	14000	6000
100	4	1595	700	3777	800	6555	1000	12707	1500
200	4	3195	1400	7577	1600	13155	2000	25707	3000
400	4	6395	2800	15177	3200	26355	4000	51107	6000
100	5	1595	700	4757	800	8515	1000	16627	1500
200	5	3195	1400	9557	1600	17115	2000	33427	3000
400	5	6395	2800	19157	3200	34315	4000	67027	6000

Table 5.4: Number of explored states during model checking

As the numbers show, increasing the number of repetitions of our model always improves the relative performance of using max-plus polyhedra, but the results are more varied for branches and clocks. This makes sense: the model is designed to incur some degree of state space explosion for DBMs, so significantly more operations need to be performed than for overapproximating max-plus, negating the impact of the higher complexity of the algorithms.

It is notable that in the test data given above, models with 3 branches always require less relative time to analyze with max-plus polyhedra than models with 2 branches, but once we exceed 3 clocks, the same does not apply for 5 and 10 branches. This can be explained by the necessity to perform cleanup after each union; more clocks means this operation takes more time for every execution, and more branches means the operation must be executed more often. This implies the existence of a limit on the number of branches, after which max-plus polyhedra are unable to perform better than DBMs, determined by the number of clocks and repetitions; alternatively, the number of branches and repetitions can be said to limit the number of clocks we can support while maintaining a performance improvement.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

The main objective of this project was to expand upon our previous work [17, 23] and implement our work in a manner that allows us to sensibly compare our performance against the UPPAAL DBM library. We succeeded in showing that models can be created such that a max-plus overaproximation can give the correct answer where DBM overaproximation would not, while simulatenously outperforming exact DBM analysis. Additionally, we have provided a set of theorems and lemmas which may be used in the attempt to create an extrapolation algorithm to ensure termination of the reachability algorithm.

The missing extrapolation and other 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.

6.1 Performance

Although we have shown a case where max-plus polyhedra can use overapproximation to obtain the correct result faster than DBMs, max-plus polyhedra are still slower than DBMs in the general case, with many algorithms having a higher complexity than the corresponding DBM algorithm, or incurring a high additional complexity due to the need to perform cleanup after performing the operation. It is possible that better algorithms exist which can reduce the total complexity.

We have considered the possibility of having the down algorithm alter the generators rather than relying on intersection. Several attemps came close, however none of them turned out to work for arbitrary polyhedra. Despite a lack of success, we believe an algorithm with better complexity than $O(p^2n^2)$ is possible but finding and proving such an algorithm is left as future work.

Additionally, the implementation can likely be improved to increase its practical performance, which would mitigate the effect of the higher complexities. For example, our current representation of vectors is quite naive, and e.g. by using a representation suitable for specialized CPU vector operations, a significant change may be observable.

6.2 Strict constraints

As we have shown, max-plus polyhedra can be used to represent timing information for real-time model checking as combinations of conjunctions, as well as some disjunctions, of constraints. If all constraints represented by a polyhedron are non-strict, no additional information for the polyhedron is needed. However, if strict constraints are introduced, the generators for the polyhedron cannot trivially be used to represent this strictness information. While we do not have a complete theory for this, we will present the current state of our work on this.

The basic idea comes from looking at the elements of constraints. Trivially, a strict constraint is equivalent to a non-strict constraint without the equality, i.e. $x-y < t \iff x-y \leq t \backslash x-y = t$. We attempt to transfer this principle to polyhedra by representing strict constraints as a set of polyhedra representing the strict faces, in addition to the usual non-strict polyhedron which defines the shape of the polyhedron. Formally, we write this as $P = \text{co}(V) \oplus \text{cone}(W) \setminus \{P_1 = \text{co}(V_1) \oplus \text{cone}(W_1), \ldots, P_s = \text{co}(V_s) \oplus \text{cone}(W_s)\}$, where $s \geq 0$ is the number of strict constraints, and each $P_i = \text{co}(V_i) \oplus \text{cone}(W_i)$, $1 \leq i \leq s$ represents one strict constraint. For convenience, we will denote $P = \bar{P} \setminus \tilde{P}$, where $\bar{P}$ is the non-strict max-plus polyhedron defining the shape, and $\tilde{P} = \{P_1, \ldots, P_s\}$ is the set of strict constraints as polyhedra.

Any polyhedron P with strictness information can be converted to a max-plus cone G in the regular manner by converting $\bar{P}$ and all $P_i \in \tilde{P}$ individually, using by Algorithm 3 on page 19; the same principles applies for converting back to a polyhedron. The notation of a cone carrying strictness will be the intuitive $G = \bar{G} \setminus \tilde{G}$.

We start by providing an algorithm for checking if one given point can be generated by a cone. This is done by first checking if the point can be generated by $\bar{G}$ using Algorithm 5 on page 21; if so, we check that it cannot be generated by any of cones in $\tilde{G}$.

Algorithm 19: `contains-point-strict`$(G, \mathbf{x})$

```
if contains-point(Ḡ, x) then
  for all G_i ∈ G̃ do
    if contains-point(G_i, x) then
      return false
    end if
  end for
  return true
end if
return false
```

The complexity of this algorithm is $O(spn)$ where s is the number of strict constraints represented as cones, p is the maximal number of generators in any of the cones in G and n is the number of clocks. Thus this is done in polynomial time.

The membership test can be used to create an algorithm for subset testing between two polyhedra. It uses Algorithm 6 on page 21 to take advantage of the fact that the non-strict polyhedron $\bar{P}$ must contain the non-strict polyhedron $\bar{P}'$ in order for $P \supseteq P'$ to be possible. If this is the case, we also need to ensure that the strict parts of the two polyhedra are handled correctly by stating that if a generator is strictly included in P' it must also be strictly included in P.

Algorithm 20: `contains-strict`(P, P')

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

The complexity of the containment algorithm is $O(sp^2n)$. This ensures that the relationship between polyhedra can be calculated using the given representation of strictness.

The first, and so far only, transforming algorithm we have devised for this data structure is constraint intersection. It assumes an input $\varphi = x_i - x_j \sim c, \sim \in \{\leq, <\}$ since all other constraint types can be rewritten to this form. The algorithm works by intersecting the non-strict part $\bar{P}$ with the non-strict form of the constraint, regardless of the actual $\sim$, using Algorithm 10 on page 24. If the result is empty, then P is empty regardless of all strictness information; if this initial intersection is non-empty, all constraints in $\tilde{P}$ is intersected with the non-strict form of φ, to keep them within the shape of our polyhedron, and the new constraint is added to $\tilde{P}$.

One thing to notice is that `and-strict` is an actual modifying algorithm, whereas `and` as presented in Algorithm 10 on page 24 ends up returning a modified copy instead.

Algorithm 21: `and-strict`$(P, x_i - x_j \sim c)$

$\bar{P} :=$ `and`$(\bar{P}, x_i - x_j \leq c)$
if $\bar{P} \neq \emptyset$ **then**
  for all $P_i \in \tilde{P}$ **do**
    $P_i :=$ `and`$(P_i, x_i - x_j \leq c)$
  end for
  if $\sim$ is $<$ **then**
    $P_{new} :=$ `and`$(\bar{P}, x_i - x_j = c)$
    if $P_{new} \neq \emptyset$ **then**
      $\tilde{P} := \tilde{P} \cup \{P_{new}\}$
    end if
  end if
end if

The non-strict constraint intersection has a complexity of $O(sp^2n)$ as it has to do regular intersection s times.

Using the intersection algorithm, we have been able to create an satisfied algorithm for this strict representation. It takes advantage of the simple principle that for φ to be satisfied, intersecting polyhedron P with φ must not be empty. As only a single intersection is necessary, the complexity is the same as for Algorithm 21, $O(sp^2n)$.

Algorithm 22: `satisfied-strict`(P, φ)

```
if and-strict(P, φ) = ∅ then
  return false
end if
return true
```

Finally, we can create the overapproximating union of two strict polyhedra. To union two polyhedra P and P', we can simply re-use the non-strict algorithm by unioning the non-strict parts $\bar{P}$ and $\bar{P}'$ and discarding all strictness information. This is valid since the union between two polyhedra is already an overapproximation, and removing the strictness information would only result in a slightly worse approximation.

Problems One of the first problems that arises with this structure is how to check for emptiness. An intuitive way is to check if the non-strict part is empty and if it is not, check that the non-strict part is not entirely contained in any of the strict faces. However, this can be a problem if, e.g., we have $P = \text{co}((1,1),(3,2)) \setminus \{P_1 = \text{co}((1,1),(2,1)), P_2 = \text{co}((2,1),(3,2))\}$, as shown in Figure 6.1. Here P_1 represents the constraint $x \neq 1$ and P_2 the constraint $x - y \neq 1$. Non-strict part $\bar{P} = \text{co}((1,1),(3,2))$ is not entirely contained in either, however it is entirely contained in the combination of the two strict faces. This combination is not trivially calculated as we have no algorithm for determining whether the union of two polyhedra is exact.

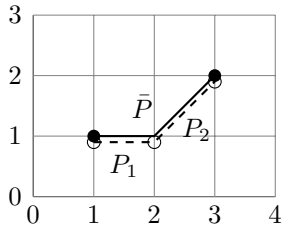


Figure 6.1: The problem of emptyness checking with strictness.

For the modifying algorithms except for intersection, the main problem is that modification – whether projecting time or resetting or freeing clocks – will sometimes require that certain strict faces must be discarded. So far, we have not been able to construct any algorithms that were able to correctly do so.

6.3 Extrapolation

As described in Chapter 4 on page 31, we have a complete extrapolation algorithm for 1-dimensional polyhedra, as well as a proof sketch for an algorithm for 2-dimensional polyhedra; however for polyhedra containing more clocks, we are unable to guarantee termination of the reachability algorithm. Therefore models with a finite number of branches or models where an upper bound has been imposed by adding invariants to all states must be used in these cases.

Nevertheless, we were able to prove a number of principles which can be used in an max-plus extrapolation algorithm, and these hold for any number of clocks.

BIBLIOGRAPHY

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] Rajeev Alur and David L. Dill. The theory of timed automata. In J. W. de Bakker, Cornelis Huizing, Willem P. de Roever, and Grzegorz Rozenberg, editors, *REX Workshop*, volume 600 of *Lecture Notes in Computer Science*, pages 45–73. Springer, 1991. ISBN 3-540-55564-1.

[7] 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`.

[8] 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.

[9] 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.

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

[11] Johan Bengtsson and Wang Yi. Timed Automata: Semantics, Algorithms and Tools. pages 87–124. 2004.

[12] Tommaso Bolognesi, Ferdinando Lucidi, and Sebastiano Trigila. From timed petri nets to timed lotos. In Luigi Logrippo, Robert L. Probert, and Hasan Ural, editors, *PSTV*, pages 395–408. North-Holland, 1990. ISBN 0-444-88810-1.

[13] 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.

[14] Patricia Bouyer, Serge Haddad, and Pierre-Alain Reynier. Timed petri nets and timed automata: On the discriminating power of zeno sequences. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming*, volume 4052 of *Lecture Notes in Computer Science*, pages 420–431. Springer Berlin / Heidelberg, 2006. doi: 10.1007/11787006_36. URL `http://dx.doi.org/10.1007/11787006_36`.

[15] 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`.

[16] 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.

[17] 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`.

[18] 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`.

[19] 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`.

[20] Hans-Michael Hanisch. Analysis of place/transition nets with timed arcs and its application to batch process control. In Marco Ajmone Marsan, editor, *Application and Theory of Petri Nets*, volume 691 of *Lecture Notes in Computer Science*, pages 282–299. Springer, 1993. ISBN 3-540-56863-8.

[21] Lasse Jacobsen and Morten Jacobsen. Timed-arc petri nets. Project report, Aalborg University, December 2010.

[22] Lasse Jacobsen and Morten Jacobsen. VerifyTAPN. `https://launchpad.net/verifytapn`, 2010. Retrieved April 7, 2011.

[23] Qi Lu, Michael Madsen, Martin Milata, and Søren Ravn. Computations on zones using max-plus algebra. Project report, Aalborg University, January 2011. `http://download.birdiesoft.dk/maxplus.pdf`.

[24] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für instrumentelle Mathematik, Bonn, 1962.

[25] F. Richman and G. Stolzenberg. Well quasi-ordered sets. *Advances in Mathematics*, 97 (2):145 – 153, 1993. ISSN 0001-8708. doi: DOI:10.1006/aima.1993.1004. URL `http://www.sciencedirect.com/science/article/pii/S0001870883710042`.

[26] V. Valero Ruiz, David de Frutos Escrig, and F. Cuartero Gomez. On non-decidability of reachability for timed-arc petri nets. In *Proc. 8th. International Workshop on Petri Nets and Performance Models*, pages 188–196, 1999.

[27] Jiří Srba. Timed-arc petri nets vs. networks of timed automata. In Gianfranco Ciardo and Philippe Darondeau, editors, *ICATPN*, volume 3536 of *Lecture Notes in Computer Science*, pages 385–402. Springer, 2005. ISBN 3-540-26301-2.

[28] Aalborg University. TAPAAL. `http://tapaal.net`, 2011. Retrieved May 26, 2011.

[29] Aalborg University. Modelling features (tapaal user manual). `http://wiki.tapaal.net/documentation/usermanual/modelling/features`, 2011. Retrieved May 31, 2011.

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

[31] Uppsala University and Aalborg University. UPPAAL DBM library. `http://www.cs.aau.dk/~adavid/udbm/index.html`, 2011. Retrieved May 25, 2011.

[32] Sergio Yovine. KRONOS: A verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer*, 1(1–2):123–133, 1997.

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

Summary

In this report, we perform real-time model checking, using a new abstract representation for the symbolic state space, namely max-plus polyhedra represented as the Minkowski sum of sets of generators. Previously we have shown that the required basic operations needed for forward (resp. backward) reachability analysis of timed models, delay, reset (resp. backward delay and free) and intersection can be done using this representation. We present the algorithms and give a suggestion for a possible optimization of the delay algorithm, which could potentially could adapted and applied to the rest. To combat the problem of state space explosion which can occur in models with extensive branching, unions, exact or overapproximating, are needed. It has been shown that max-plus polyhedra are more expressive than DBMs, and we have previously shown that the overapproximating union of max-plus polyhedra are at least as good as those for DBMs.

Additionally, an extrapolation algorithm is needed to ensure termination. We have provided theorems sufficient for creating a such algorithm for 1-dimensional polyhedra and a claim that we have theorems sufficient for 2-dimensional extrapolation. For models of more than two clocks, we cannot devise a complete extrapolation, however most of the lemmas and theorems still apply to arbitrary n-dimensional polyhedra.

The main purpose of the report was to show the approach of using max-plus polyhedra as a data structure for the symbolic state space in real-time model checking to be applicable in practice. We implement our approach on top of the model checker VerifyTAPN, for timed-arc Petri nets. The algorithms were originally developed based on the theory of timed automata, but they are easily converted to operate on timed-arc Petri nets, despite the differences in behavior between the two classes of models. We construct a model to show the concrete advantage of the ability of max-plus polyhedra to express more accurate overapproximations. The test shows for these concrete models, the DBM overapproximation always generates false positives and thus exact analysis is needed, while overapproximation with max-plus polyhedra overapproximation get the correct result, thereby not requiring exact analysis. For some of these models, this property makes the max-plus overapproximation faster than exact DBM analysis.