Université de Liège
Faculté des Sciences Appliquées

Implicit Real Vector Automata

Thèse présentée par

Jean-François Degbomont

en vue de l'obtention du grade de

Docteur en Sciences,
orientation Informatique

Année académique 2012-2013

Abstract

This thesis introduces a new data structure, the *Implicit Real Vector Automaton (IRVA)*, suited for representing symbolically polyhedra, i.e., regions of n-dimensional space defined by finite Boolean combinations of linear inequalities. IRVA can represent exactly arbitrary convex and non-convex polyhedra, including features such as open and closed boundaries, unconnected parts, and non-manifold components. In addition, they provide efficient procedures for deciding whether a point belongs to a given polyhedron, and determining the polyhedron component (vertex, edge, facet, ...) that contains a point.

An advantage of IRVA is that they can easily be minimized into a canonical form, which leads to a simple and efficient test for equality between represented polyhedra. Elementary IRVA representing primitive polyhedra, such as linear (in)equations and vector spaces are easily constructed and algorithms have been developed for computing Boolean combinations as well as projections of polyhedra represented by IRVA.

These algorithms are illustrated by complete examples of executions as a support for the comprehension of their mechanisms.

Another contribution is a first prototype implementation of an IRVA library, containing functions for building and manipulating arbitrary n-dimensional polyhedra. We reinforce the presentation of the implementation by discussing some design choices. Such choices include the use of exact arithmetic.

Finally, experimental results are presented and discussed. These experiments pave the way to future adaptations and improvements.

Résumé

Cette thèse introduit une nouvelle structure de donnée, appelée *automate implicite à vecteurs de réels*, ou *Implicit Real Vector Automaton (IRVA)*, destinée à représenter symboliquement des polyèdres, c'est-à-dire des ensembles définis par une combinaison booléenne d'un nombre fini de contraintes linéaires dans un espace à n dimensions.

Les IRVA sont capables de représenter avec exactitude des polyèdres quelconques aussi bien convexes que concaves et permettent, entre autres caractéristiques, de représenter des faces ouvertes ou fermées, de présenter des parties non connectées, etc.

De plus, ils disposent de méthodes efficaces pour décider de l'appartenance ou non d'un point donné à un polyèdre, ainsi que pour désigner dans quelle composante du polyèdre (parmi les sommets, les arrêtes, les faces, ...) le point est situé.

Un avantage des IRVA est qu'ils peuvent facilement être réduits à une forme canonique, ce qui permet de tester l'égalité de deux polyèdres représentés par des IRVA de façon très efficace.

Des IRVA élémentaires représentant des polyèdres primitifs, tels que des contraintes linéaires ou des espaces vectoriels, sont faciles à construire. Des algorithmes ont été développés pour calculer des combinaisons booléennes et des projections de polyèdres représentés par des IRVA.

Ces algorithmes sont illustrés au travers d'exemples d'exécutions complètes afin cerner leurs mécanismes avec davantage de facilité.

Une autre contribution est le développement d'une librairie prototype fournissant des fonctions de construction et de manipulation de polyèdres quelconques dans un espace à n dimensions.

La présentation de ce prototype est accompagnée d'une discussion des choix qui ont été opérés lors de son développement, tels que l'utilisation de représentations de nombres entiers de taille arbitraire.

Enfin, des résultats expérimentaux sont présentés et analysés. Ceux-ci ouvrent la voie à de futurs travaux et de futures améliorations.

Remerciements

Qu'il me soit permis de remercier M. Bernard Boigelot, le promoteur de cette thèse, pour m'avoir accordé sa confiance en me proposant de travailler sur un projet aussi intéressant. C'est en grande partie grâce à ses conseils avisés, sa constante disponibilité, ses bonnes idées et son enthousiasme contagieux qu'il m'a été possible de réaliser ce travail.

Ma gratitude est également acquise à MM. Eric Béchet, Thomas Brihaye, Pascal Gribomont, Jérôme Leroux et Pierre Wolper pour le temps qu'ils consacreront à l'évaluation de cette thèse.

De plus, je tiens à remercier M. Julien Brusten pour les nombreuses discussions et les aides ponctuelles pendant l'élaboration de ce travail.

Je profite de ces quelques lignes pour remercier aussi les membres anciens ou actuels de l'Institut Montefiore pour les agréables moments passés avec eux. Je pense notamment, entre autres, à Julien Brusten, Vincent Pierlot, Vincent Botta, Olivier Barnich, Alexander Linden, Sébastien Pierard et la "Montefiore Team".

Enfin, je remercie particulièrement Vinciane et Elisa.

Contents

Chapter 1

Introduction

The representation and the study of geometric objects have played an important role in scientific and technical history.

Polyhedra are one of the most encountered class of geometric figures. Yet a polyhedron does not have the same properties for everyone, depending on the context of application. Frequent constraints imposed on polyhedra are to have a non-zero volume, a bounded volume, or topologically closed boundaries. Such restrictions on polyhedra forbid the definition of a theory closed under Boolean operations and call for complex methods in order to regularize polyhedra obtained as the result of combination operations [Req80, Sha02].

Nef-polyhedra [Nef78] is a class of geometric entities that corresponds to regions of n-dimensional space delimited by finitely many planar boundaries. Formally, it corresponds to the class of sets definable by a finite Boolean combination of linear constraints. Such polyhedra enjoy interesting properties. Indeed, they can be non convex, have zero volume or present disjoint parts. Moreover, this definition is closed under Boolean operations. Nef-polyhedra are the class of polyhedra considered in this work.

Since polyhedra are defined over a dense domain, we can immediately exclude representations that explicitly enumerate all of their points.

A good symbolic representation should be expressive enough to represent effectively arbitrary polyhedra, and should make it possible to carry out efficiently operations such as computing Boolean combinations or projection of polyhedra, to decide whether a point belongs or not to a polyhedra and to decide in which part of the polyhedron the point is located.

Designing such a structure can be beneficial to a large variety of areas of computer science, such as Computer-Aided Design, calculation of a trajectory of a mobile object that avoids obstacles into an environment, physical simulation systems, etc.

Another, more historical, motivation behind developing such a data structure comes from *computer-aided verification*. The aim of verification of programs is to provide automated techniques for verifying that specified problematic situations do not occur during any execution of a program or system. In

model checking, the system to be verified is transposed into an abstract model [CGP00], and its reachable state space can be represented by a state machine, in which states are tuples containing a control location of the program as well as a value for each of its variables.

The state space of such system are thus sets of vectors in which one has to verify that some forbidden states do not occur. Such a test is called *a safety property*. One can explore the state space explicitly, by starting in an initial state and by following every transition [Wes78].

This approach is only possible when the number of states is finite, for example when the program uses only bounded integer variables, and small enough for the exploration to finish in a reasonable amount of time. When dealing with systems of impractical size, or when dealing with hybrid systems, i.e. systems adding real variables to simulate the flow of time [HRP94, AKNS95, BMP99, KLSV10] or simply when unbounded variables are used, one has to represent the set of states symbolically [McM93].

A symbolic representation of the state space of programs that use integer and real variables on which linear operations are performed can be based upon n-dimensional polyhedra where n is the number of variables. In order to perform symbolic state space exploration using such symbolic representations, one must be able to easily manipulate the represented polyhedra. Such manipulations consists in tests of inclusion or emptiness, and basic operations such as union, intersection, complement, Cartesian product and projection.

Over the years, different data structures have been defined to create and manipulate polyhedra. Among others, we can name Boundary-Representations [Edm60], Constructive Solid Geometry [Hof89, Req80], Cell Decompositions [Mau91] and Selective Nef Complexes [BN88, GHH$^+$03].

While each representation have its strengths and weaknesses, none perfectly fits our context of application, sometimes because they are restricted to objects with non zero volume, or forbid the definition of an efficient test for point membership. Another limitation can be a difficulty to be reduced to a canonical form, resulting in a data structure on which testing equality is difficult. Finally some data structures are inadequate for our application because they are restricted at representing only two or three dimensional polyhedra [Hof89, GHH$^+$03, Mau91].

A different approach for representing polyhedra consists in the *Real Vector Automaton* (RVA) [BBR97, BJW05]. A RVA is essentially a Büchi automaton [Büc62] used for representing sets of real vectors by employing an encoding scheme that maps those vectors into infinite words over a finite alphabet.

The advantages of RVA are that computing Boolean combinations of polyhedra reduces to applying similar operations on the languages recognized by the automata, for which simple algorithms are known. Also, RVA are an interesting choice for testing point membership, since they reduce to generating an encoding for the point coordinates and checking whether the encoding is accepted or not by the RVA.

Another good property of RVA is that they can easily be minimized into a canonical form [Löd01]. By being able to minimize RVA even for intermediate results, one ensures that the number of states does not depend on history of the creation of polyhedra. Furthermore, comparing represented polyhedra reduces to a syntactic comparison on the representing RVA.

Unfortunately, RVA are inefficient at representing polyhedra characterized by linear constraints with large coefficients since their size grows linearly with the value of those coefficients. Recent studies of RVA have brought to light valuable insight about the connection between the internal organization of RVA and the structure of their represented polyhedron [BBL09, BBB10, Bru11]. In this work, we introduce a new data structure, the Implicit Real Vector Automaton (IRVA) which keeps the good aspects of RVA by operating on similar principles, but replaces some internal structures of RVA by introducing the concept of implicit states.

The IRVA data structure enjoys interesting properties. It can straightforwardly represent linear constraints. A general *product* algorithm has been developed for applying Boolean combinations of represented polyhedron. Moreover, IRVA admits an easily computable canonical which reduces the test of equality of represented polyhedron to direct syntactic comparison. Another advantage of this canonical form is that it corresponds to a minimal size for the IRVA. A projection algorithm has also been developed in order to compute the projection of the represented polyhedron.

Earlier versions of the work done in this thesis have been published [BBD10, BBD12].

1.1 Outline of This Work

Chapter 2 : We present the basic concepts used in the rest of this work. First we present some algebraic notions by introducing set notations, defining linear constraints, affine and vector spaces and the concept of conical set. Then, with topology, we explain notions like neighborhood of points. And finally, we present language theory and recall the definitions of symbols and alphabets, words, languages and finite automata over finite and infinite words.

Chapter 3 : We define polyhedra, polyhedral components, pyramids and local pyramids and generalize the concept of polyhedron to polyhedral partitions.

Chapter 4 : This chapter begins with a survey of other works that propose data structures to represent polyhedra. We compare them with respect to a set of desirable operations on polyhedra. We then introduce the RVA data structure, along with the insights recently gained of their organization to represent polyhedra. We connect this organization to polyhedral components defined in Chapter 3.

Chapter 5 : We define the IRVA data structure. First, with an abstract and simple data structure. It is based on the same principles than RVA. We show how this abstract structure can be organized in order to operationally be able to manipulate it by the use of the test of inclusion of a point, *the point decision problem* as a case study. We show the design choices that enable to keep the fundamental mechanics of RVA on this abstract data structure. We then introduce the IRVA data structure by first defining how the previously made choices translates concretely in IRVA. Then, a formal and complete definition of the syntax of the structure is given. We enumerate a set of syntactic constraints that IRVA must respect in order to be syntacticly valid. After those considerations, we give a semantics to the data structure with the point decision procedure. We then give a property that each IRVA must satisfy in order to be *well formed*. The chapter ends with a complete example of creation of an instance of the data structure for a particular polyhedron.

Chapter 6 : We present the canonical form of IRVA. We motivate our need for such canonical form and make some observations about the reasons why two IRVA representing the same polyhedron can take different forms. We then propose an algorithm that computes the canonical form of any IRVA. Additionally we present a complete and detailed execution of the algorithm on a given example.

Chapter 7 : This chapter presents a technique to combine two IRVA in order to perform some combination operation like, for example, the union, intersection or difference, of their represented polyhedra. First, we explain how elementary IRVA can be constructed from linear constraints or vector spaces. Then, we present the concepts of product of two IRVA and color function as the general scheme to define any combination operation. We then propose an algorithm to compute the product of two IRVA. Finally, an execution of this algorithm on two given IRVA is detailed as an example.

Chapter 8 : We define two different type of projections. First, we propose an algorithm to compute a lower dimensional IRVA from an input IRVA as an image by a particular case of projection, the aligned projection. We then see a complete example with an execution of the algorithm on a given IRVA. Second, we analyze our algorithm to see how it could be adapted to compute images of the input IRVA by a generalized notion of projection.

Chapter 9 : A prototype implementation of the IRVA data structure and its manipulation algorithms has been developed. We first present the features of this library. Then, we detail and discuss design choices made in this implementation. We then explain how the data structure itself is implemented, and what data types and organization it uses.

We identify a set of primitive tools that are used in multiple parts of the program and present how and when they are used, in order to point out which procedures are critical in order for a future package to be practical. After that, we present additional secondary data structures used in the implementation of the procedures presented by algorithms inside this work.

Then, we show the measured results based on practical tests. We describe some case studies in order to give meaning to the measures. We propose an analysis of those results.

Chapter 2

Basic Notions

2.1 Algebra

Although we assume that the reader has knowledge in basic set theory, we will present notations for sets of numbers used in this work.

The symbols $\mathbb{N}, \mathbb{Z}, \mathbb{Q}$ and $\mathbb{R}$ denote, respectively, the sets of natural numbers, integer numbers, rational numbers and real numbers. Also, we use the notation $S_{\#x}$ with $S \in \{\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$, $\# \in \{\leq, <, =, \neq, >, \geq\}$ and $x \in \mathbb{R}$ to represent $\{x' \in S \mid x' \# x\}$. For example, the set of strictly negative real numbers is noted $\mathbb{R}_{<0}$.

A point in the Euclidean $n-$dimensional space $\mathbb{R}^n$ is characterized by a vector with n components.

If $\vec{v} = (v_1, v_2, \ldots, v_n) \in \mathbb{R}^n$ is a point, then $\vec{v}[i]$ denotes its i^{th} component v_i.

In contexts where the dimension n is clearly defined, we use the symbol $\vec{0}$ for $(\underbrace{0, 0, \ldots, 0}_{n})$.

Definition 2.1 *Let $\vec{p} = (p_1, p_2, \ldots, p_n)$ be a point of $\mathbb{R}^n$. The **aligned projection** of $\vec{p}$ onto the coordinate component different from i, or in short w.r.t. i, noted $\vec{p}_{|\neq i}$, is the point*

$$\vec{p}_{|\neq i} = (p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n).$$

□

Definition 2.2 *Let $n \in \mathbb{N}$ be a dimension. A **linear constraint** over points $\vec{x} \in \mathbb{R}^n$ is a constraint of the form*

$$\vec{a} \cdot \vec{x} \# b$$

with $\vec{a} \in \mathbb{Z}^n$, $b \in \mathbb{Z}$ and $\# \in \{<, \leq, =, \geq, >\}$. □

When a linear constraint is of the form $\vec{a} \cdot \vec{x} = b$, we call it a *linear equality*. Otherwise, it is a *linear inequality*.

2.1.1 Affine and Vector Spaces

Definition 2.3 *A vector space X over $\mathbb{R}^n$ is a set of vectors closed under arbitrary finite linear combinations or, equivalently, the set of all linear combinations of a given finite set of vectors.* □

A basis of the vector space X is a minimal set of vectors $V' \subseteq V$ such that the coordinates of every point of $\mathbb{R}^n$ can be expressed as a linear combination of vectors of V'.

Definition 2.4 *A set $S \in \mathbb{R}^n$ is a **vector subspace** of a vector space X if it is a vector space that is a subset of X.* □

Definition 2.5 *The set of the solutions of a finite conjunction of linear equalities is an **affine space**.* □

Property 2.6 *Let $n \in \mathbb{N}$ be a dimension, a set $A \subseteq \mathbb{R}^n$ is an affine space iff there exist a vector space $S \subseteq \mathbb{R}^n$ and a vector $\vec{v} \in \mathbb{R}^n$ such that*

$$A = S + \vec{v}.$$

Sometimes, it can be useful to find the smallest affine space that contains a particular set S. For example, for a segment $[AB]$, with $A \neq B$, it corresponds to the (infinite) line passing by A and B. Such a superset is called the *affine closure* of S.

Definition 2.7 *Let $n \in \mathbb{N}$ be a dimension. The **affine closure** of a set $S \subseteq \mathbb{R}^n$, noted* $\text{aff}(S)$*, is the intersection of all affine spaces that contain S.* □

At some point of this work, we will study sets that preserve their shapes at any zoom factor around some points.

Definition 2.8 *Let $n \in \mathbb{N}_0$ be a dimension. A set $S \in \mathbb{R}^n$ is **conical** with respect to the apex $\vec{v} \in \mathbb{R}^n$ if and only if for all $\vec{x} \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}_{>0}$, we have :*

$$\vec{x} \in S \Leftrightarrow \vec{v} + \lambda(\vec{x} - \vec{v}) \in S.$$

□

A cone can have more than one apex. We have the following property :

Property 2.9 *The set of apexes of a cone forms an affine space.*

By extension, if a cone has $\vec{0}$ for apex, the set of its apexes is a vector space.

2.2 Topology

Topology is an important tool for the study of local properties of shapes of sets. In this work, we use Euclidean distance as base metric.

Definition 2.10 *Let $n \in \mathbb{N}$ be a dimension and $\vec{x}$,$\vec{y} \in \mathbb{R}^n$ be two points. The* ***Euclidean distance*** *between $\vec{x}$ and $\vec{y}$ is :*

$$d(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{n} (\vec{x}[i] - \vec{y}[i])^2}.$$

□

We are now ready to define the notion of *neighborhood* of a point of space. Intuitively, any open region that contains the point is a neighborhood of this point.

In this work, we introduce a particular type of neighborhood for a point : the *cubic neighborhood*. It consists in a cubic region centered on the point and characterized only by this point and a size. We obtain the following definition :

Definition 2.11 *Let $\vec{v}$ be a point in $\mathbb{R}^n$ and $\varepsilon \in \mathbb{R}_{>0}$. The* ***cubic neighborhood*** *of size ε of $\vec{v}$ is the set*

$$N_\varepsilon(\vec{v}) = \left(\vec{v}[1] - \frac{\varepsilon}{2}, \vec{v}[1] + \frac{\varepsilon}{2}\right) \times \left(\vec{v}[2] - \frac{\varepsilon}{2}, \vec{v}[2] + \frac{\varepsilon}{2}\right) \times \cdots \times \left(\vec{v}[n] - \frac{\varepsilon}{2}, \vec{v}[n] + \frac{\varepsilon}{2}\right).$$

□

The notation (a, b) refers to the open interval $\{x \mid a < x < b\}$.

A point is a boundary of a set S iff it is arbitrarily close to points in S as well as points not in S. Formally we have the following definition :

Definition 2.12 *The* ***boundary*** *of a set S of points of $\mathbb{R}^n$, is the set*

$$\{\vec{v} \mid (\exists \vec{v'}, \vec{v''} \in N_\varepsilon(\vec{v}))(\vec{v'} \in S \wedge \vec{v''} \notin S))\}$$

with $\varepsilon \in \mathbb{R}_{>0}$. □

2.3 Words, Languages and Automata

2.3.1 Words

A word is a sequence of symbols taken out of a finite set, called an alphabet. A word can either be finite or infinite. The length of a finite word, denoted by $|w|$, is the number of symbols it contains.

Definition 2.13 *A **finite word** of length k over an alphabet Σ is a finite sequence $\sigma_1\sigma_2\ldots\sigma_k$ such that for all $i \in \{1, 2, \ldots, k\}$, $\sigma_i \in \Sigma$.* □

Definition 2.14 *An **infinite word** over an alphabet Σ is an infinite sequence*

$$\sigma_0\sigma_1\ldots$$

where for all $i \in \mathbb{N}$ we have $\sigma_i \in \Sigma$. □

A word composed of no symbols at all it is said to be *empty* and is denoted by ε.

Let w_1 be a finite word and w_2 be a finite or infinite word. The concatenation of w_1 and w_2, in this order, is denoted by w_1w_2.

2.3.2 Languages

Sets of words are called *languages*. We have the following definition :

Definition 2.15 *A **finite (infinite) language** over an alphabet Σ is a finite (infinite) set of words, each defined over Σ.* □

When a language is finite, it can be represented just by enumerating its content. With an infinite one, this is not possible. Let us introduce a representation of languages that enables the definition of all finite languages and some infinite languages. The family of those languages is called *regular languages*.

A regular language can be

- empty,
- a single word composed of one symbol taken out of the alphabet,
- a combination of two other languages.

Languages can be combined by *union*, *difference*, *concatenation* or *Kleene closure*.

Definition 2.16 *The **union** of a language L_1 and a language L_2 is the language :*

$$L_1 \cup L_2 = \{w \mid w \in L_1 \vee w \in L_2\}.$$

□

Definition 2.17 *The **difference** of finite-word language L_1 and a language L_2 is the language :*

$$L_1 \setminus L_2 = \{w \mid w \in L_1 \wedge w \notin L_2\}.$$

□

Definition 2.18 *The **concatenation** of a language of finite words L_1 and a language L_2 is the language :*

$$L_1 L_2 = \{w_1 w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2\}.$$

□

The Kleene closure of a finite non-empty language L is the infinite language L^* that contains the empty word and all possible concatenations of words of L. More formally, we have :

Definition 2.19 *The **Kleene closure** of a finite language L of finite words is the language L^* such that :*

$$L^* = \{w \mid (\exists k \in \mathbb{N})(\exists w_1, w_2, \ldots, w_k \in L)(w = w_1 w_2 \ldots w_k)\}.$$

□

In this work, we also use two variants of the Kleene closure operator. The first one is noted L^+, where L is a language of finite words. It is defined as follows :

$$L^+ = LL^*.$$

The second one is noted L^ω, with L a language of finite words. It is used to generate infinite languages from finite ones and is defined as :

$$L^\omega = \{w_0 w_1 w_2 \ldots \mid (\forall i \in \mathbb{N})(w_i \in L \wedge w_i \neq \varepsilon)\}.$$

2.3.3 Automata

An *automaton* is a finite-state machine that recognizes a language. It consists of a set of states linked by a transition relation. Each transition is characterized by a symbol and a pair of states, the origin and the destination of this transition. Some states are *initial states*. Every state can be either *accepting* or *not accepting*. Without loss of generality, we will only consider automata with a unique initial state. Indeed, automata with more than one initial state can always be adapted to have only one initial state and still accept the same language [HU79].

Definition 2.20 *A finite automaton is a tuple* $(S, \Sigma, \delta, s_0, F)$ *with :*

- S *: a finite set of states,*
- Σ *: an alphabet,*
- $\delta : S \times \Sigma \to S$ *: a (partial) transition function.*
- $s_0 \in S$ *: an initial state,*
- $F \subseteq S$ *: a set of accepting states,*

□

We will now study how automata accept finite-word languages and infinite-word languages.

Finite-word languages : For a finite word $w = \sigma_1\sigma_2\sigma_3 \dots \sigma_k$ with $k = |w|$ and, for all $i \in \{1, 2, 3, \dots, k\}$, $\sigma_i \in \Sigma$ to be accepted by an automaton $(S, \Sigma, \delta, s_0, F)$, there must exist a sequence of the form $q_0q_1q_2 \dots q_k$ with $q_k \in F$ such that $q_0 = s_0$ and for all $i \in \{0, 1, \dots, k\}$ we have $q_i \in S$, and such that for all $i \in \{1, \dots, k\}$, we have $\delta(q_{i-1}, \sigma_i) = q_i$. If the word is ε, s_0 belongs to F.

Infinite-word languages : There exists different conditions for accepting infinite words. One of them has been proposed by Büchi in [Büc62]. We will use this accepting scheme in this work. It can be explained as follows : for an infinite word $w = \sigma_1\sigma_2\sigma_3 \dots$ with $\sigma_i \in \Sigma$ to be accepted by an automaton $(S, \Sigma, \delta, s_0, F)$, there must exist an infinite sequence E of the form $s_0s_1s_2 \dots$ such that for all $i \in \mathbb{N}_{>0}$ we have $s_i \in S$, $\delta(s_{i-1}, \sigma_i) = s_i$ and E contains an infinite number of occurrences of states of F.

Reading a word of size k into an automaton from a state s_1 identifies a sequence of the form :

$$s_1 \xrightarrow{\sigma_1} s_2 \xrightarrow{\sigma_2} \dots \xrightarrow{\sigma_k} s_{k+1}$$

Such sequence is called a **run** of the automaton.

By following transitions of a finite automaton, it is possible to wander from state to state. When there exist a path leading from one state s_1 to another s_2, we say that the later is *reachable* from the first. It means that it is possible to find a word such that reading it by the transition function from s_1 terminates into s_2.

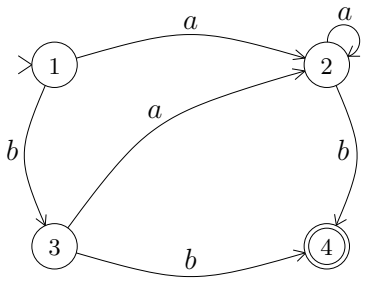


Figure 2.1: Example of finite-word automaton accepting the language $(\varepsilon \cup b)(a^+b) \cup (bb)$.

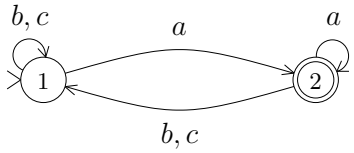


Figure 2.2: Example of automaton accepting the language $((b \cup c)^*a)^\omega$.

Definition 2.21 *Let $(S, \Sigma, \delta, s_0, F)$ be a finite automaton. A state $s' \in S$ is **reachable** from a state s with respect to δ (noted $s \xrightarrow{\delta} s'$) iff one of the following conditions is satisfied :*

- $s = s'$
- $(\exists \sigma \in \Sigma)(\delta(s, \sigma) \to s')$

□

Example 2.22 *Figure 2.1 shows an automaton accepting the finite-word language $(\varepsilon \cup b)(a^+b) \cup (bb)$. The alphabet is $\{a, b\}$, the set of states is $\{1, 2, 3, 4\}$, the initial state is state 1, the set of accepting states is $\{4\}$, and the transition function δ is given by the following table :*

s	$\delta(s, a)$	$\delta(s, b)$
1	2	3
2	2	4
3	2	4
4	$\perp$	$\perp$

◇

Example 2.23 *Figure 2.2 shows an automaton accepting the infinite-word language* $((b \cup c)^* a)^\omega$*. This is the language of words over the alphabet* $\{a, b, c\}$ *containing the words with an infinite number of* a*. The set of states is* $\{1, 2\}$*, the initial state is state* 1*, the set of accepting states is* $\{2\}$*, and the transition function* δ *is given by the following table :*

s	$\delta(s, a)$	$\delta(s, b)$	$\delta(s, c)$
1	2	1	1
2	2	1	1

◇

As mentioned previously, Büchi automata accept infinite-word languages. It appears clear that, since the number of states in such an automaton is finite, their transition function must contain cycles. A run reading an infinite word must then end in a set of states visited infinitely many times. Let us now study briefly how those sets of states can be characterized more precisely.

With the notion of reachability, we can identify maximal subsets of states such that every state from this subset can be reached from any other state of it by following transitions defined by the transition function. Such a subset is called a *strongly connected component*.

Definition 2.24 *A* ***Strongly Connected Component*** *(SCC) of a finite automaton* $(\Sigma, S, s_0, F, \delta)$ *is a maximal subset* $S' \subseteq S$ *such that :*

$$(\forall s, s' \in S)(s \to s')$$

□

Note that a single state that does not belong to any cycle of an automaton forms a SCC composed only of itself. Such a SCC is called a *trivial strongly connected component*.

Chapter 3

Polyhedra and Polyhedral Partitions

3.1 Polyhedra and Pyramids

A polyhedron is a set of points in $\mathbb{R}^n$ that satisfies linear constraints, but the precise definition of a polyhedron can vary greatly depending on the context in which it is used. A painter could consider a single point to be a very simple polyhedron, an architect could be a little more restrictive and impose that a polyhedron must always have some volume and a mathematician could declare the previous two uninteresting because they exclude infinite or empty sets. All of them will, however, agree on the fact that they correspond to a region of space delimited by planar boundaries.

Over the years, different classes of polyhedra have been defined by various authors. The lack of a unified definition is in fact quite surprising for a class of geometric entities that is so ubiquitous. A possible reason for this resides on the fact that polyhedra are often considered as solid or continuous entities and that they always have a representation in our spatial environment [Bru90]. Although it is true that three-dimensional polyhedra can be used to represent solid objects with planar boundaries, limiting their definition to such solids is somehow problematic. Indeed, if we need polyhedra to be closed under Boolean operations – intersection, union, complement and difference – we quickly come to realize that a definition depicting polyhedra as solid objects is inconsistent. For example, the intersection of two cubes sharing only a vertex has zero volume, although it is not empty. This need for a precise and clean theory of polyhedra is however of great importance for computer graphics or computer assisted design (CAD) amongst other fields.

A class of polyhedra closed under Boolean combinations has been studied by W. Nef [Nef78], and corresponds to the class of polyhedra that we will consider in this work. We therefore define a *polyhedron* as a set satisfying a finite Boolean combination of linear constraints over $\mathbb{R}^n$.

Definition 3.1 *A **polyhedron** $\pi \subseteq \mathbb{R}^n$ is a set defined as follows :*

$$\underset{i=1}{\overset{k}{\mathcal{B}}} \left(\vec{a}_i \cdot \vec{x}_i \ \#_i \ \vec{b}_i\right)$$

where $\mathcal{B}$ is a finite Boolean combination of k linear constraints, and for all i, $\vec{a}_i$ and $\vec{b}_i \in \mathbb{R}^n$ and $\#_i \in \{\leq, <, =, >, \geq\}$. □

With this definition, we have a theory of polyhedra closed under Boolean operations, which is precious for their construction and manipulation. Indeed, the properties of polyhedra are such that they can have *open* and *closed boundaries* as well as *non-convex*, *unconnected*, *dangling* and *non-manifold parts*.

Let us analyse those properties and explain their impact on the shapes of polyhedra.

Open boundary means that points of the boundary of a polyhedron does not belong to the polyhedron itself. A boundary is a set of points such that, every neighborhood of these points, contains points belonging to the polyhedron and points that do not belong to it. Figure 3.2(a) shows a 2-dimensional polyhedron with an open boundary (labeled c).

Closed boundary means that points of the boundary does belong to the polyhedron. Figure 3.2(b) shows a 2-dimensional polyhedron with three closed boundaries.

Non-convex means that it is possible to find two points inside the polyhedron such that the segment connecting them contains a point outside the polyhedron. Figure 3.2(c) shows a non convex 2-dimensional polyhedron. We can see that the segment linking $\vec{x}$ and $\vec{y}$, two points of the polyhedron, contains both points inside and outside the polyhedron.

Unconnected parts refers to polyhedra composed of disjoint parts. Figure 3.2(d) shows a 2-dimensional polyhedron with two distinct parts.

Dangling parts or regions of the n-dimensional polyhedron that contain points but have no volume in $\mathbb{R}^n$. Figure 3.2(e) shows a 2-dimensional polyhedron with a dangling edge.

Non-manifold refers to a property of the boundary of the polyhedron. A boundary is said to be a d-manifold, if there exists, for each point of this boundary, a neighborhood such that the boundary inside this neighborhood is homeomorphic to a d-dimensional space [Per01]. Intuitively, it means that there is a way of deforming the boundary enclosed in the neighborhood in such a way that it coincides with a hyperplane of d-dimension. If one can not deform the boundary of a polyhedron without tearing it apart, this polyhedron is said to be a *non-manifold*. Figure 3.2(f) shows such a 2-dimensional polyhedron.

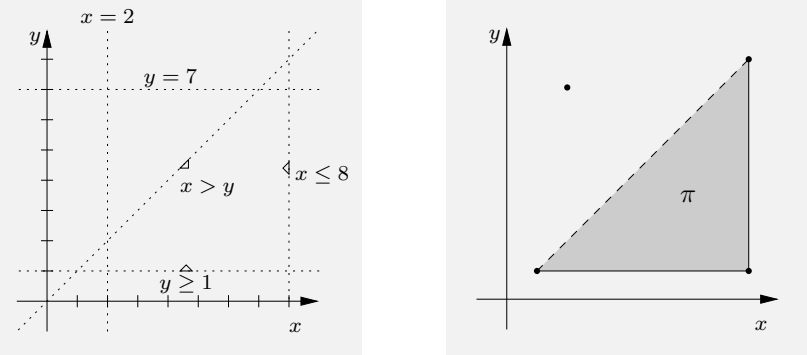


Figure 3.1: The polyhedron $\pi \subset \mathbb{R}^2$ represented by $(x = 2 \wedge y = 7) \vee (x > y \wedge x \leq 8 \wedge y \geq 1)$.

Example 3.2 *Figure 3.1 shows the polyhedron $\pi \subset \mathbb{R}^2$ corresponding to the set : $(x = 2 \wedge y = 7) \vee (x > y \wedge x \leq 8 \wedge y \geq 1)$. This polyhedron has two unconnected parts, one is an isolated point (and is thus a non-manifold component), the other is a triangle with one open and two closed boundaries.* ◇

In the context of this work, we are interested in pyramids, a particular type of polyhedra that shows a local invariance by scaling operation around particular points.

Definition 3.3 *A **pyramid** is a conical polyhedron.* □

Or, said otherwise, a pyramid is a conical set defined by a finite Boolean combination of linear constraints over points of $\mathbb{R}^n$.

As a consequence, we can observe the following property.

Property 3.4 *The set of apexes of a pyramid forms an affine space [BN88].*

3.1.1 Polyhedral Components

It is known that the structure of every polyhedron $\pi \subseteq \mathbb{R}^n$ is pyramidal in arbitrarily sufficiently small neighborhoods of any point $\vec{v} \in \mathbb{R}^n$ [BN88, BBL09].

Formally, we have the following theorem :

Theorem 3.5 *Let $\pi \subseteq \mathbb{R}^n$ be a polyhedron. For every point $\vec{v} \in \mathbb{R}^n$, there exists $\varepsilon \in \mathbb{R}_{>0}$ and a pyramid $\pi' \subseteq \mathbb{R}^n$ of apex $\vec{v}$, such that*

$$\pi \cap N_\varepsilon(\vec{v}) = \pi' \cap N_\varepsilon(\vec{v}).$$

Definition 3.6 *Let $\pi \subseteq \mathbb{R}^n$ be a polyhedron, and $\vec{v} \in \mathbb{R}^n$ be an arbitrary point. The **local pyramid** adjoined to π in $\vec{v}$ is the pyramid $P_\pi(\vec{v})$ such that*

$$\pi \cap N_\varepsilon(\vec{v}) = P_\pi(\vec{v}) \cap N_\varepsilon(\vec{v})$$

with $\varepsilon \in \mathbb{R}_{>0}$ being small enough for the local pyramid to be defined. □

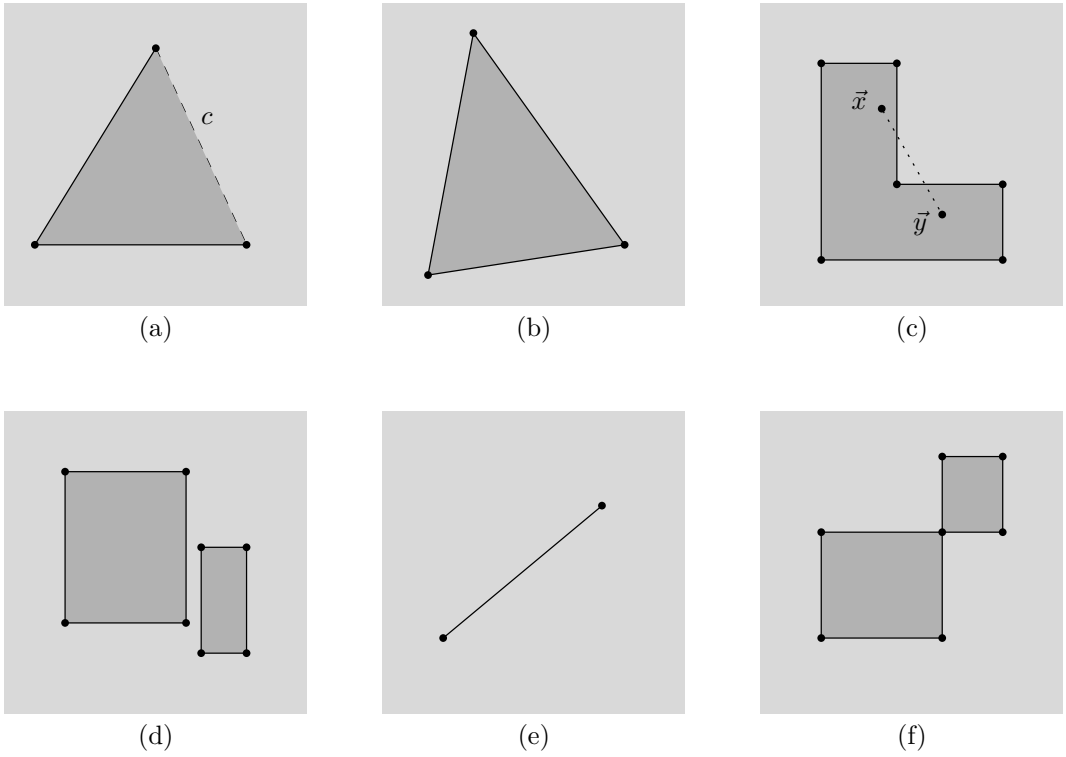
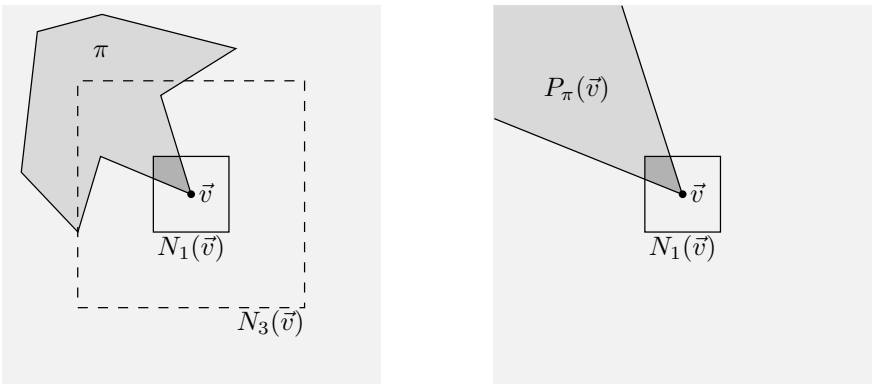


Figure 3.2: Examples of 2-dimensional polyhedra with (a) An open boundary. (b) Closed boundaries. (c) Non-convex shape. (d) Unconnected parts. (e) Dangling part. (f) Non-manifold polyhedron.



(a) (b)

Figure 3.3: Example of (a) a polyhedron π and (b) the adjoined local pyramid $P_\pi(\vec{v})$ in $\vec{v}$.

The local pyramid adjoined to a polyhedron in a point of space is what an infinitely small observer would see within a bounded horizon. Local pyramids make it possible to decompose an arbitrary polyhedron into a set of local pyramids and it has been established that a finite number of distinct pyramids are sufficient to describe the whole structure of any polyhedron [BN88, BBL09].

Theorem 3.7 *[BN88] For each polyhedron $\pi \subseteq \mathbb{R}^n$, the set*

$$\{P_\pi(\vec{v}) \mid \vec{v} \in \mathbb{R}^n\}$$

is finite.

Example 3.8 *Figure 3.3 shows the pyramid π' coinciding with a polyhedron π in the cubic closed neighborhood of size 1 of the point $\vec{v} \in \mathbb{R}^n$. We can see that there exists no pyramid coinciding with π for the dashed neighborhood of size 3 of point $\vec{v}$.* ◇

We know that for any polyhedron $\pi \subseteq \mathbb{R}^n$, and every point $\vec{v} \in \mathbb{R}^n$ a local pyramid $P_\pi(\vec{v})$ is defined. It is then possible to regroup points of $\mathbb{R}^n$ into a finite number of equivalence classes, based on the adjoined local pyramid associated to each of them. We call those equivalence classes *polyhedral components*, or more simply *components* of π.

Definition 3.9 *Let $\pi \subseteq \mathbb{R}^n$ be a polyhedron and $\vec{v}, \vec{v'} \in \mathbb{R}^n$ two points. The relation $\overset{\pi}{\sim}$ over $\mathbb{R}^n \times \mathbb{R}^n$ is defined as $\vec{v} \overset{\pi}{\sim} \vec{v'}$ iff $P_\pi(\vec{v}) = P_\pi(\vec{v'})$.* □

Theorem 3.10 *Let $\pi \subseteq \mathbb{R}^n$ be a polyhedron and $\vec{v}, \vec{v'} \in \mathbb{R}^n$ two points. The relation $\vec{v} \overset{\pi}{\sim} \vec{v'}$ is an equivalence relation.*

Definition 3.11 *A **polyhedral component** of a polyhedron $\pi \subseteq \mathbb{R}^n$ is an equivalence class of the relation $\overset{\pi}{\sim}$.* □

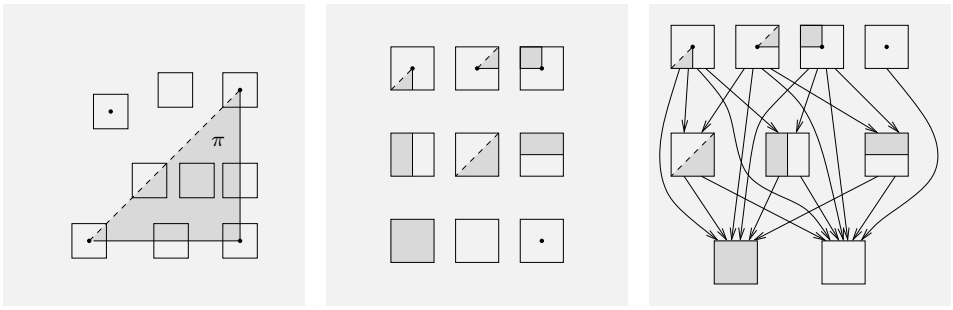


Figure 3.4: (a) polyhedron π from Fig. 3.1, (b) equivalence classes based on local pyramids, (c) incidence between equivalence classes.

Since all points of a component have the same adjoined pyramid, it is possible to define the unique adjoined local pyramid in a component of a polyhedron.

Definition 3.12 *Let $\pi \subseteq \mathbb{R}^n$ be a polyhedron, the **local pyramid** adjoined to π in a polyhedral component C is the pyramid :*

$$P_\pi(C) = P_\pi(\vec{v})$$

with $\vec{v} \in C$. □

For a polyhedron $\pi \subseteq \mathbb{R}^n$, and a polyhedral component C of π, all the points of C are apexes of the unique local pyramid π' associated to every point of C. The set of all apexes of π' forms the *characteristic affine space* of the component C, denoted $\text{aff}(C)$. The dimension of this space defines the *dimension* of the component, denoted $\dim(C)$. Intuitively, for three dimensional-polyhedra, the dimension of a component characterizes the number of degree of freedom among its points. Components of dimension 0, 1, 2 thus correspond to the classical notions of *vertexes*, *edges* and *facets* of polyhedra.

3.1.2 Incidence Relation

When we consider components of a polyhedron with concepts such as vertexes, edges and facets, it is usual to consider that some of them are *connected*. For example, it is common to think that a vertex is the result of the intersection of two (or more) edges.

Formally, the components of a polyhedron are connected by an incidence relation defined as follows.

Definition 3.13 *A component C_2 of a polyhedron $\pi \subseteq \mathbb{R}^n$ is incident to a component C_1, which is denoted $C_1 \preceq C_2$, if for every point of C_1 and every $\varepsilon > 0$ there exist points of C_2 that belongs to $N_\varepsilon(\vec{v})$.* □

If we look at our previous example of a vertex being the intersection of two edges, it seems quite clear that any neighborhood of the vertex will always contain points belonging to both edges.

The reverse property also holds. It is not true that this vertex is present in any neighborhood of every point of any of the two edges.

This observation leads us to the following theorem.

Theorem 3.14 *For every polyhedron, the incidence relation $\preceq$ is a partial order over its components. This relation is such that*

$$C_1 \preceq C_2 \Rightarrow \text{aff}(C_1) \subseteq \text{aff}(C_2)$$

Proof : The relation $\preceq$ is :

Reflexive : Let C be a component, having $C \preceq C$ defined would mean that in every neighborhood of any point of C, we have points of C. This is trivially true as a point always belongs to any of its own neighborhood.

Antisymmetric : Let C_2 be a component incident to another component C_1. We known, by definition, that C_1 is a set of apexes of the set of apexes of C_2. Let us show that $C_1 \preceq C_2 \wedge C2 \preceq C_1 \Rightarrow C_1 = C_2$. Would $C2 \preceq C_1$ be defined, it would mean that in any neighborhood of any point of C_2, there is always a point of C_1. Which would mean that C_2 is a set of apexes of the set of apexes of C_1. It can only be true if $C_1 = C_2$.

Transitive : Let C_1, C_2 and C_3 be components of a polyhedron $\pi \subset \mathbb{R}^n$. If $C1 \preceq C_2$ is defined, it means that in any neighborhood of any point of C_1, there is always a point of C_2. If $C2 \preceq C_3$ is defined, it means that in any neighborhood of any point of C_2, there is always a point of C_3. A neighborhood N of C_1 is an open set, and it contains at least one point of C_2. As N contains a point of C_2, it is also a valid neighborhood of this point of C_2, which implies that there is always a point of C_3 in N. We obtain that there is always a point of C_3 in any neighborhood of C_1. We have $C_1 \preceq C_3$.

■

3.2 Extension to Polyhedral Partitions

The complement of a polyhedron $\pi \subseteq \mathbb{R}^n$ is the set $\{\vec{v} \in \mathbb{R}^n \mid \vec{v} \notin \pi\}$, which is also a polyhedron. Thus, a polyhedron and its complement form a binary partition of $\mathbb{R}^n$.

We can extend this idea to more general partitions of $\mathbb{R}^n$ by introducing a notion of *color* (or symbol) associated to a polyhedron.

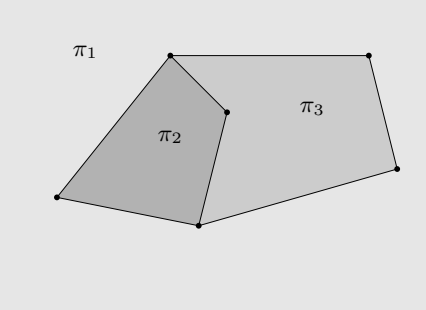


Figure 3.5: Example of a polyhedral partition $\Pi = \{\pi_1, \pi_2, \pi_3\}$ of $\mathbb{R}^2$.

Definition 3.15 *A **polyhedral partition** $\Pi = \{\pi_1, \pi_2, \ldots, \pi_m\}$ is a partition of $\mathbb{R}^n$ such that each part π_i is a polyhedron.* □

By definition, parts of a partition are *collectively exhaustive*, this means that their union covers $\mathbb{R}^n$ and *mutually exclusive*, this means that parts are mutually disjoint.

Polyhedral partitions also be seen as coloring functions. If a color is associated to each part of a polyhedral partition $\Pi = \{\pi_1, \pi_2, \ldots, \pi_m\}$, then Π associates a color to every point of $\mathbb{R}^n$. More formally, a polyhedral partition is a function $\Pi : \mathbb{R}^n \to \{1, 2, \ldots, m\}$ of points of $\mathbb{R}^n$, where the color corresponds to an integer in $[1, m]$.

Note that a polyhedron is a particular instance of a polyhedral partition, associating only two colors to points of $\mathbb{R}^n$, which can be called *in* and *out*.

If a polyhedral partition is not affected by a uniform scaling operation around a point $\vec{v} \in \mathbb{R}^n$, then it is also said to be pyramidal with respect to the apex $\vec{v}$.

Definition 3.16 *A **pyramidal partition** $\Pi = \{\pi_1, \pi_2, \ldots, \pi_m\}$ over $\mathbb{R}^n$ is pyramidal with respect to the apex $\vec{v}$ if for every $\vec{x} \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}_{>0}$, we have*

$$\Pi(\vec{v}) = \Pi(\vec{v} + \lambda(\vec{x} - \vec{v}))$$

□

Definition 3.17 *Let $\Pi = \{\pi_1, \pi_2, \ldots, \pi_m\}$ be a polyhedral partition of $\mathbb{R}^n$, and $\vec{v} \in \mathbb{R}^n$ be an arbitrary point. The **local pyramidal partition** of $\mathbb{R}^n$ adjoined to Π in $\vec{v}$ is the pyramidal partition*

$$P_\Pi(\vec{v}) = \{\pi' \subseteq \mathbb{R}^n \mid (\exists 0 < i \leq m)(\pi' = P_{\pi_i}(\vec{v}) \wedge P_{\pi_i}(\vec{v}) \neq \emptyset)\}$$

□

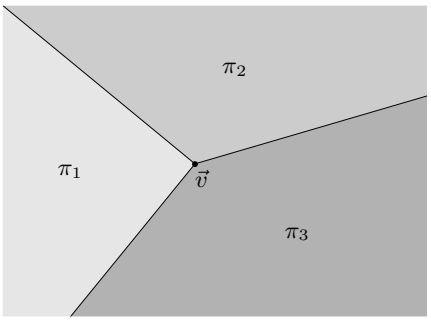


Figure 3.6: Example of a pyramidal partition $\Pi = \{\pi_1,\pi_2,\pi_3\}$ of $\mathbb{R}^2$ of apex $\vec{v}$.

Theorem 3.18 *For each polyhedral partition* Π *of* $\mathbb{R}^n$*, the set*

$$\{P_\Pi(\vec{v}) \mid \vec{v} \in \mathbb{R}^n\}$$

is finite.

Proof : A local pyramidal partition of $\mathbb{R}^n$ is, by definition, a set of local pyramids. By Theorem 3.7, we know that for any polyhedron only a finite number of local pyramids will be associated to the whole set of points of $\mathbb{R}^n$. We can then conclude that if the number of parts is finite, the set $\{P_\Pi(\vec{v}) \mid \vec{v} \in \mathbb{R}^n\}$ is finite. ■

Another proof can be found in [Nef78].

Definition 3.19 *Let* Π *be a polyhedral partition of* $\mathbb{R}^n$ *and* $\vec{x}, \vec{y} \in \mathbb{R}^n$ *two points. The relation* $\overset{\Pi}{\sim}$ *over* $\mathbb{R}^n \times \mathbb{R}^n$ *is defined as* $\vec{x} \overset{\Pi}{\sim} \vec{y}$ *iff* $P_\Pi(\vec{x}) = P_\Pi(\vec{y})$. □

Theorem 3.20 *The relation* $\overset{\Pi}{\sim}$ *is an equivalence relation.*

Definition 3.21 *A polyhedral component of a polyhedral partition* Π *of* $\mathbb{R}^n$ *is an equivalence class of the relation* $\overset{\Pi}{\sim}$. □

Again, we will just call *component* a polyhedral component of a polyhedral partition of $\mathbb{R}^n$.

Definition 3.22 *A component* C_2 *of a polyhedral partition* Π *of* $\mathbb{R}^n$ *is incident to a component* C_1*, which is denoted* $C_1 \preceq C_2$*, if for every point* $\vec{v} \in C_1$*, there exist points of* C_2 *inside* $N_\varepsilon(\vec{v})$*, for any value of* ε. □

3.3 Minimum Polyhedral Component

We have seen that a pyramidal polyhedron is conical with respect to a certain set of apexes. This brings the following theorem on pyramidal partitions :

Theorem 3.23 *If a polyhedral partition* Π *of* $\mathbb{R}^n$ *is pyramidal, then it contains a unique polyhedral component* C *that is minimum with respect to the incidence relation* $\preceq$*. Formally, it means that for all component* C_i *of* Π*, we have* $C \preceq C_i$*. This minimum component corresponds to the set of apexes of* Π*.*

Proof : Let C_m be the set of apexes of Π. The local pyramidal partition associated to every point of C_m is Π. Indeed, the local pyramidal partition adjoined to a point a is invariant to scaling around a, which is precisely the definition of a pyramidal partition having a as apex. Thus, C_m is a polyhedral component. Each polyhedral component of a pyramidal partition is a pyramid (again, polyhedral components are invariant by scaling around points in their set of apexes). This implies that every polyhedral component has points arbitrarily close to points of C_m. By the definition of the incidence relation, we deduce that each polyhedral component C of Π is such that $C_m \preceq C$. This proves that C_m is the minimum element with respect to the incidence relation. ■

This theorem is true for pyramidal partitions, but is not always true when considering only a certain region of space covering a certain subset of the components of a pyramidal partitions. We have the following definition for the minimum component of a subset of components of a polyhedral partition.

Definition 3.24 *Let* Π *be a pyramidal partition of* $\mathbb{R}^n$*,* C *be the set of components of* Π*,* $\preceq$ *be the incidence relation between them and* $C' \subseteq C$ *a subset of the components. A component* $C_m \in C'$ *is the* ***minimum component*** *of* C' *iff :*

$$(\forall C_i \in C')(C_m \preceq C_i).$$

□

This component C_m can be undefined for certain subsets of C.

The function *min_component*(C) returns the minimum element of C, or $\perp$ if such an element is undefined.

Definition 3.25 *Let* Π *be a pyramidal partition of* $\mathbb{R}^n$*,* C *be the set of components of* Π*,* $\preceq$ *be the incidence relation between them and* $R \subseteq \mathbb{R}^n$ *be a non-empty convex region. A* ***minimum covered component*** *of* Π *by* R *is* $C_m = min\_component(\{C_i \in C \mid C_i \cap R \neq \emptyset)\})$*.* □

Note that this component can be undefined for some regions.

Chapter 4

Symbolic Representations of Polyhedra

4.1 Motivation

With the rise of *Computer-Aided Design* (CAD), the need for good data structures for representing solids has grown in the past decades. The *Solid Modeling* community is devoted to develop good representations of solid objects. Another framework is *Computational Geometry*, which is more focused on solving geometrical problems, such as convex hull computation or Delaunay triangulation. The problem of defining good data structures for representing polyhedra is relevant to both approaches.

The purpose of those data structures is to be implemented into actual software libraries. Those software libraries are mainly intended to be part of CAD programs or other geometric software packages.

We motivate the need for a good representation by presenting some key operations that have to be performed on polyhedra.

A first operation consists in building incrementally a polyhedron by making combinations of simple objects, such as Boolean operations. We call this problem *incremental construction*.

Second, in many situations, detecting collisions between objects is an important problem. For such an application, being able to test whether or not a given point belongs to a given polyhedron is a key operation. This is the *point decision problem*. Moreover, it is also interesting to determine to which polyhedral component a given point belongs. For example, one could ask in which component of a cube a particular point of space is located, amongst the vertexes, edges, facets or interior of this cube. We will call this problem the *classification problem*.

Testing equality between polyhedra is another essential operation. For this operation, having a structure with an easily computable *canonical form* has a major advantage. Indeed, if a given set always have the same representation, testing equality between sets simply reduces to a syntactical comparison.

Recall that in this work we use Nef polyhedra, a definition of polyhedra that allows to have features such as dangling parts or non-manifold boundaries. In Solid Modeling or Computational Geometry, having such properties is not always the priority. When an application requires the polyhedra to have some physical reality, considering polyhedra with no volume can be problematic. This explains why a large amount of effort in order to present *regularized* operations [Req80, Män88, Hof89, Bru90], or advanced techniques to deal with inconsistencies.

4.2 State of the Art

As mentioned before, Solid Modeling is focused on developing good representations of solids. The two structures in the domain are *Constructive Solid Geometry* (CSG) and *Boundary Representation* (B-REP). As presented and explained by C. Hoffmann in [Hof89], the benefit of using one structure over the other depends on the application, and therefore, general CAD libraries often use both representations simultaneously. Other representations have also been defined. In particular, some are based on an idea of space partitioning.

Let us now have a brief view at some popular representations.

Constructive Solid Geometry (CSG) : is a binary tree with nodes representing Boolean or geometric operations and leaves being solid primitives [Req80, Hof89]. Those primitives can include, for example, spheres, cubes or any geometric object. The Boolean combination of the primitives defines a non ambiguous shape. To represent polyhedra, a CSG scheme using linear constraints as primitives is sufficient. With this representation, performing an incremental construction is trivially easy, as it only requires to build the tree corresponding to the specified combination. The point decision problem is immediate, since it can be carried out by testing the given point against the primitives and checking whether the constraint corresponding to each node is satisfied – provided that the point decision problem is easily solvable on the primitives themselves. However, we can see that the point classification problem and the canonical form become two rather difficult problems. A long justification is given in [Hof89]. Briefly, the problem lies in the fact that classifying point only over the primitives and the operations of the nodes only is not sufficient to classify the point on the represented model. Indeed, if a point lies on the boundary of two planar half-spaces, it is impossible to tell where the point must be classified inside their intersection. It depends on the respective spatial positioning of the two primitives. The result can either be a universal set, a hyperplane or a wedge. So the local shape of the sets around the point must be taken into account. A lot of elaborated rules are derived from all the possible cases and making a set of rule for only 3-dimensional set is already a sensible task. The compu-

tation of a canonical form is also not straightforward, because it requires to be able to establish a minimal set of primitives. Consider for instance a simple example that consists of a union of four 3-dimensional cubes assembled to form a bigger one, and the same set, defined directly with one primitive cube. We can easily imagine the large amount of different configurations of primitives all producing the same set. Being able to identify an exact correspondence in order to express the CSG with other primitives in a canonical way is not an easy task.

Cell Decomposition (CD) : Another possible representation for polyhedra is the *Cell Decomposition* (CD) [Req80, Mau91, Tam07]. A set of hyperplanes subdivides the space into *cells*. If the space is n-dimensional, then cells can be of any dimension between 0 and n included. A single hyperplane cuts the space into two n-dimensional cells, the hyperplane itself is a $(n-1)$-dimensional cell, the intersection of two hyperplanes can be a $(n-2)$-dimensional cell, etc... The idea behind the representation is to describe each polyhedron component by a cell or a union of cells. Each point of space is thus associated to a cell. The polyhedron is represented as an enumeration of cells. The representation of a cell takes the form of a selection function inside the set of hyperplanes. Given a set of k hyperplanes $\{h_1, h_2, \ldots, h_k\}$ and a normal vector for each of them, each point can be classified as either belonging to the hyperplane, being on one side of it or on the other side of it (referring to a normal vector). Two different points sharing the same selection function of hyperplanes will be associated to the same cell. With this representation, the point decision problem and the point classification problem are both straightforward and efficient. The iterative construction is also very easy as it only requires to adapt the selection functions associated to each class and add hyperplanes to the set. Notice that the iterative construction, to be fast, must be simple and thus is prone to be history-dependent. This implies that keeping a canonical form becomes problematic as there is a need for being able to identify and maintain a canonical set of hyperplanes [Tam07].

Binary Space Partition (BSP) : In Computational Geometry, several representation of polyhedra are based on a partitioning scheme of $\mathbb{R}^n$. Such schemes include *Binary Space Partition* [Nay90]. A BSP is a tree in which each node recursively subdivides $\mathbb{R}^n$. A BSP tree divides a set into two subsets using a hyperplane. If a point lies inside this hyperplane, the point is considered belonging to both subsets. Building a balanced tree is difficult as it requires the use of a backtracking algorithm that builds several partitioning schemes and selects the best one, or at least the one giving one of the smallest possible tree in term of depth. Indeed, having a small depth can guarantee to keep good performances for the point decision procedure. For this reason BSP is not suited for the

incremental construction problem although methods for merging BSP trees have been proposed [NAT90]. The point decision and classification problems are quite efficient, as requires only a few recursive simple tests. Computing a canonical representation is once again a difficult problem.

Boundary Representation (B-REP) : This data structure represents implicitly a solid by its boundary[Edm60]. This boundary is used to separate points of space into the interior and the exterior of a represented volume.

The representation technique for the boundary itself is quite flexible, as long as the boundary is a complete and oriented shell. This boundary is not limited to be a flat surface; it can be any kind of curved surface, such as *Non-Uniform Rational Basis Splines* (NURBS) [Pie91]. Since the boundary is used to separate points belonging to the inside and the outside of the set, a first limitation is that the represented set must have a non zero volume. A discussion about adequate methods of representing the boundaries can be found in [Wei85]. A good overview of this structure is given in [Sha02, Hof89]. The point classification problem can be sketched without fixing a particular representation for the boundary. When the membership of a point is to be assessed from the known status of another point, the number of times a path linking the two points intersects the boundary determines if both points have the same membership status. From this, it is very common to use a point projected at infinity to reduce the point decision problem to a test of intersections between the boundary and a half-line. But this procedure is only applicable to boundaries defining finite sets. Moreover, the efficiency of this kind of search is dependent to the organization of the adjacency between elements composing the boundary. Nevertheless, some studies have discussed the inherent cost of the general representation scheme as opposed to some particular implementations [NB94]. The canonical representation of the boundary leads to a canonical representation of the set itself but this unique representation of the boundary can be difficult to compute and maintain with the application of successive Boolean combinations. Indeed, a consequence of the way the representation works is that there must be a notion of valid and invalid boundaries. This problem is described in details in [Hof89]. In order to represent a valid polyhedra with a non zero volume, the boundary has to be valid, in the sense that it must be *complete* (all inside and outside points must be separated by a boundary) and the orientations of its elements have to be *collectively consistent*. For example, two adjacent faces of a cube must have an orientation that are not conflicting with each other. With this notion of validity of boundaries in mind, we can understand that representing non manifold objects can become very problematic and difficult, but not impossible [LL01]. In practice however, popular boundary representations,

because of the restrictions, use quite a large amount of case by case local solutions [Hof89] to detect and avoid problematic situations and are thus not general enough to represent arbitrary n-dimensional polyhedra.

It is worth mentioning that B-REP method probably owe a part of their popularity to the fact that they are well suited to be represented by drawing method such as *rasterisation* [Pav79].

Würzburg Structure / Selective Nef Complexes : This data structure has been proposed in [BN88] to represent Nef polyhedra. The principle is based on a decomposition of the represented polyhedron into local pyramids. There are known algorithms to manipulate this data structure but those are not very efficient in practice. Some variants of the structure such as, for example, the addition of incidence information, have been studied later to overcome this issue and have improved efficiency, but even those variants remained impractical in certain situations [GHH+03].

Over the years, more usable data structures have been derived and studied from those results in a more restricted context of three dimensional polyhedra. One of the latest is presented in [GHH+03] along with a brief history of previous work leading to this idea. The limitation to three dimensional polyhedra is a consequence of the use of ad-hoc sub data structures to represent vertexes and higher dimensional polyhedral components. Indeed, this latter representation uses a *Sphere Map* (called *local graphs* in [DMY93]) to represent local pyramids of each vertex of the polyhedron and subsequent local pyramids are represented by the use of a data structure called *Selective Nef-complexes (SNC)*. A Sphere Map is a virtual arbitrary small sphere centered on a vertex of the represented polyhedron that intersects all polyhedral components incident to the vertex.

A SNC is a decomposition of the space into *cells* by the use of planes and is basically the identical to the previously presented cell decomposition. The difference lies in the selection function that associates arbitrary labels to each cell instead of just selecting or excluding cells from the set. Cells are of multiple nature and each type of cell is represented differently. There can be edges, facets, volumes and some more information to link those elements with respect to the incidence between them. This later information is useful to guarantee a certain efficiency of the manipulation algorithms, although the point decision problem is quadratic in the size of the polyhedron [Hac07].

This representation is by nature not suited for representing general n-dimensional polyhedra. The lack of a unified way of representing pyramids of any dimension with one single structure does prevent any attempt at using Sphere Maps in a more general n-dimensional space.

4.3 Real Vector Automata

Another approach to represent symbolically polyhedra is to build a finite automaton on infinite words that recognizes encodings of points. This structure is called the *Real Vector Automaton* (RVA) [BBR97, BJW05].

4.3.1 Encoding Vectors

Since automata are finite state machines recognizing words, one must define a mapping between points of $\mathbb{R}^n$ onto words over some finite *alphabet*.

The encoding scheme is based on the positional numeration system, using symbols in the alphabet $\Sigma_r = \{0, \ldots, r-1\}$, with $r \in \mathbb{N}_{>0}$ being a chosen *base*. An additional symbol $\star$ is used to separate the integral part from the fractional part of encodings.

In a given base r, a number $z \in \mathbb{R}_{\geq 0}$ is encoded by infinite words of the form :

$$a_{p-1}a_{p-2}\ldots a_0 \star a_{-1}a_{-2}\ldots$$

such that $p > 0$ is the length of the integer part, $a_i \in \Sigma_r$ for all $i < p$ and $z = \sum_{i<p} a_i r^i$. Negative numbers are encoded by their r's complement : to encode a number $z \in \mathbb{R}_{<0}$, we just encode $z + r^p$ where p is the length of its integer part. The number of symbols of the integer part must be chosen so that $-r^{p-1} \leq z \leq r^{p-1}$.

This results in having 0 as the leading symbol of encodings of positive numbers (or zero) and $(r-1)$ for negative numbers. Moreover, every number admits infinitely many encodings as this leading symbol can be repeated at will.

Furthermore, repeated symbols corresponding to the sign of the number asside, some real numbers admit two different encodings. Such numbers are said to admit *dual encodings*. For instance, the real number $\frac{3}{10}$, when encoded in base 10, admits the two following families of encodings :

$$0^+ \star 3(0)^\omega \text{ and } 0^+ \star 2(9)^\omega$$

RVA requires that the encodings of a point are either all accepted or all rejected. This question of dual encodings is not the object of this work, but we mention that having this requirement introduces some artifacts in the data structure. It is worth mentioning that a study has been conducted in order to avoid those dual encodings [EK08] and that a complete study of this problem is made in [Bru11].

In order to keep this presentation simple, we will not consider further the special cases associated with dual encodings in RVA.

Example 4.1 *The encodings of* 1 *in base* 2 *form the language* $0^+1 \star 0^\omega$
The encodings of $-\frac{69}{8}$ *in base* 3 *form the language* $2^+00 \star (10)^\omega$ ◇

To encode a point $\vec{z} = (z_1, z_2, \ldots, z_n) \in \mathbb{R}^n$ in a base r, one encodes each component z_i with an identical length p for their integer parts such that :

$$(\forall i \in \{1, 2, \ldots, n\})(-r^{p-1} \leq z_i \leq r^{p-1})$$

The encodings of all components can then be merged into a single infinite word over the alphabet $\{1, \ldots, r-1\}^n \cup \{\star\}$ by synchronously reading their encodings one symbol at a time. Since the separator symbol $\star$ is read only once and at the same time in all encodings of the components, the tuple ($\star$, $\star$, ..., $\star$) can be replaced by a single occurrence of $\star$. This method of encoding vectors is called *synchronous encoding*.

The main disadvantage of this method is its intrinsic exponential explosion of the size of the alphabet with respect to the number of dimensions. It is possible to address this problem by *serializing* the encodings. Instead of reading one symbol $(\sigma_1, \sigma_2, \ldots, \sigma_n) \in \{0, 1, \ldots, r-1\}^n$, one reads the word $\sigma_1\sigma_2 \ldots \sigma_n$ expressed over $\{0, 1, \ldots, r-1\}$. This method is called *serialized encoding*.

Example 4.2 *The word*

$$\begin{pmatrix} 0 \\ 9 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 9 \\ 0 \end{pmatrix} \begin{pmatrix} 8 \\ 9 \\ 0 \end{pmatrix} \star \begin{pmatrix} 0 \\ 3 \\ 3 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 3 \end{pmatrix}^{\omega}$$

encodes the vector $(18, -0.7, 1/3)$ *in base* 10.

The serialized encoding of the same vector, in base 10*, is*

$$090190890 \star 033(003)^{\omega}.$$

◇

From the prefix of a synchronized encoding of points, it is possible to characterize the set of points that admit an encoding thet shares this particular prefix :

Definition 4.3 *Let* $r, n \in \mathbb{N}$ *be a base and a dimension, and* $u \star v$ *be a word with* $u \in \{0, \ldots, r-1\}^+$*, and* $v \in \{0, \ldots, r-1\}^*$.

If $\vec{x}$ *is the point encoded by* $u \star w\vec{0}^{\omega}$ *in base* r*, then the hypercube*

$$H_{u\star v} = \vec{x} + [0, \frac{1}{r^{|v|}}]^n$$

contains exactly all the points that admit an encoding prefixed by $u \star v$. □

This definition is limited to synchronized encodings, it can be adapted to serial encodings, by keeping the length of w a multiple of n. For the case where w is not a multiple of n, the set of points sharing this prefix for their encodings is not an hypercube, but an orthotope[1].

Also, we only consider prefixes containing the symbol $\star$ to only deal with defined integer parts which always bound a region of space.

[1] A set defined as the Cartesian product of intervals

4.3.2 Syntax

Definition 4.4 *A **Real Vector Automaton** representing a set V of vectors in $\mathbb{R}^n$ in base r is a Büchi automaton that recognizes the language of all the encodings of elements of V.* □

The expressiveness of RVA is large enough to represent all sets definable in $\langle \mathbb{R}, \mathbb{Z}, +, \leq \rangle$, the *first-order additive theory of real and integer numbers* [BBR97]. As polyhedra are definable in $\langle \mathbb{R}, \mathbb{Z}, +, \leq \rangle$, every polyhedron can be represented by a RVA.

It has been established that it is possible to restrict the syntax of RVA to *weak deterministic RVA*, a subclass of general RVA. This class corresponds to the set of all RVA such that every state belonging to a given strongly connected component (SCC) has the same accepting status. An interesting fact is that this restricted class is still expressive enough to represent any set definable in $\langle \mathbb{R}, \mathbb{Z}, +, \leq \rangle$ [BJW05].

The weak deterministic RVA have several practical advantages over general RVA, but mainly they are more easily manipulated.

In particular, computing Boolean combinations of their represented sets, projecting or computing Cartesian products between them becomes feasible in practical applications. Another very interesting fact is that they admit an easily computable canonical form as explained in [Löd01]. Performing those operations on general RVA is feasible, but more difficult, as discussed in [Var07] for weak deterministic Büchi automata versus general Büchi automata.

From now on in this work, we use the term RVA in place of weak deterministic RVA.

Example 4.5 *Figure 4.1 shows a RVA accepting synchronized encodings of points $(x, y) \in \mathbb{R}^2$ satisfying $(x = \frac{1}{3} \wedge y \geq \frac{1}{3} \wedge y \leq \frac{2}{3})$.* ◇

4.3.3 Point Decision with RVA

As discussed before, each word read by a RVA is infinite and its acceptance condition is based on the fact that the reading of this word ends in an infinite cycle composed of accepting states. This infinite cycle is, by essence, located inside a strongly connected component (SCC) of the RVA.

Consider $\mathcal{A}$, a deterministic weak RVA representing a polyhedron $\pi \subseteq \mathbb{R}^n$ in a base $r \in \mathbb{N}_{>0}$.

In order to check whether a point p belongs to the polyhedron represented by $\mathcal{A}$, a simple solution consists in generating an encoding e of $\vec{p}$ in base r. The next step is to check if e is accepted by $\mathcal{A}$. As $\mathcal{A}$ is deterministic, at most one path of the automaton corresponds to the reading of e. This path, if any, will inevitably end in one SCC of $\mathcal{A}$. If the states of this SCC have an accepting status, e is accepted by $\mathcal{A}$, otherwise, it is not.

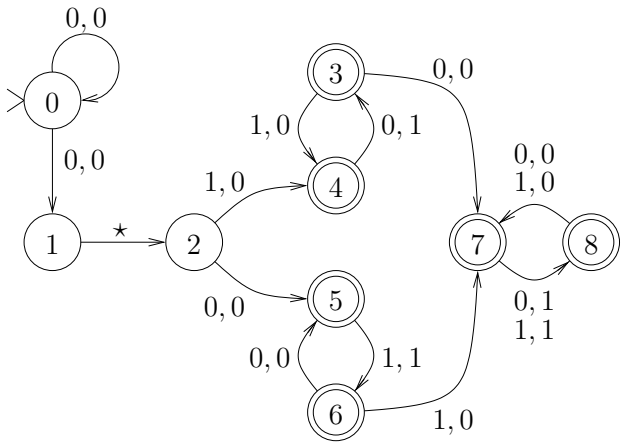


Figure 4.1: Example of synchronized RVA representing the set $\{(x, y) \in \mathbb{R}^2 \mid (x = \frac{1}{3} \wedge y \geq \frac{1}{3} \wedge y \leq \frac{2}{3})\}$.

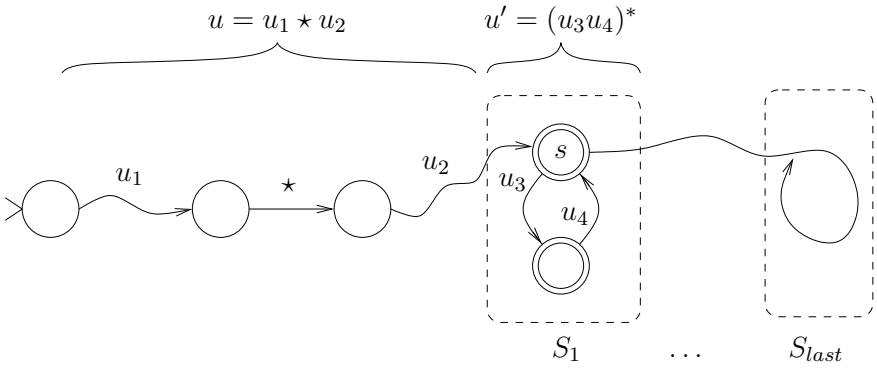


Figure 4.2: A pause in the point decision procedure of RVA.

It has been shown in [BBL09, BBD10, Bru11] that the SCC of a RVA representing a polyhedron are related to the components of this polyhedron. If certain situations are carefully avoided, then each SCC can be associated to a polyhedral component. This can be explained intuitively as follows.

First, we can make a general observation. When one checks whether a word is accepted by a RVA and pauses after having read an arbitrary prefix of this word, the only memory of the run kept so far is the current reached state. One still has to check if the remaining suffix is accepted from that state. It is clear that the outcome of this latter test does only depend on this state and this suffix to read.

Now, consider again the procedure that tests whether a point $\vec{p}$ (encoded in a base r by the infinite word e) belongs to a polyhedron $\pi \subseteq \mathbb{R}^n$ represented by the RVA $\mathcal{A}$. Let us pause after having read a finite prefix $u \in \Sigma_r^+ \star \Sigma_r^*$ of e that leads to a state s which, without loss of generality, belongs to a non trivial SCC S_1 (see Figure 4.2).

The prefix u characterizes the set H_u of the points prefixed by u. Since

the possible infinite suffixes accepted from s only depend on s, we can deduce that the shape of π inside H_u does also only depend on s. Remember that s is in S_1, a SCC of $\mathcal{A}$, it implies that it is possible to find points of $\mathbb{R}^n$ such that they admit an encoding of the form uu' such that reading uu' also end in s. Finding such points is indeed feasible as it reduces to read u and then follow an arbitrary number of cycles inside S_1 from s to s. As the shape of π inside $H_{uu'}$ does also only depend on s, we conclude that the set of points whose encoding suffixes are accepted from s is the same inside H_u as in $H_{uu'}$, up to scaling. This has been established in [BBL09], and the invariance of the shape of π inside H_u and $H_{uu'}$ actually holds for arbitrary scaling factors of π, wich implies that the sets represented by states of SCC are pyramidal.

A path generally visits several different SCC before cycling infinitely into a final one. It depicts the following procedure : in order to decide whether to accept or not the encoding of a point $\vec{p} \in \mathbb{R}^n$, the RVA first chooses deterministically a polyhedral component C_1 of π in the vicinity of which this decision can be carried out. This component is represented by the first SCC that the encoding of the point visits after reading a prefix u_1, and the vicinity is given by H_{u_1}. Then the RVA checks whether of not the point $\vec{p}$ belongs to C_1, which amounts to checking if the encoding of $\vec{p}$ stays forever inside the SCC. In the affirmative, the procedure concludes that the point belongs to π. If the negative, the RVA chooses a component C_2 incident to C_1 and the same procedure is repeated from there. It means another prefix u_2 is read until reaching the next SCC in the RVA, and this procedure repeats until a final SCC is encountered.

From a different perspective, the principle of the point decision procedure carried out by RVA associate a SCC (which corresponds to polyhedral component) to every point of space in the vicinity of which the decision can be made and performs such an association recursively. This means that when a SCC is reached, if the tested point does not belong to the polyhedral component represented by this SCC, the search will continue further. An interesting fact is that we are already sure that the point belongs to a polyhedral component incident to the one chosen.

Example 4.6 *Figure 4.1 shows a RVA having three different non trivial SCC in its fractional part. Those SCC are* $S_1 = \{3, 4\}$, $S_2 = \{5, 6\}$ *and* $S_3 = \{7, 8\}$. *Following cycles in* S_1 *corresponds to reading the point* $(2/3, 1/3)$. *Following cycles in* S_2 *corresponds to recognizing the point* $(1/3, 1/3)$ *and following cycles in* S_3 *corresponds to reading the points belonging to the open segment* $((1/3, 1/3)(2/3, 1/3))$. *In this example, we see that the three polyhedral components corresponding to the segment and its two extremities, are each associated with SCC of the RVA. Notice that the fourth polyhedral component, the empty set, is not shown in the RVA, but is also associated to a SCC of the RVA, as it corresponds to a non accepting cycle.* ◇

4.3.4 Limitations of RVA

Although RVA are expressive enough to represent arbitrary polyhedra and have efficient algorithms to manipulate them, they have some major drawbacks. The size of a RVA representing the set of solutions of a linear constraint $\vec{a}\vec{x}\#b$ with $\vec{a} \in \mathbb{Z}^n, b \in \mathbb{Z}$ and $\# \in \{\leq, <, =, >, \geq\}$ grows logarithmically with the value of b but linearly with the values of the components of $\vec{a}$ [BRW98]. Representing a polyhedron by combining several RVA representing a linear constraint, will most of the time only worsen the situation.

Another weakness of the data structure is that its size grows exponentially with the number of dimensions.

In this work, we tackle the first drawback by presenting a new data structure that inherits the qualities of RVA but avoids the size blowup of the representation of linear constraints.

Chapter 5

Implicit Real Vector Automata

The main purpose of the *Implicit Real Vector Automaton* (IRVA) is to provide a new data structure for representing the components of a polyhedron and the incidence relation between them, that inherits the strengths of RVA such as an efficient point decision procedure and large expressive power, but represents polyhedral components more concisely. Another advantage is that it admits an easy computable canonical form.

5.1 Principles of IRVA

We now introduce the main principles of IRVA, first intuitively by defining an abstract data structure, and then more formally. For clarity sake, we address the problem of representing polyhedra as opposed to the more general case of polyhedral partitions (as introduced in Section 3.2), that will be discussed later.

We have seen in Chapter 3 that a polyhedron partitions the space into polyhedral components. Moreover, every such component associates a single local pyramid to all of its points (see Definition 3.11). Since the set of apexes of a pyramid forms an affine space [BN88], the affine closure of a polyhedral component is the set of apexes of the local pyramid associated to the points belonging to this component.

Example 5.1 *Figure 5.1(a) shows a polyhedron $\pi \subset \mathbb{R}^2$ which is the union of a triangle and an isolated point. This polyhedron can be decomposed into as set of polyhedral components, each corresponding to a specific local pyramid. Figure 5.1(b) shows such a decomposition.* ◇

We naturally obtain the following decomposition scheme for polyhedra : every polyhedron decomposes the space into a set of polyhedral components, hence a set of local pyramids. There exists a partial order between polyhedral components : the incidence relation (see Section 3.1.2).

From these principles, let us sketch an abstract data structure for representing polyhedra. This data structure is a graph containing *implicit states*

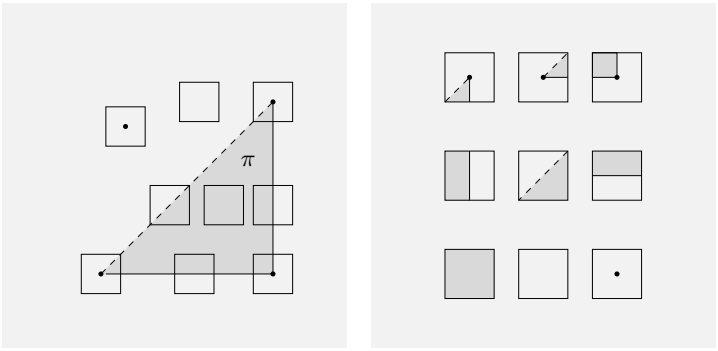


Figure 5.1: Example of (a) a polyhedron $\pi \subset \mathbb{R}^2$ and (b) the set of polyhedral components and local pyramids.

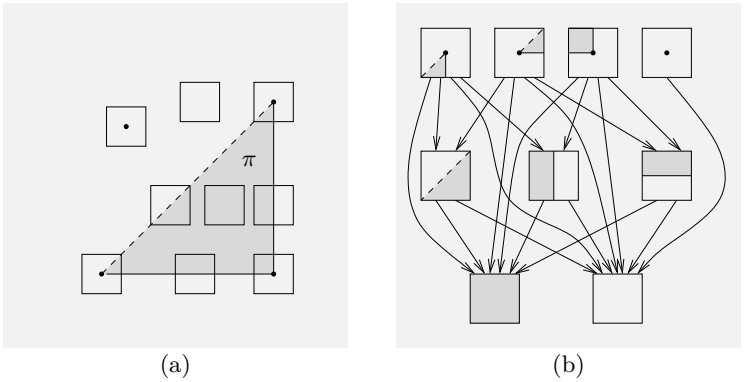


Figure 5.2: Example of (a) a polyhedron π and (b) an abstract structure representing π by implicit states and a transition function.

linked by an acyclic transition function. An implicit state represents a polyhedral component of the polyhedron and the transition function represents the incidence relation between those components. A polarity is associated to each implicit state in order to represent the fact that a polyhedral component belongs or not to the represented polyhedron.

An example of such a decomposition, augmented with incidence information, is given in Figure 5.2(b).

From Chapter 4, where valuable insight was obtained on the way polyhedra are represented inside RVA, we can see our abstract data structure as a very simplified RVA. Implicit states correspond to non trivial SCC of the RVA and the transition relation represents the transitions between them.

We have also seen in Section 4.3.3 how RVA make it possible to solve the point decision problem. We now consider the high-level steps of the procedure solving the point decision problem on RVA and see how those steps could be translated in order to build such a procedure on our proposed data structure.

The point decision procedure consists in executing these steps when deciding whether a point p belongs to the represented polyhedron.

Point Decision Procedure 1

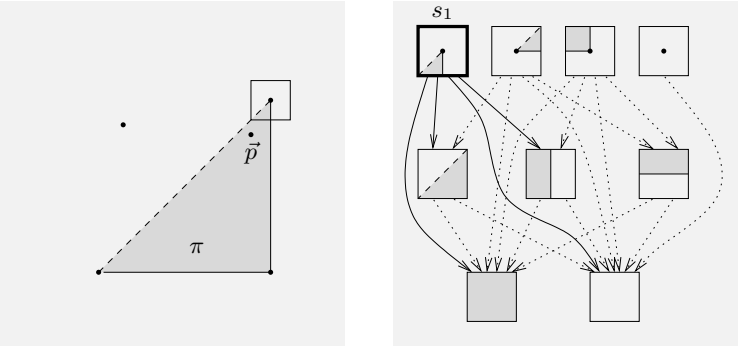
Step 1 : The encoding of p visits a first non trivial SCC inside the RVA. On our structure, this corresponds to selecting an initial implicit state.

Step 2 : If the encoding of p does not leave the current SCC, then the procedure terminates. p is accepted if the SCC is composed of accepting states, or rejected if the SCC is composed of non accepting states. In our structure this test can be achieved by algebraic means. The exact nature of this test will be addressed later. p is accepted or rejected depending on the polarity of the implicit state.

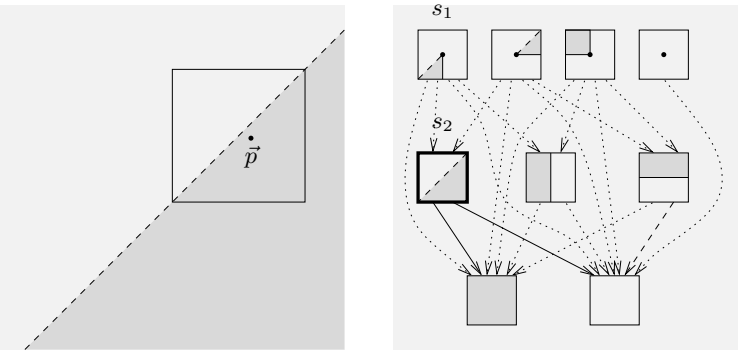
Step 3 : If p does not belong to the current polyhedral component, a path corresponding to an encoding of p is followed inside the RVA until reaching another non trivial SCC. The procedure returns to Step 2 from this implicit state. Termination is guaranteed because RVA is finite-state, hence a final SCC will always be reached. In our structure, we have to perform some kind of decision by following the transition relation until reaching another implicit state and then return to Step 2 from this state, too. Once again the procedure is guaranteed to terminate since it will eventually end up in states of the automaton that correspond to universal or empty polyhedral components.

Example 5.2 *Let us consider again the polyhedron π of Example 5.1. We describe a simple execution of the point decision procedure with the point p.*

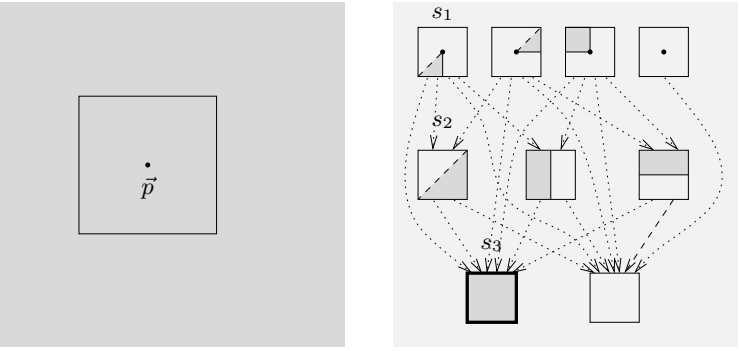
First, the initial state is s_1 selected, by a mechanism not discussed here. Intuitively, it appears clear that the point decision procedure can be carried out from s_1. This implies that only considering the pyramid associated to s_1 is sufficient for the procedure to arrive at a right conclusion. State s_1 represents a vertex of π. Since p does not belong to the affine closure of this vertex, the procedure continues by navigating into the transition structure outgoing from s_1. Potential transitions are shown in the next illustration by solid arrows.



Then, the next state s_2 is reached. Once again, it appears clear that the procedure can effectively continue from s_2, by only navigating inside its associated pyramid. Notice that state s_2 represents an edge of π. Since p does not belong to the affine closure of this edge, the procedure continues further by moving into the transition relation from s_1.



Finally the state s_3 is reached and the procedure terminates there because p belongs to is associated polyhedral component (in the present case, the universal set). Since the polarity of s_3 is in*, p is accepted, which concludes that it belongs to π.*



$\diamond$

Conceptually, the structure that we have just sketched has some nice properties as far as the point decision procedure is concerned : the number of visited implicit states is small and the procedure is deterministic. From an operational point of view, however, it raises three questions :

Question 1 : How is the initial implicit state chosen ?

Question 2 : What information could be associated to implicit states ?

Question 3 : What is the nature of the transition function ?

Question 1 will be addressed in Section 5.1.1 and Questions 2 and 3 will be addressed respectively into Sections 5.1.2 and 5.1.3.

5.1.1 Selecting an Initial State

In the previous section, we have seen that in order to test whether a point belongs or not to the polyhedron represented by our abstract data structure, the first step consists in choosing an initial implicit state from which to start the exploration.

An obvious observation is that this choice becomes trivial when there is only one initial implicit state. Also, having exactly one initial implicit state, correspond to the case of a polyhedron that has a pyramidal structure.

We show that any non pyramidal polyhedron can be transformed without loss of generality into a *representing pyramid*. The representing pyramid of a polyhedron is defined in an higher dimensional space.

Definition 5.3 *Let $n \in \mathbb{N}$ be a dimension and $\pi \subseteq \mathbb{R}^n$ be a polyhedron. The* ***representing pyramid*** *of π is the polyhedron*

$$\pi' \subseteq \mathbb{R}^{n+1} = \{\lambda(x_1, \ldots, x_n, 1) \mid \lambda \in \mathbb{R}_{>0} \wedge (x_1, \ldots, x_n) \in \pi\}$$

□

This definition implies that $\vec{0}$ is always an apex of every representing pyramid.

Applying elementary operations over polyhedra, such as computing Boolean combinations, testing equality or inclusion, and solving the point decision and point classification problems, can be straightforwardly translated into similar operations over their representing pyramids.

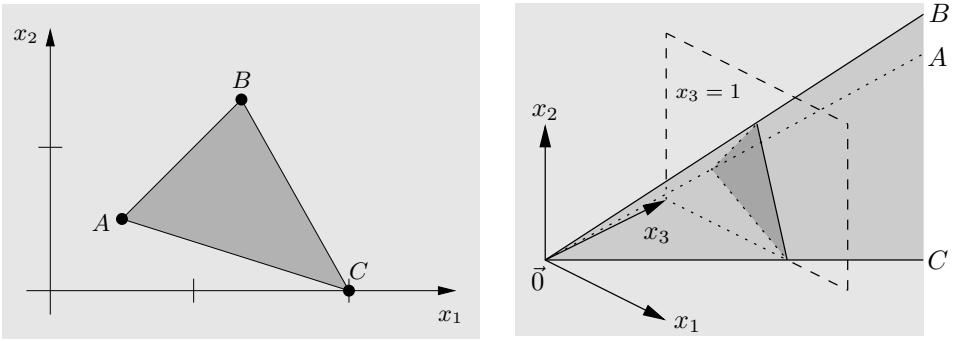
The inverse operation, which is the retrieval of the represented polyhedron $\pi \in \mathbb{R}^n$ from its representing pyramid $\pi' \in \mathbb{R}^{n+1}$ is the calculation of the set

$$\pi = \{(x_1, \ldots, x_n) \mid (x_1, \ldots, x_n, 1) \in \pi'\}$$

and it corresponds to the intersection of π' with the plane $x_{n+1} = 1$.

So, in summary, the test that establishes if $\vec{p}$ belongs to π can be reduced to testing whether the point $(\vec{p}[1], \vec{p}[2], \ldots, \vec{p}[n], 1)$ belongs to the representing pyramid of π.

As a consequence, from now on, we will consider without loss of generality that the polyhedra we represent are pyramidal with apex $\vec{0}$.



(a) (b)

Figure 5.3: (a) Triangle with vertexes A, B and C of Example 5.4 (b) The representing pyramid of this triangle.

Example 5.4 *Figure 5.3(a) shows a 2–dimensional polyhedron, formed by three vertexes* $A = (1/2, 1/2)$, $B = (4/3, 4/3)$ *and* $C = (2, 0)$ *delimiting a closed filled triangle.*

The representing pyramid of this polyhedron is defined as :

$$-x_1 + x_2 \leq 0 \wedge x_1 + 3x_2 - 2x_3 \geq 0 \wedge 2x_1 + x_2 - 4x_3 \leq 0$$

and is shown in Figure 5.3(b) (note that it is a subset of $\mathbb{R}^3$*). The plane* $x_3 = 1$ *is depicted in the figure to emphasize the construction of the representing pyramid and also decide how the original triangle can be retrieved from its represented pyramid. The pyramid admits the apex* $\vec{0}$ *and is built upon three rays A, B and C, which are the extensions of the original vertexes A, B and C. The directions of those rays are* $A = (1/2, 1/2, 1)$, $B = (4/3, 4/3, 1)$ *and* $C = (2, 0, 1)$. *The three edges of the triangle AB, BC and AC become respectively the three 2–faces AB, BC and AC of the pyramid. The interior of the triangle is a 3–space.* ◇

Representing only pyramids with apex $\vec{0}$, besides of eliminating the need for a particular first step in the point decision procedure, simplifies substantially the represented polyhedra : the incidence relation always has a unique initial polyhedral component, to which all other components are incident (cf.

Theorem 3.23). This implies that, although $\vec{0}$ is not necessarily a component of the represented pyramid, every pyramid admits $\vec{0}$ as apex and thus every component of every non-empty pyramid contains points that are arbitrarily close to $\vec{0}$. As a consequence, all those components are pyramids centered on $\vec{0}$. This implies that the affine closure of any component is a vector space.

Representing only pyramids renders the first step of procedure 1 trivial, and thus answers our first question.

5.1.2 Definition of Implicit States

In this section, we study the exact nature of the information that needs to be associated to implicit states.

As explained before, our abstract data structure is composed of an acyclic graph of *implicit states* linked together by a transition relation. Each implicit state represents a pyramid, the set of apexes of which forms a vector space.

Let us consider an implicit state s representing a polyhedral component C representing a pyramid ρ. Since all polyhedral components of ρ have to be represented by implicit states reachable from s, it is sufficient to associate s with a representation of the affine closure of C, which has been shown to be a vector space. In addition, one must also store a polarity for representing the fact that points of C belongs to ρ or not. With this scheme, the characterization of the pyramid represented from s is distributed among the implicit states reachable from s, as well as inside the transition function linking s to its successors.

Let us come back to the point decision procedure. The test that establishes whether a point p belongs or not to the polyhedral component associated to s reduces to testing whether p belongs to the vector space associated to s. If p belongs to the vector space, then the procedure terminates and p is accepted if and only if the polarity associated to s is accepting. If the polarity is non accepting.

From an operational point of view, we represent vector spaces by associating a vector basis to each implicit state, along with its polarity. This vector basis is a generator of the vector space corresponding to the affine closure of the polyhedral component associated to this implicit state.

In summary, the test carried out in the second step of Procedure 1 reduces to testing whether a point belongs or not to a vector space. We have answered the second question.

5.1.3 Nature of the Transition Function

In Step 3 of Procedure 1, since p does not belong to the vector space of s, following the transition relation must induce a decision by selecting the next implicit state to visit. From now on, we will call the transition relation originating from s the *outgoing decision structure* from s.

Since the set represented from s is a pyramid, only the direction of p with respect to the vector space of s is relevant. By Definitions 2.8 and 3.3, we know that if a point belongs to a pyramid, all scalings of this point around one of its apexes also belong to that pyramid.

Operationally, we define an encoding scheme for directions in order to advance in the decision structure. This encoding scheme, in order to be practical, has to rely on a finite alphabet. This calls for a second type of states inside IRVA, which are called *explicit states*. Taking a decision inside the decision structure is made by reading a word from an implicit state s_1, visiting only explicit states and arrive in another implicit state s_2.

Definition 5.5 *A **decision path** of an IRVA is a sequence of the form :*

$$s_1 \xrightarrow{\sigma_1} s_2 \xrightarrow{\sigma_2} \dots \xrightarrow{\sigma_{k-1}} s_k$$

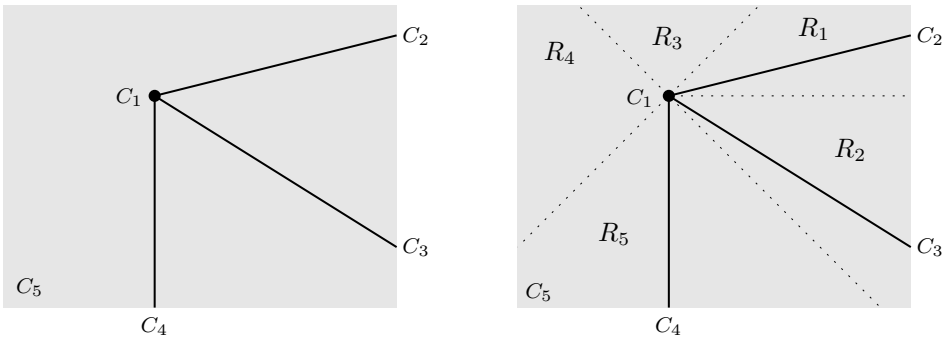
where s_1 and s_k are implicit states, each s_i with $1 < i < k$ is an explicit state and σ_j for $0 < j < k$ are symbols. □

Since space is dense, there are infinitely and even uncountably many possible directions. Recognizing each of them individually would require an infinite structure, which is of course impossible. Instead, the decision structure regroups together directions that do not need to be distinguished. These groups of directions have to be small enough to unambiguously identify the next polyhedral component that needs to be visited by for the point decision procedure.

So, as only directions are considered, every label in the decision structure corresponds to a region of $\mathbb{R}^n$ that is pyramidal and composed of points having a direction whose encoding is prefixed by that label. Each subsequent reading of a symbol refines this region further.

Moreover, since all the possible directions are to be handled by the decision structure, words labeling paths inside this decision structure from s describe regions that have to cover all the possible directions from C. From now on we will call such regions *path regions*.

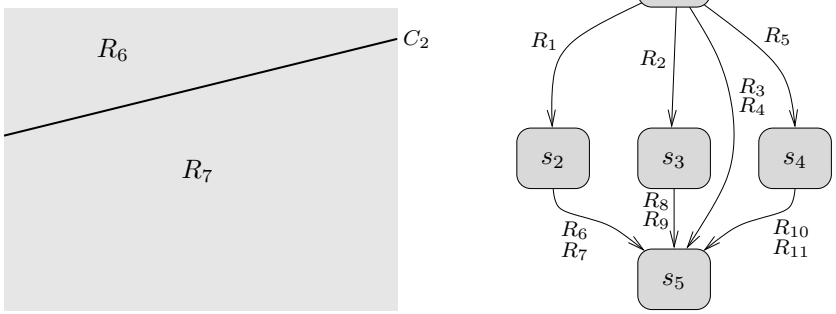
Example 5.6 *Figure 5.4 (a) shows, a pyramidal 2-dimensional polyhedron π, representing three half-lines C_2, C_3 and C_4 all incident to a point $C_1 = (0, 0)$. C_5 is the exterior of the polyhedron. Figure 5.4 (b) shows five path regions of $\mathbb{R}^2$, that corresponds to groups of directions, that are labeled R_1, R_2, R_3, R_4 and R_5 and are delimited by dotted lines. Figure 5.5(b) sketches the structure of an IRVA representing this set. In this sketch, rounded boxes are implicit states $\{s_1, s_2, s_3, s_4, s_5\}$. Vector spaces and polarities associated to implicit states are not shown. The implicit states represent respectively the pyramids having for set of apexes the components C_1, C_2, C_3, C_4 and C_5. The decision procedure is not represented explicitly here, only the regions induced by the outgoing decision structure from the implicit state s_1 are represented as transition labels. For example, the decision path labeled R_2 has s_3 as destination, so we know that*



(a) (b)

Figure 5.4: (a) A polyhedron $\subset \mathbb{R}^2$ and (b) the set of path regions $\{R_1, R_2, R_3, R_4, R_5\}$ from Example 5.6.

from s3 is represented $P_\pi(R_2)$, which is the coinciding pyramid with π over R_2. The regions identified by R_6 and R_7 are shown in Figure 5.5(a). Regions R_8, R_9, R_{10} and R_{11} are not shown, but are easy to deduce. ◇



(a) (b)

Figure 5.5: (a) The polyhedral component C_2 and the set of path regions $\{R_6, R_7\}$ from Example 5.6. (b) The structure representing the set given in Figure 5.4.

Without having defined precisely the actual encoding scheme for directions, we have answered our third question by explaining at a high level the operation of the transition structure.

Technical details aside, we now have a complete view of our proposed data structure.

5.1.4 A Second Look at the Point Decision Procedure

We have mentioned that implicit states are linked by a decision structure that recognizes encodings of directions. Before explicitly explaining this encoding, we now study how the incidence relation is represented inside our abstract data structure.

As it has already been explained, when a polyhedron π is represented, each of its polyhedral component corresponds to an implicit state. Let us consider a particular implicit state s representing a polyhedral component $C \subset \mathbb{R}^n$. Recall that the paths that can be followed from s correspond to the local pyramid $P_\pi(C)$. Moreover, each of these paths corresponds to a cover region of space.

Let us consider such a path outgoing from s. Its destination of can either be an implicit or an explicit state. If it is an explicit state, then we can follow more transitions and extend the path further. If, on the other hand, the destination is an implicit state, then it means that the path identifies a path region in which $P_\pi(C)$ is pyramidal. This pyramid has the affine closure of C_2, a polyhedral component of π incident to C, as set of apexes. In fact, the component C_2 is the minimum covered component of $P_\pi(C)$ by this region, see Definition 3.25.

In other words, when trying to establish whether a point belongs or not to the local pyramid $P_\pi(C)$, all the points belonging to the path region defined by the labels of the path α leaving s, will belong to one of the components of $P_{P_\pi(C)}(C_2)$, hence the implicit state s_2, that corresponds to C_2 will be marked as destination for α.

Let us now see what the point decision procedure becomes with respect to our structure.

Recall that the problem is to decide whether of not a point $p \in \mathbb{R}^n$ belongs or not to a represented polyhedron.

Point Decision Procedure 2

Step 1 : Set the initial implicit state as current state.

Step 2 : If p belongs to the vector space associated to the current state, terminate and return the polarity associated to the current state.

Step 3 : From the current state, follow a path inside the decision structure leading to another implicit state. This path must identify a region containing p. Set this new implicit state as current state and go to step 2.

The procedure terminates when it reaches an implicit state associated to a vector space that contains p. At worse, since an implicit state associated to $\mathbb{R}^n$ always exists, it will eventually be reached.

The central operation in Step 2 is to find a path inside the decision structure

outgoing from a state s in order to identify the next implicit state to visit. To know which path can be followed inside the decision structure, one first computes a *direction* in which the point can be reached from points inside the vector space associated to s, then the resulting direction is *encoded* by an infinite word over a finite alphabet and this word is followed until an implicit state is reached.

Generalization to pyramidal partitions

We have so far considered that our abstract data structure represents a pyramid. In fact, this structure can serve as symbolic representation of pyramidal partitions of space.

Indeed, the previous presentation is easy to generalize to pyramidal partitions, simply by generalizing the concept of polarity to the more expressive notion of color as explained in Section 3.2.

In other words, in the previous presentation of the data structure, the concept of represented pyramid is straightforwardly valid for any chosen color.

5.2 Definition of IRVA and Syntax

In Section 5.1, we have sketched an abstract data structure for representing polyhedra. This structure is composed of implicit states that represent polyhedral components, along with a transition function that describes the incidence relation between them. Several questions had to be addressed, which we have answered by presenting design choices.

We will now see how these design choices are organized inside the IRVA data structure.

Section 5.2.1 will present the implicit states inside IRVA. Section 5.2.3 will define the encoding scheme used by IRVA in order to recognize directions. Before that, we define precisely what we call direction in Section 5.2.2.

5.2.1 Implicit States

The implicit states in IRVA are exactly the same as presented in Section 5.1.2.

In summary, each implicit state represents a polyhedral component. A vector space equal to the affine closure of the represented polyhedral component is associated to each implicit state. This vector space is the set of apexes of the local pyramid associated to the points inside the polyhedral component.

We associate a vector basis and a polarity (a color) to each implicit state. This vector basis is a generator of the vector space associated to the implicit state.

5.2.2 Computing Directions

In Step 2 of point decision Procedure 2, we conclude that $\vec{p}$ does not belong to the vector space of the current implicit state s. It is equivalent to say that $\vec{p}$ does not belong to the set of apexes A of the pyramid represented from s. The decision structure leaving s associates $\vec{p}$ with one or several decision paths that lead to another implicit state. As mentioned before, each prefix read from s corresponds to a pyramidal region of space. Furthermore, this pyramid has A as set of apexes. This implies that the decision structure considers as equivalent two vectors $\vec{u}$ and $\vec{v}$ such that $\vec{v} = \lambda(\vec{u}-\vec{x})$ with $\lambda \in \mathbb{R}_{>0}$ and $\vec{x} \in A$.

This last property holds two important aspects. First, the direction of p does not depend on its magnitude. Second, this direction is independent of any translation inside A.

Therefore, in order for the structure to present this property, one expresses $\vec{p}$ in another lower dimensional vector space, such that it is *complementary* to A and then *normalizes* the result. Notice that A is equal to the affine closure of the vector space associated to s.

Definition 5.7 *Let $n \in \mathbb{N}$ be a dimension and V be a vector space of $\mathbb{R}^n$. A vector space $W \subseteq \mathbb{R}^n$ is **complementary** to V if $V \cap W = \{\vec{0}\}$ and every vector $\vec{p} \in \mathbb{R}^n$ can be expressed as $\vec{p} = \vec{v} + \vec{w}$, with $\vec{v} \in V$ and $\vec{w} \in W$.* □

Since the decision structure fully covers all space and $\vec{0}$ always belongs to the set of apexes of any path region[1], we know that if a point $\vec{p}$ belongs to a path region, then $\lambda\vec{p}$ (with $\lambda \in \mathbb{R}_{>0}$) also belongs to that region.

We obtain :

$$(\vec{p} \in P) \Leftrightarrow (\lambda(\vec{p} - \vec{x}) \in P)$$

with $\lambda \in \mathbb{R}_{>0}$ and $\vec{x} \in A$.

To summarize, if a point belongs to a path region R, then any uniform scaling of this point centered on any apex of R does also belong to the path region.

In order to define an encoding of $\vec{p}$, the decision structure leaving an implicit state s needs only to consider directions inside a vector space W, complementary to the vector space V associated to s. Let $\{\vec{v}_1, \ldots, \vec{v}_m\}$ be a basis of V.

When $m = n$, no decision structure leaves s, since $V = \mathbb{R}^n$ hence having $\vec{p} \notin \mathbb{R}^n$ is impossible.

On the contrary, when $m < n$, a variable change operation is applied i.e. the point $\vec{p}$ is expressed in another coordinates system

$$\mathcal{S} = \{\vec{v}_1, \ldots, \vec{v}_m, \vec{w}_1, \ldots, \vec{w}_{n-m}\}$$

where $\{\vec{w}_1, \ldots, \vec{w}_{m-n}\}$ is the *canonical basis* of the complementary vector space of V.

[1] A path region is pyramidal with respect to a set of apexes equal to a vector space associated to an implicit state.

Definition 5.8 *The **canonical basis** of the complementary vector space of a vector space V, noted* $\mathrm{CVS}(V)$*, is a basis of the complementary vector space of V formed by selecting among the rows $\vec{e}_1$, ..., $\vec{e}_n$ of the identity matrix of size n, in order, $n - \dim(V)$ vectors linearly independent with V where $\vec{e}_1$, ..., $\vec{e}_n$ are*

□

The new coordinate of $\vec{p}$ inside $\mathcal{S}$ forms the vector

$$\vec{p'} = \{p_1, \ldots, p_m, p_{m+1}, \ldots p_n\}$$

One then keeps only the $n - m$ last components of $\vec{p'}$, to obtain a vector $\vec{p''}$ with $n - m$ components. This operation can be formalized as the application of a projection that maps every point of $\mathbb{R}^n$ in V onto $\vec{0}$ (in a $(m - n)$–dimensional space) and then expresses the resulting points into the coordinate system formed by the basis of $\{w_1, w_2, \ldots, w_{n-m}\}$.

To get rid of the magnitude of the resulting vector $\vec{p''}$, which is not relevant since the problem is invariant by scalings, a normalization operation is applied to it. This operation replaces the vector by its intersection with the faces of a *normalization hypercube*, also called *normalization cube* for the sake of readability. This closed set is given by :

$$\Big\{(x_1, \ldots, x_{n-m}) \in \mathbb{R}^{n-m}) \mid (\exists 0 < i \leq n - m)(\forall 0 < j \leq n - m)(i \neq j) ((x_i \in \{-1/2, 1/2\}) \wedge (x_j \in [-1/2, 1/2]))\Big\}$$

and it is, by this definition, centered on the origin, and of size 1.

5.2.3 Encoding Scheme for Directions

Let us now study the encoding scheme applied to a normalized point $\vec{p} \in \mathbb{R}^{n-m}$ in order to follow transitions inside a decision structure. The scheme is based on labeling subdivisions of the surface of the normalization hypercube.

The first symbol of the encoding identifies a face of the cube in which the normalized point belongs (it always belongs to at least one face and at most $n - m$ faces). Formally, the encoding of a normalized point $\vec{p} \in \mathbb{R}^{n-m}$ begins with a symbol $\sigma \in \{-1, +1, -2, +2, \ldots, -k, +k\}$ with $k = n - m$. If $\sigma = -i$ with $0 < i \leq n - m$, then $\vec{p}[i] = -\frac{1}{2}$. If $\sigma = +i$ with $0 < i \leq n - m$, then $\vec{p}[i] = \frac{1}{2}$. Remark that $\pm i$ is a single symbol of the alphabet.

The suffix is an infinite word $w \in \{0, 1\}^\omega$ that encodes the position of $\vec{p}$ inside the face of the normalization cube that corresponds to σ. If the leading symbol is $+i$ or $-i$, then $\vec{p}[i] = \frac{1}{2}$ or $\vec{p}[i] = -\frac{1}{2}$. The suffix following the face symbol is a serial binary encoding of the vector $\vec{p}_{|\neq i} + \left[\frac{1}{2}\right]^{(n-m)-1} \in \mathbb{R}^{n-m-1}$.

To summarize, the following procedure computes an encoding of a point $\vec{p} \in \mathbb{R}^{n-m}$.

Encoding Procedure

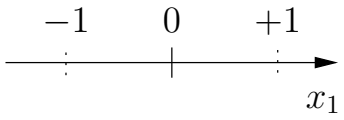
Step 1 : Select $i \in \{1, 2, \ldots, n-m\}$ such that $\vec{p}[i] \in \{-\frac{1}{2}, \frac{1}{2}\}$.
Step 2 : The first symbol $\sigma \in \pm(\mathbb{N})$ is $+i$ if $\vec{p}[i] = \frac{1}{2}$ and $-i$ otherwise.
Step 3 : Let $\vec{p'} = \vec{p}_{|\neq i} + \left[\frac{1}{2}\right]^{n-m-1}$.
Step 4 : Subsequent symbols are the fractional part of the classical serial binary encoding of p'.

This encoding scheme allow points to have multiple encodings which, as it will be discussed in Section 5.4.2, turns out to be an essential property. It is nevertheless important to guarantee that following paths labeled by different encodings of the same point lead to identical decisions.

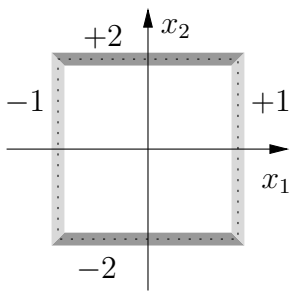
We denote by $\langle \vec{x}, V \rangle$ the set of encodings of a point $\vec{x}$ in the canonical complementary vector space of the vector space V.

Example 5.9 *We give some examples of face numeration inside $\mathbb{R}^n$. In each example, $\vec{e}_i$ is the i^{th} vector of the Cartesian basis of $\mathbb{R}^n$. The normalization hypercube is represented with dotted lines.*

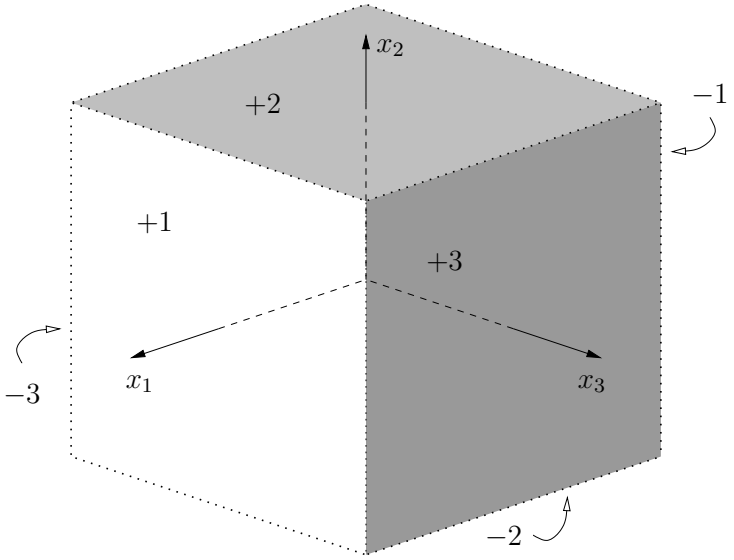
- *$n = 1$: The vector basis is $\{\vec{e}_1\}$. There are two faces $+1$ and -1.*



- *$n = 2$: The vector basis is $\{\vec{e}_1, \vec{e}_2\}$. There are four faces $+1$, $+2$, -1 and -2. In the following figure, light gray areas shows all the points of $\mathbb{R}^2$ having as face decision in their encoding $+1$ or -1. Dark gray areas shows those with $+2$ or -2.*



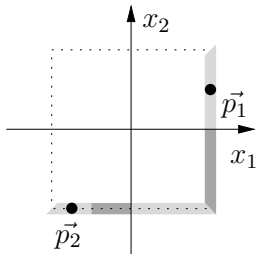
- $n = 3$: *The vector basis is* $\{\vec{e}_1, \vec{e}_2, \vec{e}_3\}$. *There are six faces* $+1$, $+2$, $+3$, -1, -2 *and* -3. *In the following figure, the white area shows the* $+1$ *face of the normalization cube. Light gray shows the face* $+2$ *and dark gray shows the face* $+3$. -1, -2 *and* -3 *are not visible in the figure, but their emplacement are suggested with arrows.*



◇

Example 5.10 *We now illustrate the encoding of points in* $\mathbb{R}^2$.

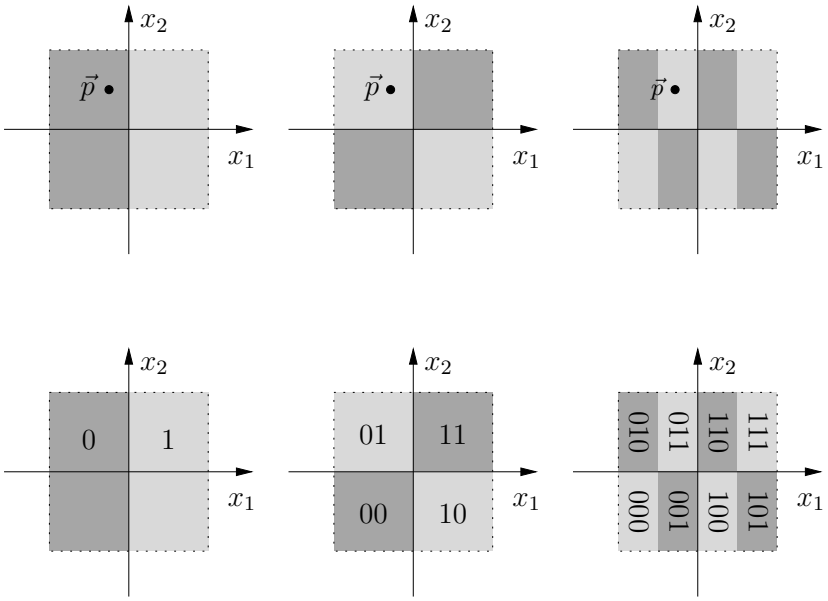
Let $\vec{p}_1 = (1/2, 1/4)$ *and* $\vec{p}_2 = (-3/8, 1/2)$ *be two normalized points. An encoding of length* 2 *for* $\vec{p}_1$ *is* $+1$ 1. *An encoding of length* 3 *for* p_2 *is* -2 00. *The infinite encodings of* $\vec{p}_1$ *are given by* $+1$ 101^ω *and* $+1$ 110^ω *and for* $\vec{p}_2$ *by* -2 0001^ω *and* -2 0010^ω.



◇

Example 5.11 *We now show how the normalized point* $\vec{p} = (-\frac{1}{8}, \frac{1}{4}, -\frac{1}{2})$ *is encoded in* $\mathbb{R}^3$ *up to* 4 *symbols.*

$\vec{p}$ *belongs to the face* -3*, so its encoding starts by symbol* -3*. It is followed by symbol* 0 *since* $\vec{p}_{|\neq 3} \in [-\frac{1}{2}, 0] \times [-\frac{1}{2}, \frac{1}{2}]$*. The next symbol is* 1*, as* $\vec{p}_{|\neq 3} \in [-\frac{1}{2}, 0] \times [0, \frac{1}{2}]$ *regions, followed by* 1 *because* $\vec{p}_{|\neq 3} \in [-\frac{1}{4}, 0] \times [0, \frac{1}{2}]$*, etc.*



◇

5.2.4 Syntax

Let us now introduce precisely the syntactic definition of IRVA.

Definition 5.12 *An Implicit Real Vector Automaton (IRVA) is a tuple* $(n, S_I, S_E, s_0, \delta, \mathrm{VS}, \mathrm{col})$*, where*

- $n \in \mathbb{N}$ *is a* ***dimension****,*
- S_I *is a finite set of* ***implicit states****,*
- S_E *is a finite set of* ***explicit states****,*
- $s_0 \in S_I$ *is an* ***initial state****,*
- $\delta : (S_I \times \pm\mathbb{N}_{>0}) \cup (S_E \times \{0,1\}^*) \to S_I \cup S_E$ *is a partial* ***transition function****,*
- $\mathrm{VS} : S_I \to 2^{\mathbb{R}^n}$ *associates a* ***vector space*** *to each implicit state,*
- $\mathrm{col} : S_I \cup S_E \to \mathbb{N}_{>0}$ *associates a* ***color*** *to each implicit state.*

□

Definition 5.13 *Let* $(n, S_I, S_E, s_0, \delta, \mathrm{VS}, \mathrm{col})$ *be an IRVA. The set of symbols that labels the outgoing transitions of an implicit state* $s \in S_I$*, noted* $\Sigma(s)$*, is the set :*

$$\Sigma(s) = \bigcup_{1 \leq i \leq n - \dim(\mathrm{VS}(s))} \{+i, -i\}$$

□

In addition, to facilitate the presentation of concepts for the rest of this work, we define the following functions :

- δ^* is the reflexive and transitive closure of δ. We thus have : $\delta^*(s, \varepsilon) = s$ and ε the empty word, and

  $$\delta^*(s, w) = \delta(\ldots(\delta(\delta(s, \sigma_1), \sigma_2), \ldots), \sigma_k)$$

  with $s \in S_I \cup S_E$, $w = \sigma_1\sigma_2\ldots\sigma_k$ and $(\forall i \in \{1, 2, \ldots, k\})(\sigma_i \in \pm\mathbb{N} \cup \{0, 1\})$

- $\delta^*(s)$ is the set of all reachable states from a state $s \in S_I \cup S_E$.

  $$\delta^*(s) = \{s' \in S_I \cup S_E \mid (\exists w \in (\pm\mathbb{N} \cup \{0, 1\})^*(\delta^*(s, w) = s')\}$$

- $\delta_I(s)$ is the set of the first reachable implicit states from a state $s \in S_I \cup S_E$:

  $$\delta_I(s) = \begin{cases} \{s' \in S_I \mid (\exists w \in \{0, 1\}^*)(\delta^*(s, w) = s')\} & \text{if } s \in S_E \\ \{s' \in S_I \mid (\exists w \in \Sigma(s)\{0, 1\}^*)(\delta^*(s, w) = s')\} & \text{if } s \in S_I \end{cases}$$

- $L_\delta(s, s')$ the set of words labeling paths from a state s to a state s', with $s, s' \in S_I \cup S_E$:

  $$L_\delta(s, s') = \{w \in ((\pm\mathbb{N}) \cup \{0, 1\})^* \mid \delta^*(s, w) = s'\}$$

5.2.5 Syntactic Constraints

All the structures that can be produced from this syntax does not correspond to valid IRVA. For this reason we will give a set of syntactic rules that an IRVA candidate must fulfill to be syntactically valid.

An IRVA $(n, S_I, S_E, s_0, \delta, \mathrm{VS}, \mathrm{col})$ has to satisfy the following syntactic constraints :

1. The transition function must be acyclic, i.e. such that :

   $$(\forall s, s' \in S_I \cup S_E)((s' \in \delta^*(s)) \wedge (s \neq s') \Rightarrow s \notin \delta^*(s'))$$

2. The transition function must be complete :

$$(\forall s \in S_I)(\forall \sigma \in \Sigma(s))(\exists s' \in S_I \cup S_E)(\delta(s,\sigma) = s')$$

$$(\forall s \in S_E)(\exists s', s'' \in S_I \cup S_E)(\delta(s,0) = s' \wedge \delta(s,1) = s'')$$

3. Each state must be reachable from the initial state :

$$(\forall s \in S_I \cup S_E)(s \in \delta^*(s_0))$$

4. When an implicit state s' is reachable from another implicit state s, the associated vector space of s must be proper a subset of the one of s' :

$$(\forall s, s' \in S_I)(s' \in \delta_I(s) \Rightarrow \mathrm{VS}(s) \subset \mathrm{VS}(s'))$$

Note that this implies $\dim(\mathrm{VS}(s)) < \dim(\mathrm{VS}(s'))$.

5. Each word that can be read from one implicit state s to another s' must be a prefix of an encoding of at least one point that belongs to $\mathrm{VS}(s')$:

$$(\forall w \in L_\delta(s, s'))(R_{\mathrm{VS}(s),w}) \cap \mathrm{VS}(s') \neq \emptyset$$

with $s, s' \in S_I$.

5.3 Semantics

In order to associate a semantics to this syntactic definition, we will introduce a notion that associates a path leaving an implicit state with a region of space in which every point can be encoded locally by the labels of the transitions composing this path. It corresponds to what we have called *path regions* since Section 5.1.3. We now define them more formally.

5.3.1 Path Regions

For each word that can be followed inside a decision structure leaving an implicit state s of an IRVA, we consider the set of points that admit an encoding prefixed by this word.

More formally, given a prefix w and a vector space V the *path region* of w with respect to V, noted $R_{\mathrm{VS}(s),w}$, is the set of points that admit an encoding prefixed by w.

Definition 5.14 *Let $V \subset \mathbb{R}^n$ and $w \in \pm\mathbb{N}\{0,1\}^*$ be a word. The **path region** $R_{V,w}$ is the set :*

$$\{\vec{x} \in \mathbb{R}^n \mid (\exists e \in \langle \vec{x}, V \rangle)(\exists w' \in \{0,1\}^\omega)(e = ww')\}$$

□

Such path regions are useful in order to consider at once all the points that share a path from a particular implicit state. Such paths begin in an implicit state and follow transitions inside the decision structure from this implicit state. Once another implicit state is reached, the encoding scheme is reset, and thus reading more symbols without changing the encoding is not possible. Indeed, the transitions in the decision structure of the newly reached implicit state have symbols corresponding to encodings of points expressed with respect to the vector space of this new state.

Let us point out that for a point $\vec{x} \in \mathbb{R}^n$ and an IRVA $(n, S_I, S_E, s_0, \delta, \mathrm{VS}, \mathrm{col})$, it is possible to define a path that starts in an implicit state s of the IRVA, follows some transitions that corresponds to one encoding $e \in \langle \vec{x}, \mathrm{VS}(s) \rangle$, until reaching a second implicit state s'. From s', it is possible to extend further the path by following some transitions that corresponds to one encoding $e' \in \langle \vec{x}, \mathrm{VS}(s') \rangle$ and so on.

This leads us to the characterization of a *combined path region* defined by an implicit state s_1 and a word $w = w_1, w_2, \ldots, w_k \in \pm\mathbb{N}\{0,1\}^*$. The combined path region of a word read from an implicit state s_1 and visiting implicit states $s_2, \ldots, s_k$ is the set of points $\vec{x}$ such that, for each state s_i, there exists w_i, a prefix of at least one element of $\langle \vec{x}, \mathrm{VS}(s_i) \rangle$ and such that $w_1 w_2 \ldots w_k = w$.

Definition 5.15 *Let $\mathcal{A} = (n, S_I, S_E, s_0, \delta, \mathrm{VS}, \mathrm{col})$ be an IRVA, $s_1 \in S_I$ an implicit state and w be a word in $(\pm\mathbb{N}\{0,1\}^*)^k$ such that $k \in \mathbb{N}_{>0}$ and such that reading w from s_1 visits k implicit states. We have the following sequence :*

$$s_1 \xrightarrow{w_1} s_2 \xrightarrow{w_2} \ldots s_k \xrightarrow{w_k}$$

*The **combined path region** $\mathrm{CR}_{\mathcal{A},s_1,w}$ is the set*

$$\mathrm{CR}_{\mathcal{A},s_1,w} = \bigcap_{i=1}^{k} R_{\mathrm{VS}(s_i),w_i}$$

□

5.3.2 Point Decision in IRVA

Let $\mathcal{A} = (n, S_I, S_E, s_0, \delta, \mathrm{VS}, \mathrm{col})$ be an IRVA and $\vec{x} \in \mathbb{R}^n$ be a point.

Definition 5.16 *A **run** of $\mathcal{A}$ over $\vec{x}$ is a finite sequence*

$$(s_0, w_1)(s_1, w_2) \ldots (s_{m-1}, w_m)(s_m, \varepsilon)$$

where $0 \leq m \leq n$, $s_0, s_1, \ldots, s_m \in S_I$, $w_1, w_2, \ldots, w_{m-1}, w_m \in \pm(\mathbb{N}_{>0}\{0,1\}^*$ and such that $\vec{x} \in \mathrm{VS}(s_m)$, and for every $i \in \{0, 1, m-1\}$, $\vec{x} \notin \mathrm{VS}\, s_i$ and $\vec{x} \in R_{\mathrm{VS}(s_i),w_{i+1}}$. □

5.3.3 Well-Formed IRVA

Some syntactically valid IRVA can nevertheless be inconsistent. Indeed, since a point can admit multiple encodings, inconsistencies appears if the reading of those encodings are not coherent with one another.

Those IRVA are called badly formed. *Well formed* IRVA are defined as follows :

Definition 5.17 *An IRVA* $\mathcal{A} = (n, S_I, S_E, s_0, \delta, \mathrm{VS}, \mathrm{col})$ *is* ***well-formed*** *if it is syntactically valid and if, for every* $\vec{x} \in \mathbb{R}^n$*, each of its runs over* $\vec{x}$ *ends up in the same implicit state.* □

Note that the RVA data structure has similar restriction, since it requires that all the encodings of a point have to be recognized accordingly.

5.3.4 Coloring Function of an Implicit State

We have seen that a color information is associated to each implicit state of IRVA. The use of two colors permits the definition of classical pyramidal polyhedra and the use of more colors enables the representations of pyramidal partitions. Use have extensively used binary color scheme for clarity sake. In this section we are in the context of the more general arbitrary color scheme.

An IRVA representing a pyramidal partition associates, to each point of space, a part containing this point. We can formalize this association by defining the *coloring function* of the IRVA. In order to do so, we associate a unique color for each part.

Technically, we will associate a coloring function to each implicit state, as it is possible to consider only a fraction of the IRVA from one implicit state as another IRVA.

Theorem 5.18 *Let* $(n, S_I, S_E, s_0, \delta, \mathrm{VS}, \mathrm{col})$ *be an IRVA and* $s \in S_I$ *be an implicit state. The coloring function of* s*, noted* COLOR_s *is such that* $\mathrm{COLOR}_s : \mathbb{R}^n \to \mathbb{N}_{>0}$ *as :*

$$\mathrm{COLOR}_s(\vec{x}) = \begin{cases} \mathrm{col}(s) & \text{if } \vec{x} \in \mathrm{VS}(s) \\ \mathrm{COLOR}_{s'}(\vec{x}) & \text{if } \vec{x} \in R_{\mathrm{VS}(s),w} \end{cases}$$

with $\vec{x} \in \mathbb{R}^n$*,* $s' \in \delta_I(s)$*,* $w \in L_\delta(s, s')$*.*

We now have a constructive description of the coloring function associated to each implicit state of the IRVA. The coloring function of the initial state s_0 is then the partition represented by the IRVA. We obtain the following theorem :

Theorem 5.19 *Every well-formed IRVA* $(n, S_I, S_E, s_0, \delta, \mathrm{VS}, \mathrm{col})$ *represents a pyramidal partition of* $\mathbb{R}^n$*.*

5.3.5 Sets Represented from an Implicit State

The following formula gives, constructively, the set represented from a particular implicit state s of an IRVA $(n, S_I, S_E, s_0, \delta, \mathrm{VS}, \mathrm{col})$ for a particular color c :

$$\mathrm{SET}(s,c) = \begin{cases} \bigcup\limits_{s' \in \delta_I(s)} \bigcup\limits_{w \in L_\delta(s,s')} \Big(R_{\mathrm{VS}(s),w} \cap \mathrm{SET}(s',c)\Big) \cup \mathrm{VS}(s) & \text{if } \mathrm{col}(s) = c \\ \bigcup\limits_{s' \in \delta_I(s)} \bigcup\limits_{w \in L_\delta(s,s')} \Big(R_{\mathrm{VS}(s),w} \cap \mathrm{SET}(s',c)\Big) & \text{if } \mathrm{col}(s) \neq c \end{cases}$$

5.3.6 Point Classification in IRVA

Let $\mathcal{A} = (n, S_I, S_E, s_0, \delta, \mathrm{VS}, \mathrm{col})$ be an IRVA, $s, s' \in S_I$ be two implicit states and $\vec{x} \in \mathbb{R}^n$ be a point.

The expression

$$s' = \mathbf{class}(\mathcal{A}, s, \vec{x})$$

states that the run of $\mathcal{A}$ over $\vec{x}$ from s terminates in s'.

5.4 Minimum Element

Let s_1 be an implicit state and $w \in \pm\mathbb{N}\{0,1\}^*$ be a word. We have seen that reading w from s_1 defines a path region $R_{\mathrm{VS}(s_1),w}$ containing all points of space such that w is a prefix of one of their encodings.

This means that, in order to classify any point of $R_{\mathrm{VS}(s),w}$ from s_1, one can follow the path labeled with w and take a decision from the destination of the path labeled by w. If this path leads to an implicit state s_2, the classification procedure can be carried out from s_2.

As it is possible to have points of $R_{\mathrm{VS}(s),w}$ that do not belong to $\mathrm{VS}(s_2)$, the runs of those points over the IRVA from s will eventually end into another implicit state. This other implicit state is necessarily reachable from s_2.

This leads us to the notion of minimum element, similar to the notion of minimum polyhedral component introduced in Section 5.1.4.

Let $\mathcal{A} = (n, S_I, S_E, s_0, \delta, \mathrm{VS}, \mathrm{col})$ be an IRVA and $S \subseteq S_I$ a selection of implicit states of this IRVA. An implicit state s is the minimum element of S iff :

$$(\forall s_i \in S)(s_i \in \delta^*(s))$$

This minimum element may not be defined. When it is the case, it is denoted by the use of a distinguished symbol $\perp$.

We have :

$$\mathbf{minelset}(S) = \begin{cases} s \in S & \text{if } (\forall s_i \in S)(s_i \in \delta^*(s)) \\ \perp & \text{otherwise.} \end{cases}$$

5.4.1 Minimum Covered Element

Later in this work, we will need to find the implicit state of an IRVA that corresponds to the minimum covered component of the pyramidal partition represented from an implicit state of an IRVA with respect to a region of space.

We have presented the notion of minimum covered component (see Definition 3.25). We will now see how, given an IRVA $\mathcal{A}$ and an implicit state from where a pyramidal partition Π is represented, we can find the implicit state representing the minimum covered component of Π by a region R. We call this implicit state the *minimum covered element* by R from s in $\mathcal{A}$. This operation is noted $\mathbf{minel}(\mathcal{A}, s, R)$.

Let $\mathcal{A} = (n, S_I, S_E, s_0, \delta, \mathrm{VS}, \mathrm{col})$ be an IRVA, $R \subseteq \mathbb{R}^n$ be a non empty convex region and $s \in S_I$ be an implicit state. The minimum element of $\mathcal{A}$ covered by R from s is the implicit state $s_m \in S_I$, which is the minimum element of the set :

$$\{s_i \in S_I \mid (\exists w \in L_\delta(s))(\mathrm{CR}_{\mathcal{A},s,w} \cap \mathrm{VS}(s') \cap R \neq \emptyset)\}$$

the set of reachable implicit states from s such that $\mathrm{VS}(s)$ has an non empty intersection with $\mathrm{CR}_{\mathcal{A},s,w}$ and the convex region (w labeling paths which lead from s to these implicit states).

Theorem 5.20 *For an IRVA $\mathcal{A} = (n, S_I, S_E, s_0, \delta, \mathrm{VS}, \mathrm{col})$, two implicit states $s, s' \in S_I$ (s and s' can be the same state) and a region $R \subseteq \mathbb{R}^n$, we have :*

$$s' = \textbf{\textit{minel}}(\mathcal{A}, s, R) \Rightarrow (\forall \vec{x} \in R)(\textbf{\textit{class}}(\mathcal{A}, s, \vec{x}) = \textbf{\textit{class}}(\mathcal{A}, s', \vec{x}))$$

Proof : In the case where s and s' is the same state, the result is straightforward.

Otherwise, the state returned by the classification procedure of $\vec{x}$ over $\mathcal{A}$ from the state s, noted $\mathbf{class}(\mathcal{A}, s, \vec{x})$, can only be an implicit state that belongs to the set $\delta^*(s)$.

The fact that we have $s' = \mathbf{minel}(\mathcal{A}, s, R)$, ensures us that :

$$(\forall \vec{x} \in R)(\mathbf{class}(\mathcal{A}, s, \vec{x}) \in \delta^*(s'))$$

Otherwise, it would be possible to find a point $\vec{x'} \in R$ such that $\mathbf{class}(\mathcal{A}, s, \vec{x'}) = s''$ with $s'' \notin \delta^*(s')$. But as $\vec{x'} \in \mathrm{VS}(s'')$ and $\vec{x'} \in R$, we obtain that $\mathrm{VS}(s'') \cap R \neq \emptyset$. And thus that $R_{\mathrm{VS}(s),w'} \cap \mathrm{VS}(s'') \cap R \neq \emptyset$ with $w' \in L_\delta(s, s'')$ being an encoding of $\vec{x'}$ from s. s'' would then have made impossible for $\mathbf{minel}(\mathcal{A}, s, R)$ to return s' in the first place, which was not the case.

As a consequence, it is clear that we can search for $\mathbf{class}(\mathcal{A}, s, \vec{x})$ from s' and thus that we will obtain : $\mathbf{class}(\mathcal{A}, s, \vec{x}) = \mathbf{class}(\mathcal{A}, s', \vec{x})$ ■

Now, let us examine the case were $\mathbf{minel}(\mathcal{A}, s, R)$ is undefined. When it is the case, it is due to the fact that a set of polyhedral components of the

pyramidal partition represented by $\mathcal{A}$ from s have a non empty intersection with R but have a common parent (with respect to the incidence relation) with no intersection with R. This indicates that the search for a minimum element is a particular case of a more general procedure that calculates the set of minimal covered elements of $\mathcal{A}$, from state s by region R and we will need this operation later in this work to identify the components that prevents the minimum to be defined.

Let $\mathcal{A}$ be an IRVA, s be an implicit state and R be a non empty convex region of $\mathbb{R}^n$. If C is the polyhedral component represented by s and if Π is the pyramidal partition represented by $\mathcal{A}$ from s, it is always possible to calculate the set Q of implicit states of $\mathcal{A}$ such that each element $q \in Q$ represents a polyhedral component of Π having a non empty intersection with R and being incident or equal to C, and such that there is no other member of Q to which it is incident to.

This set is the set of minimal covered elements of $\mathcal{A}$ from s by R and is calculated by the procedure $\mathbf{minimals}(\mathcal{A}, s, R)$.

Algorithm for Finding Minimal Covered Elements

We will now develop algorithms to find the set of minimal covered elements and the minimum covered element of an IRVA $\mathcal{A} = (n, S_I, S_E, s_0, \delta, \mathrm{VS}, \mathrm{col})$ from a state s for a non empty convex region R.

First, let us make two observations : the set of minimal covered elements is always non empty, and when the minimum covered element is defined, it is due to the fact that the set of minimal covered elements contains only one element (and it is of course the minimum).

A first function will address the problem of keeping a set of minimal elements after merging two sets of implicit states. The function **merge** takes two sets of implicit states S_1 and S_2 and is defined as :

$$\mathbf{merge}(S_1, S_2) = \{s \in S_1 \cup S_2 \mid (\nexists s' \in (S_1 \cup S_2) \setminus \{s\})(s \in \delta^*(s')\}$$

The second function is called *minimals* and calculates the set of minimal covered elements of $\mathcal{A}$ from s by the region R.

It is defined as :

$$\mathbf{minimals}(\mathcal{A}, s, R) = \mathbf{search\_minimals}(\mathcal{A}, s, R, \varepsilon)$$

See Algorithm 1 for a complete definition of **search_minimals**.

Finally, the last function, called `minel` is the function that finds the minimum element inside an IRVA $\mathcal{A}$ from a state s covered by a non empty convex path region R. It is defined as

$$\mathbf{minel}(\mathcal{A}, s, R) = \begin{cases} s' & \text{if } \mathbf{minimals}(\mathcal{A}, s, R) = \{s'\}. \\ \bot & \text{if } |\,\mathbf{minimals}(\mathcal{A}, s, R)| > 1. \end{cases}$$

input : An IRVA $\mathcal{A} = (n, S_I, S_E, s_0, \delta, \mathrm{VS}, \mathrm{col})$, a state $s \in S_I$, a non empty convex polyhedron $R \subset \mathbb{R}^n$ and a word $w \in \{\varepsilon\} \cup \pm\mathbb{N}\{0,1\}^*$
output: A set of implicit states of $\mathcal{A}$

if $\mathrm{VS}(s) \cap R \neq \emptyset$ **then**
  return $\{s\}$;
else
  $S := \emptyset$;
  foreach $\sigma \in \mathtt{succ}(s')$ **do**
    $w' := w\sigma$;
    $R' := R_{\mathrm{VS}(s),w'} \cap R$;
    if $R' \neq \emptyset$ **then**
      $s' := \delta^*(s, w')$;
      if $s' \in S_I$ **then**
        if $R' \cap \mathrm{VS}(s') \neq \emptyset$ **then**
          $S :=$ **merge**$(S, \{s'\})$;
        else
          $S :=$ **merge**$(S,$ **search_minimals**$(\mathcal{A}, s', R', \varepsilon))$;
        end
      else
        $S :=$ **merge**$(S,$ **search_minimals**$(\mathcal{A}, s, R, w'))$;
      end
    end
  end
  return S;
end

Algorithm 1: Computation of **search_minimals**$(\mathcal{A}, s, R, w)$

5.4.2 Property of a Good Encoding Scheme

We have mentioned before that a point admits multiple encodings with respect to the outgoing decision structure from an implicit state. This is due to two reasons. First, each encoding of a point starts by a face selection among the faces of an hypercube of normalization. The normalization of the direction in which the point leaves the vector space can belong to multiple faces, hence there will be at least one encoding for each face. Second, each subsequent symbols decomposes the face into two pieces along one axis. We have seen that each piece is a closed set. Every point that belongs to the boundary of such piece will admit at least two encodings.

We have claimed in Section 5.2.3 that this particular property is mandatory. We will see why and how it is linked to the notion of minimum covered element by considering a counter example.

As presented in the beginning of this chapter with the abstract data structure, we consider the encoding scheme to be a tool aimed at organizing the outgoing decision structure from implicit states. Imagine that a different encoding scheme that generates exactly one encoding for each point is used. It implies that, when a symbol is read, space is partitioned into distinct parts. Some of those parts have open boundaries.

For example, in Figure 5.6, we consider a square and one of its possible partitions. In the figure, plain lines indicate closed boundaries and dashed lines indicate open ones.

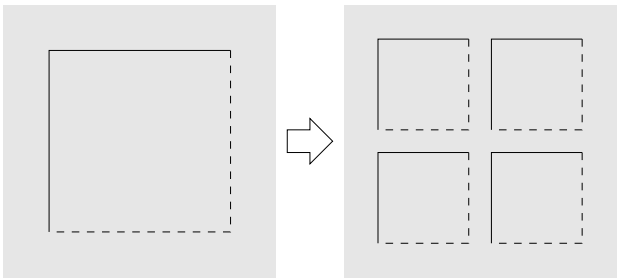


Figure 5.6: Partitioning of a square.

Recall that we have seen in Section 5.1.4 that when a path w links two implicit states s_1 and s_2 inside an IRVA indicates the fact that the polyhedral component corresponding to s_2 is the minimum covered component inside the path region $R_{\mathrm{VS}(s_1),w}$.

Let us consider, as an example, that we are constructing an IRVA representing a 3–dimensional set and let the square in Figure 5.7 correspond to the intersection of a path region $R_{\emptyset,w}$ defined by reading a word w from an implicit state representing $\vec{0}$. On Figure 5.7, we show this region and the polyhedral components a and b, corresponding to planes, visible inside it. Remark that P, which is the ray of intersection of the planes, is not visible in this region.

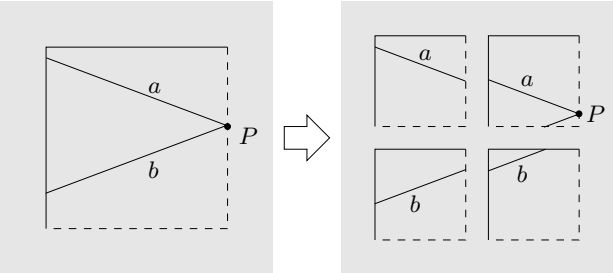


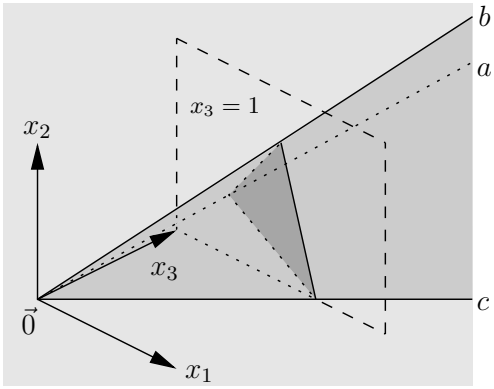
Figure 5.7: Partitioning of a square containing the polyhedral components a and b, but not P.

The decomposition on the right shows that the situation has not changed at all. More subdivisions of the upper right part are necessary because a minimum element is not defined. The problem is that this construction does not terminate. Indeed, in this situation, the distance between a and b is zero, but their intersection P does not belong to the region. Subdividing the regions further will never change this fact. That is precisely the problem with encoding schemes that allow open boundaries for spacial subdivisions.

With our encoding scheme, the problem does never occur, because if some elements are separated by a distance equal to zero, then their intersection has to be visible in this region as well.

5.5 Example of IRVA

We now illustrate a complete example of IRVA. We consider an IRVA representing the 3–dimensional pyramid already introduced in Example 5.4. Let $(\vec{e}_1, \vec{e}_2, \vec{e}_3)$ be a Cartesian coordinate system of $\mathbb{R}^3$.



The pyramid is composed of nine polyhedral components : the origin $\vec{0}$, the three rays $a = (1/2, 1/2, 1)$, $b = (4/3, 4/3, 1)$ and $c = (2, 0, 1)$, the three planes ab, bc and ac and the two 3–spaces *in* and *out*, respectively the inside and the outside of the pyramid. We assign color **gray** to the whole pyramid – boundary and interior – and **white** to its outside.

The IRVA representing this polyhedron has nine implicit states s_0, s_a, s_b, s_c, s_{ab}, s_{bc}, s_{ac}, s_{in}, s_{out} representing respectively the polyhedral components $\vec{0}$, a, b, c, ab, bc, ac, in, out.

The initial state of the IRVA is s_0.

The set of implicit states is :

$$S_I = \{s_0, s_a, s_b, s_c, s_{ab}, s_{bc}, s_{ac}, s_{in}, s_{out}\}$$

The vector spaces associated to those implicit states are given in the following table :

$\mathrm{VS}(s_0)$	$\emptyset$
$\mathrm{VS}(s_a)$	$\{(1/2, 1/2, 1)\}$
$\mathrm{VS}(s_b)$	$\{(4/3, 4/3, 1)\}$
$\mathrm{VS}(s_c)$	$\{(2, 0, 1)\}$
$\mathrm{VS}(s_{ab})$	$\{(1/2, 1/2, 1), (4/3, 4/3, 1)\}$
$\mathrm{VS}(s_{bc})$	$\{(4/3, 4/3, 1), (2, 0, 1)\}$
$\mathrm{VS}(s_{ac})$	$\{(1/2, 1/2, 1), (2, 0, 1)\}$
$\mathrm{VS}(s_{in})$	$\{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$
$\mathrm{VS}(s_{out})$	$\{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$

To define the transition function, we need to know the canonical basis of the complementary vector spaces of each state. Remember that it defines in which coordinate system the directions going out of each implicit state are going to be expressed. This of course also defines their encodings.

$\mathrm{CVS}(s_0)$	$\{\vec{e}_1, \vec{e}_2, \vec{e}_3\}$
$\mathrm{CVS}(s_a)$	$\{\vec{e}_1, \vec{e}_2\}$
$\mathrm{CVS}(s_b)$	$\{\vec{e}_1, \vec{e}_2\}$
$\mathrm{CVS}(s_c)$	$\{\vec{e}_1, \vec{e}_2\}$
$\mathrm{CVS}(s_{ab})$	$\{\vec{e}_1\}$
$\mathrm{CVS}(s_{bc})$	$\{\vec{e}_1\}$
$\mathrm{CVS}(s_{ac})$	$\{\vec{e}_1\}$
$\mathrm{CVS}(s_{in})$	$\emptyset$
$\mathrm{CVS}(s_{out})$	$\emptyset$

It remains to define the transition function and the set of explicit states. Let us examine the intersection of the pyramid and the normalization cube $[-\frac{1}{2}, \frac{1}{2}]^3$ (Fig. 5.8) in order to establish the successors of the initial state. As $\dim(\mathrm{VS}(s_0)) = 0$, we have $\Sigma(s_0) = \{-3, -2, -1, +1, +2, +3\}$. Each of those symbols identifies a face of the normalization cube. The following table shows the set of polyhedral components that each face intersects and the minimum component of those sets. When the minimum covered component is defined for the path region corresponding to the word, the implicit state representing this component can be set as destination by this word. When there is no minimum covered component defined, one simply makes the word longer until it defines a path region fine enough for a minimum covered component to be defined.

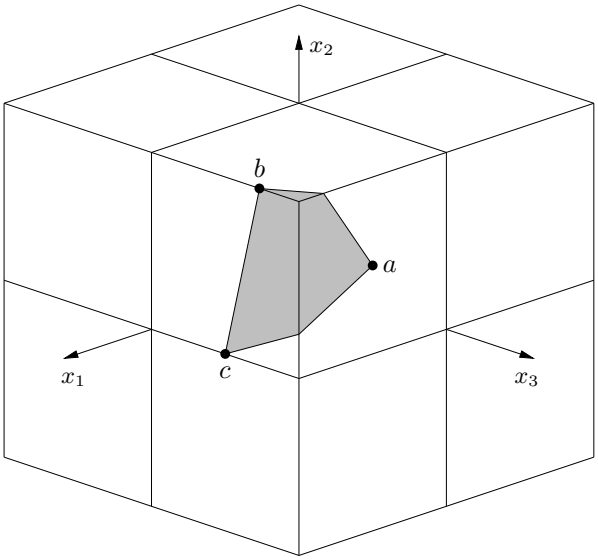


Figure 5.8: The intersection of the pyramid with the normalization cube $[-1/2, 1/2]^3$.

state	word	path region	components	min. component
s_0	$+1$	$R_{\mathrm{VS}(s_0),+1}$	$\{b, c, bc, ac, in, out\}$	$\perp$
s_0	$+1\ 0$	$R_{\mathrm{VS}(s_0),+1\ 0}$	$\{c, out\}$	c
s_0	$+1\ 1$	$R_{\mathrm{VS}(s_0),+1\ 1}$	$\{b, c, bc, ac, in, out\}$	$\perp$
s_0	$+1\ 10$	$R_{\mathrm{VS}(s_0),+1\ 10}$	$\{out\}$	out
s_0	$+1\ 11$	$R_{\mathrm{VS}(s_0),+1\ 11}$	$\{b, c, bc, ac, in, out\}$	$\perp$
s_0	$+1\ 110$	$R_{\mathrm{VS}(s_0),+1\ 110}$	$\{c, bc, ac, in, out\}$	c
s_0	$+1\ 111$	$R_{\mathrm{VS}(s_0),+1\ 111}$	$\{b, bc, in, out\}$	b
s_0	$+2$	$R_{\mathrm{VS}(s_0),+2}$	$\{b, ab, in, out\}$	b
s_0	$+3$	$R_{\mathrm{VS}(s_0),+3}$	$\{a, ab, ac, in, out\}$	a
s_0	-1	$R_{\mathrm{VS}(s_0),-1}$	$\{out\}$	out
s_0	-2	$R_{\mathrm{VS}(s_0),-2}$	$\{out\}$	out
s_0	-3	$R_{\mathrm{VS}(s_0),-3}$	$\{out\}$	out

The face $+1$ intersects both a and b, hence deciding between them is impossible after having read only the symbol $+1$. Therefore, $\delta(s_0, +1)$ directs to an explicit state and its successors need to be explored. Each of the other faces, however, identify a minimum covered component, so their destination will be the implicit states representing those components.

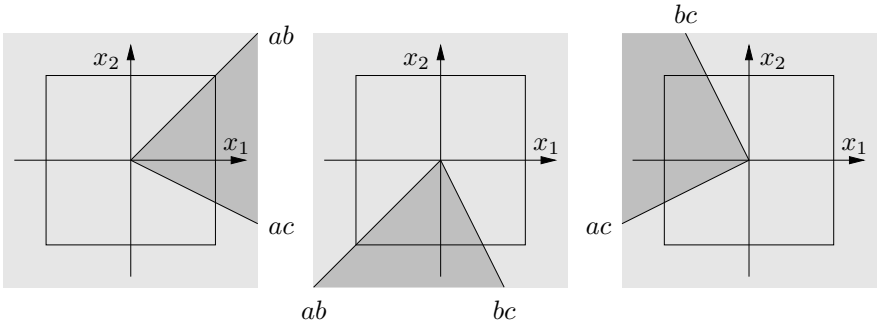


Figure 5.9: Geometrical situation of (a) component a in CVS(s_a) (b) component b in CVS(s_b) (c) component c in CVS(s_c).

Now we develop the successors of the states s_a, s_b and s_c. An illustration of the situation is represented, respectively, in Figures 5.9(a), 5.9(b) and 5.9(c). Here is the result of the exploration :

word	components	min. component
$\delta^*(s_a, +1)$	$\{ab, ac, in, out\}$	$\perp$
$\delta^*(s_a, +1\ 0)$	$\{ac, in, out\}$	ac
$\delta^*(s_a, +1\ 1)$	$\{ab, in\}$	ac
$\delta^*(s_a, +2)$	$\{ab, out\}$	ab
$\delta^*(s_a, -1)$	$\{out\}$	out
$\delta^*(s_a, -2)$	$\{out\}$	out
$\delta^*(s_b, +1)$	$\{out\}$	out
$\delta^*(s_b, +2)$	$\{out\}$	out
$\delta^*(s_b, -1)$	$\{ab, out\}$	ab
$\delta^*(s_b, -2)$	$\{ab, bc, in, out\}$	$\perp$
$\delta^*(s_b, -2\ 0)$	$\{ab, in\}$	ab
$\delta^*(s_b, -2\ 1)$	$\{bc, in, out\}$	bc
$\delta^*(s_c, +1)$	$\{out\}$	out
$\delta^*(s_c, +2)$	$\{bc, in, out\}$	bc
$\delta^*(s_c, -1)$	$\{ac, in, out\}$	ac
$\delta^*(s_c, -2)$	$\{out\}$	out

Since the canonical basis of the complementary vector spaces of the states s_{ab}, s_{bc} and s_{ac} is $\{\vec{e}_1\}$, Figure 5.9 can be used to define their successors. Indeed, the x_1 axis is there represented, making it quite easy to determine, for each plane, if leaving this plane from any of its points in a positive direction. And finally, here is the result of the exploration of the successors of states s_{ab}, s_{bc} and s_{ac} :

word	components	min. component
$\delta^*(s_{ab}, +1)$	$\{in\}$	in
$\delta^*(s_{ab}, -1)$	$\{out\}$	out
$\delta^*(s_{bc}, +1)$	$\{out\}$	out
$\delta^*(s_{bc}, -1)$	$\{in\}$	in
$\delta^*(s_{ac}, +1)$	$\{in\}$	in
$\delta^*(s_{ac}, -1)$	$\{out\}$	out

This exploration teaches us that we need five explicit states. We have :

$$S_E = \{s_1, s_2, s_3, s_4, s_5\}$$

And, in summary, we have this complete definition of the transition function :

δ	$+1$	$+2$	$+3$	-1	-2	-3	0	1
s_0	s_1	s_b	s_a	s_{out}	s_{out}	s_{out}		
s_a	s_4	s_{ab}		s_{out}	s_{out}			
s_b	s_{out}	s_{out}		s_{ab}	s_5			
s_c	s_{out}	s_{bc}		s_{out}	s_{ac}			
s_{ab}	s_{out}			s_{in}				
s_{bc}	s_{in}			s_{out}				
s_{ac}	s_{out}			s_{in}				
s_1							s_c	s_2
s_2							s_{out}	s_3
s_3							s_c	s_b
s_4							s_{ac}	s_{ab}
s_5							s_{ab}	s_{bc}

Figure 5.10 shows the graph representation of this IRVA. From now on in this work, representations of IRVA will use the following conventions. Implicit states are represented with rounded boxes and explicit states are ovals. Name and associated vector space of each implicit state is included inside each rounded boxes, and color is shown by the color of the box. The vector spaces are presented in a matrix-style notation. Each column of the matrix is a basis vector and are separated by a vertical line. The set of vectors on the left of the vertical line is the basis of the vector space associated to the implicit state. The canonical complementary vector space is generated by the set of vectors a the right of the vertical line. Although this latter set of vectors can be deduced from the associated vector space itself, it is shown in order to facilitate the reading of encodings of points in the outgoing decision structure, as the encodings are expressed inside this canonical complementary vector space.

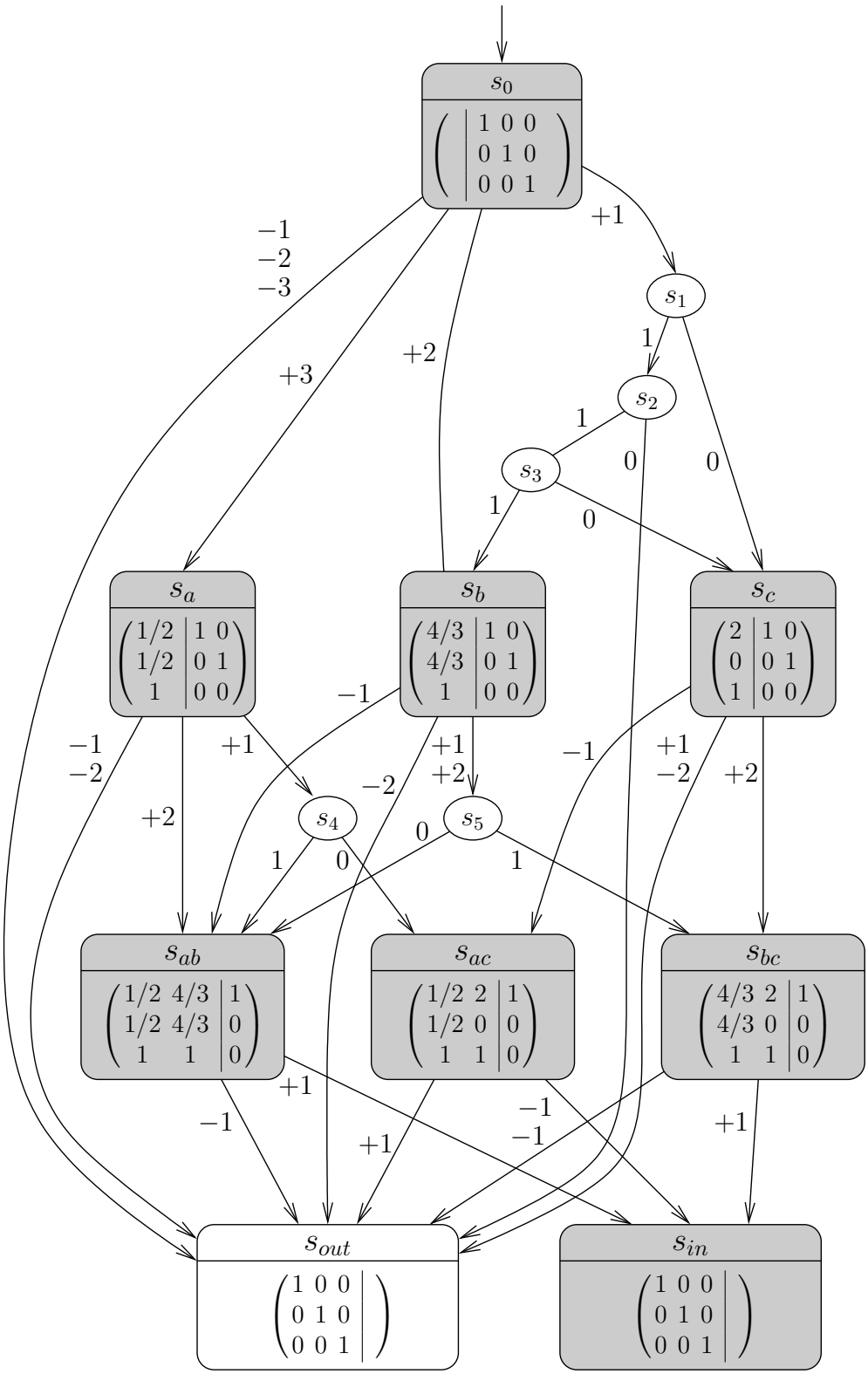


Figure 5.10: IRVA representing the *abc* pyramid.

Chapter 6

Canonical IRVA

6.1 Motivation

One of the biggest advantages of the IRVA data structure is that it admits a canonical form. For particular applications, such a property can be essential :

First, a canonical structure ensures that its size does not depend on the history of its creation. Keeping the size of the data structure under control is critical in many fields. For example, in symbolic state-space exploration, building a polyhedron that is a representation of the reachable state-space of a program is done incrementally by analyzing its model. Data structures such as B-REP or CSG can be problematic in this regard [Sha97].

Second, having a canonical representation of sets leads to an efficient test for equality, which reduces to a simple syntactic equivalence check.

6.2 Canonical IRVA Representation

We will now show that any pyramidal partition of $\mathbb{R}^n$ (with $\vec{0}$ as apex), can be represented by an IRVA, and this IRVA can be defined canonically up to isomorphism of transition relation and equality of vector spaces.

Let $\Pi \subseteq \mathbb{R}^n$ be a pyramidal partition of $\mathbb{R}^n$ with apex $\vec{0}$. To construct a canonical IRVA $(n, S_I, S_E, s_0, \delta, \mathrm{VS}, \mathrm{col})$ representing Π, a first step consists in defining S_I as a canonical set of implicit states. For each polyhedral component C_i of Π, we define an implicit state s_i, such that $\mathrm{VS}(s_i) = \mathrm{aff}(C_i)$ and $\mathrm{col}(s_i)$ as the unique color associated to the points of C_i by Π.

Since Π is pyramidal, we know by Theorem 3.23 that there exists a unique minimum component with respect to the incidence relation among all the components of Π. The implicit state that corresponds to this component is denoted by s_0 and forms the initial state of the canonical IRVA.

It remains to define the transition function δ and the set S_E of explicit states. The construction proceeds as follows. The successors of each state are developed recursively. For an implicit state s, if $\mathrm{VS}(s)$ is the universal vector space, then s does not have outgoing transitions. Otherwise, its successors are

developed one by one among $\Sigma(s)$. Each explicit state has always two outgoing transitions, one labeled by 0 and the other by 1.

So, for an implicit state s whose successors have to be explored, the labels in $\pm\mathbb{N}\{0,1\}^*$ are considered in increasing order of length. Each path labeled by a word w defines a path region $R_{\mathrm{VS}(s),w}$. If this path region identifies a unique minimum component of Π, then the destination of this path is an implicit state representing this component. If no such minimum component is defined, then the path leads to an explicit state, and will be developed further until reaching an implicit state.

If the paths are kept as short as possible, i.e. if no shorter prefix defines a path region that identifies a minimum component, then we have the guarantee that the decision structure leaving s is as small as possible. Since this construction is deterministic and does only depend on the associated vector space of each implicit state, it is bound to produce a canonical IRVA.

6.2.1 Minimality

The fact that the set of implicit states of the canonical IRVA $\mathcal{A}$ representing a pyramidal partition Π contains one implicit state for representing each component of Π prevents this set to be any smaller.

If, as described in the introduction of this section, the decision structure of $\mathcal{A}$ is as small as possible, it is not possible to define an IRVA $\mathcal{B}$ representing Π that contain less states than $\mathcal{A}$. The IRVA $\mathcal{A}$ is therefore minimal.

6.2.2 Sources of Non Minimality

There are two main reasons why two different IRVA can represent the same polyhedral partition :

The first one comes from the presence of useless implicit states in the structure. An implicit state is useless when it does not represent the affine closure of any polyhedral component.

For example, such a useless state can correspond to a line included in a plane of the same polarity (color). We immediately understand that an IRVA representing this set does not need to have an implicit state to represent this line.

The second reason is related to the decision structures linking implicit states. With the exception of those associated to $\mathbb{R}^n$, every implicit state is followed by a decision structure in which symbols are read until eventually reaching another implicit state. In the point decision procedure, reaching an implicit state s means that the point belongs to a certain path region small enough to identify the next current state from which to continue the procedure. This intuitively means that the point belongs to a polyhedral component that is adjacent or equal to the one associated with s.

Remember that, the point decision procedure, leaves implicit states by

following a path labeled by an encoding of the point of interest. For each symbol that is read, the corresponding path region is refined. When an implicit state is reached, we know that we are in a situation where the procedure can be continued from this implicit state. Nothing prevents a decision structure to be needlessly complex, for instance by forcing the point decision procedure to follow additional transitions even though they can only lead to a single implicit state.

Imagine the simple case of a line included in a 3-space of a different polarity (or color). The corresponding IRVA has an implicit state s representing the line and its outgoing decision structure has to assign any outgoing direction to an implicit state s' representing the 3-space. It is of course possible to define the decision structure such that it requires to follow many symbols, although only one would suffice to identify the state s' as destination.

Implicit States and Frontiers

Some implicit states have an associated vector space that does not correspond to the affine closure of a component of the represented pyramidal partition. In other words, such implicit states do not represent a component of the pyramidal partition, as presented in section 6.2.2.

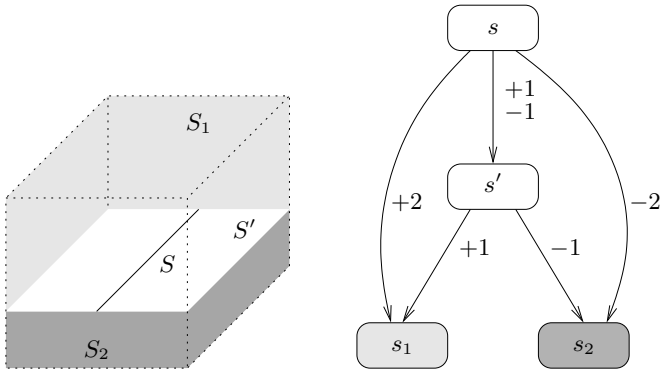
Let $\mathcal{A}$ be an IRVA representing the pyramidal partition Π. Let s and s' be implicit states of $\mathcal{A}$ such that s' is the minimum element of the set $\delta_I(s)$ of reachable implicit states from s. The implicit state s does not represent a polyhedral component of $\mathcal{A}$ iff it has the same color as s' and does not represent a frontier of the polyhedral component represented by s'.

Definition 6.1 *A polyhedral component C_1 is a **frontier** of another component C_2 when $C_1 \in \text{aff}(C_2)$ and when any neighborhood of any point of C_1 contains at least two points $p, q \in (\text{aff}(C_2) \setminus C_1)$ such that $p \in C_2$ and $q \notin C_2$.* □

This definition implies that $\text{aff}(C_1)$ is a vector space of $\text{aff}(C_2)$ such that $\dim(\text{aff}(C_1)) = \dim(\text{aff}(C_2)) - 1$.

Intuitively, we can understand that any vector space of lesser dimension than $\dim(\text{aff}(C_2)) - 1$ does not cut $\text{aff}(C_2)$ in two parts, and therefore it can not be a frontier of it.

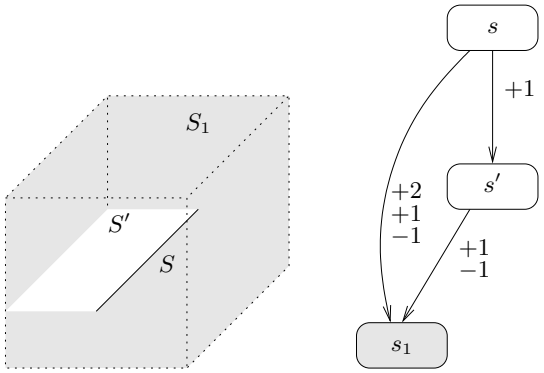
Example 6.2 *Figure 6.1(a) shows an example of a 3-dimensional polyhedral partition that contains three components : an plane S' delimiting the two half-spaces S_1 and S_2. The corresponding IRVA, shown on Figure 6.1(b), has four implicit states. The state s' represents the component S' and is such that $\dim(\text{VS}(s')) = 2$. The states s_1 and s_2 represents respectively S_1 and S_2 and are such that $\dim(\text{VS}(s_1)) = \dim(\text{VS}(s_2)) = 3$. Finally the state s represents a line S and is such that $\dim(\text{VS}(s)) = 1$. S is not a frontier of S', since the colors of s and s' are identical, and it is impossible to find any neighborhood*



(a) (b)

Figure 6.1: Illustration to the notion of useless implicit state. (a) A 3-dimensional polyhedral partition. S' is a half plane that cuts a 3-space into S_1 and S_2. S is a ray inside S' and has the same color. (b) The corresponding IRVA.

of S that contains a point that belongs to $\mathrm{aff}(C_2) \setminus C_1$ *and such that it does not belong to C_2. Intuitively, the line S is completely included inside the plane S'.* ◇



(a) (b)

Figure 6.2: Illustration of the notion of frontier. (a) Example of 3-dimensional pyramidal partition. S' is a half plane inside a 3-space S_1. S is the frontier of the component S' and has the same color. (b) The corresponding IRVA.

Example 6.3 *Figure 6.2(a) shows a 3-dimensional polyhedral partition composed of three polyhedral components. A half-plane S' is bounded by a line S and the 3-space S_1 surrounds them. Figure 6.2(b) shows the corresponding*

IRVA. The main difference with Example 6.2, except that as S' is only a half plane, is that any neighborhood of any point of the line S contains at least two points $p, q \in (\text{aff}(S') \setminus S)$ such that $p \in S'$ and $q \notin S'$. Intuitively, we see that the line S does partition the plane S'. ◇

Detecting Useless Implicit States

Operationally, an implicit state s is a useless state of an IRVA $\mathcal{A} = (n, S_I, S_E, s_0, \delta, \text{VS}, \text{col})$ iff all the following conditions are satisfied :

- $(\exists s' \in S_I)(s' = \textbf{minelset}(\delta_I(s)))$,
- $\text{col}(s) = \text{col}(s')$,
- $(\dim(\text{VS}(s)) = \dim(\text{VS}(s')) - 1) \Rightarrow$
  $(\exists w_1, w_2 \in \{0,1\}^*)(\delta^*(s, +iw_1) = s' \wedge \delta^*(s, -iw_2) = s')$

with $i \in \{1, \ldots, n - \dim(\text{VS}(s))\}$.

The last condition is aimed at detecting whether or not s is a frontier of s'. To understand it, we will use a few examples. Consider a situation that satisfies the first two conditions : s and s' are two implicit states having the same color and such that s' is the minimum element of $\delta_I(s)$. If $\dim(\text{VS}(s)) < \dim(\text{VS}(s')) - 1$ then s is not a frontier of s', hence s is detected useless.

Now let $\dim(\text{VS}(s)) = \dim(\text{VS}(s')) - 1$ and w be a path linking s to s'. w begins with the face selection symbol $+i$ (or $-i$), with $i \in \{1, \ldots, n - \dim(\text{VS}(s))\}$. If there do not exist a path w' linking s to s' beginning with the face selection symbol $-i$ (or $+i$), s is a frontier of s'. Indeed, the presence of the path labeled by w leading to s' ensures that there exists at least one point p that belongs to $R_{\text{VS}(s),w}$ that also belongs to the polyhedral component represented by s'. We know that one encoding of p begins with $+i$ (or $-i$). If s was not a frontier of s', there would exist at least one point q in the opposite direction of p, with respect to $\text{VS}(s)$, such that an encoding of q belongs to the component represented by s'. This is not the case, as there is no path beginning with $-i$ (or $+i$) in the outgoing decision structure of s. Therefore, s is a frontier of s'.

Detecting Useless Explicit States

We have explained that some paths of the decision structure can be overly complex. If each path leading to an explicit state s defines a path region in which a minimum covered element is defined, s is useless.

Operationally, an explicit state s is useless when the set $\delta_I(s)$ of implicit states reachable from s, contains a minimum element.

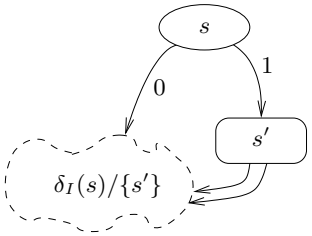


Figure 6.3: Illustration of a useless explicit state, s' is the minimum element of the set $\delta_I(s)$.

6.3 Minimization Algorithm

We address the question of computing, for a given IRVA, a canonical IRVA that represents the same set. Since it turns out that this canonical form has also the smallest possible number of states as discussed in Section 6.2.1, we call this operation *minimization*.

Let $\mathcal{A} = (n, S_I, S_E, s_0, \delta, \mathrm{VS}, \mathrm{col})$ be an IRVA. The computation of the canonical form of $\mathcal{A}$ representing the same polyhedral partition can be performed as follows.

Each step of the algorithm consists in processing one state of the automaton, considered in an a bottom-up order. Recall that the transition function of IRVA is acyclic. This means that the first states to be processed are the implicit states with an associated vector space of dimension n, since they correspond to the terminal states of the IRVA. Then, every state whose direct successors by δ have been already considered is processed. The algorithm ends when every state of the automaton has been visited.

Processing a state $s \in S_I \cup S_E$ consists in identifying states s' that have already been processed, such that s can be merged with s'. If no such state is found then s is left unchanged. When a state s' is found and merged with s, all transitions leading to s have to be redirected to s' and all transitions outgoing from s are removed from δ. Some states could become unreachable by this operation and have to be removed.

6.3.1 Merging Rules

It remains to determine under which conditions two states are considered identical and have to be merged. Let s be the state that is currently processed, and let s' be a state already processed in a previous step. There are two rules for the case where s is explicit and two rules for the case where s is implicit.

Rules for explicit states

1. A first rule checks whether s' has the same successors as s. This rule makes sure that the decision structure itself is minimized. It is similar

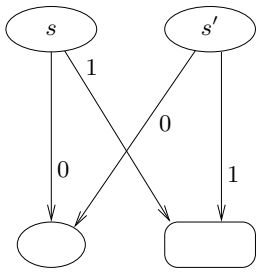


Figure 6.4: First rule for explicit states, s and s' share the same successors.

to the minimization algorithms for Binary Decision Diagrams [Ake78, Bry92].

More precisely, s and s' have to be merged iff :

$$\delta(s,0) = \delta(s',0) \wedge \delta(s,1) = \delta(s',1)$$

Figure 6.4 shows such a situation.

2. A second rule for explicit state s is aimed at reducing the size of a possibly overly complex decision structure, as mentioned in Section 6.2.2. The rule consists in checking if s' is the minimum element of the set $\delta_I(s)$ of reachable implicit states from s. Remember that for s' to be the minimum element of a set S of implicit states, one only has to check whether each state in S belongs to $\delta^*(s')$, i.e. that this state is reachable from s'. We obtain the following condition :

$$(\forall s'' \in \delta^*(s) \setminus \{s\})(s'' \in \delta^*(s'))$$

If this condition holds, then the destination of any transition that leads to s can safely be redirected to s', since paths leading to s have s' as minimum covered element defined.

Figure 6.3 shows such a situation.

Rules for implicit states

1. The first rule for implicit states is similar to the first rule for explicit states. It is used to find potential duplicates among represented pyramids inside the IRVA. The only difference lies in the fact that when another implicit state s' has been identified as sharing the same successors as s, one still has to check whether they share the same color and the same vector space. Otherwise, the two states can not be considered equal because they do not represent the same local pyramid of Π. More formally, we have :

$$(\forall \sigma \in \Sigma(s))(\delta(s,\sigma) = \delta(s',\sigma) \wedge \mathrm{col}(s) = \mathrm{col}(s') \wedge \mathrm{VS}(s) = \mathrm{VS}(s'))$$

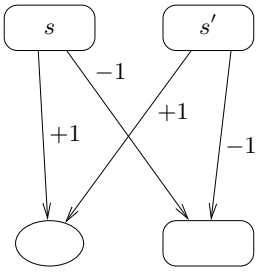


Figure 6.5: First rule for implicit states, s and s' shares the same successors and have identical associated colors and vector spaces (not represented here).

If this condition is satisfied, then s is merged with s'. Figure 6.5 shows such a situation.

2. The second and last rule for implicit states is aimed at finding useless implicit states (whose associated vector spaces do not correspond to the affine closure of a component of Π), as previously explained.

   We obtain that s is is merged with another implicit state s', when

   - $(\exists s' \in S_I)(s' = \mathbf{minelset}(\delta_I(s)))$,
   - $\mathrm{col}(s) = \mathrm{col}(s')$,
   - $(\dim(\mathrm{VS}(s)) = \dim(\mathrm{VS}(s')) - 1) \Rightarrow$
     $(\exists w_1, w_2 \in \{0,1\}^*)(\delta^*(s, +iw_1) = s' \wedge \delta^*(s, -iw_2) = s')$

   with $i \in \{1, \ldots, n - \dim(\mathrm{VS}(s))\}$.

   Figure 6.1 shows such a situation.

We can be convinced that this algorithm is correct by two observations.

First, we have identified all situations in which an IRVA is not canonical. We have seen that some explicit states can be identical with respect to their successors by the transition relation. There is no reason to keep more than one of them.

We have also seen a similar case for implicit states having the same outgoing transitions leading to the same states, the same vector space and the same color.

We have characterized a criterion that permits to identify useless explicit states. Such a situation arises when a decision path is unnecessarily long.

Finally, we have seen another criterion that identifies useless implicit states with respect to the represented pyramidal partition of the IRVA. They correspond to implicit states that do not represent any polyhedral components.

Second, the algorithm has a rule to detect occurrences of each of these four situations. It deals with those by merging duplicated states, or by redirecting transitions leading to useless states to the states that are the cause of their uselessness.

6.4 Example of Minimization

To illustrate further the mechanics, we will now see a complete example of the minimization of an 2-dimensional IRVA $(n, S_I, S_E, s_0, \delta, \mathrm{VS}, \mathrm{col})$, defined as follows :

- $n = 2$
- $S_I = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}$
- $S_E = \{s_7, s_8\}$
- s_0 is the initial state
- δ is given by :

δ	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
$+1$	s_7	s_5	s_6	s_6					
-1	s_4	s_4	s_5	s_4					
$+2$	s_1								
-2	s_4								
0								s_8	s_2
1								s_3	s_3

- VS is given by :

$\mathrm{VS}(s_0)$	$\emptyset$
$\mathrm{VS}(s_1)$	$\{(0, 1)\}$
$\mathrm{VS}(s_2)$	$\{(1, 3/4)\}$
$\mathrm{VS}(s_3)$	$\{(1, 0)\}$
$\mathrm{VS}(s_4)$	$\{(1, 0), (0, 1)\}$
$\mathrm{VS}(s_5)$	$\{(1, 0), (0, 1)\}$
$\mathrm{VS}(s_6)$	$\{(1, 0), (0, 1)\}$

- col is given by :

$\mathrm{col}(s_0)$	$\mathrm{col}(s_1)$	$\mathrm{col}(s_2)$	$\mathrm{col}(s_3)$	$\mathrm{col}(s_4)$	$\mathrm{col}(s_5)$	$\mathrm{col}(s_6)$
`gray`	`gray`	`gray`	`gray`	`white`	`gray`	`gray`

Figure 6.6(a) shows a graph representation of this IRVA. The represented pyramidal partition is given in figure 6.6(b). It consists in three half-lines s_1, s_2 and s_3 incident to the origin s_0. They partition the space into s_4, s_5 and s_6. Notice that the polyhedral partition represented by this IRVA is simpler, because some implicit states are useless.

The first phase of the algorithm consists in attributing an order to the set of states, to determine the state that is going to be processed at each step. As there are nine states, the algorithm will execute nine steps. The steps will be numbered from 1 to 9.

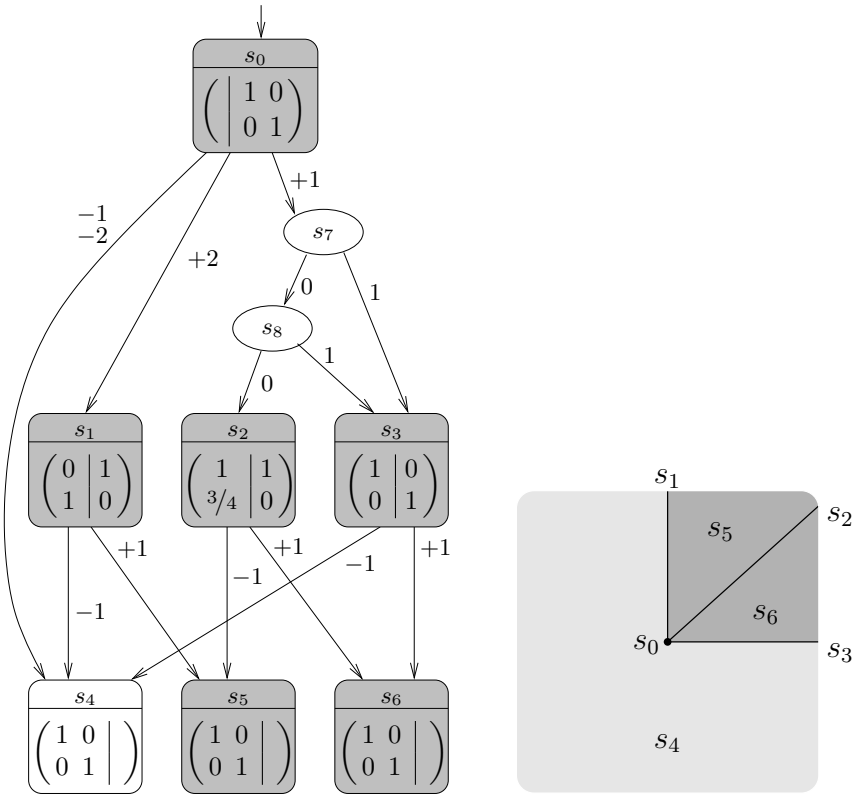


Figure 6.6: (a) Graph representation of an IRVA to be minimized (b) A geometrical illustration of it.

The first batch of states to be be processed are the terminal states s_4, s_5 and s_6 of the automaton. Then, the states s_1,s_2,s_3 are processed, since they have outgoing transitions leading only to those terminal states. The next state that can be processed is s_8, followed by s_7. Finally, the last step will process the initial state s_0.

In summary, a possible ordering of the states for the steps of the algorithm is the following :

step	1	2	3	4	5	6	7	8	9
state	s_4	s_5	s_6	s_1	s_2	s_3	s_8	s_7	s_0

Now let us analyze each step of the minimization algorithm.

Step 1 : s_4 is unchanged.

Step 2 : s_5 is unchanged since $\text{col}(s_5) \neq \text{col}(s_4)$.

Step 3 : s_6 is merged with s_5, by the first rule of implicit states. Indeed, they have the same successors (none), the same associated vector space and the same color (`gray`).

Step 4 : s_1 is unchanged.

Step 5 : s_2 is merged with s_5, by the second rule for implicit states. The set of implicit states reachable from s_2 is $\{s_5\}$. Therefore, s_5 is the minimum element of this set. s_2 and s_5 have the same color (`gray`) and s_2 is not a frontier of s_5.

Step 6 : s_3 is unchanged.

Step 7 : s_8 is merged with s_3 by the second rule for explicit states. At that point, the set of implicit states reachable from s_8 is $\{s_3, s_5\}$, from which s_3 is the minimum element.

Step 8 : s_7 is merged with s_3 by the second rule for explicit states. The set of reachable implicit states from s_8 is $\{s_3\}$, s_3 is the minimum element.

Step 9 : s_0 is unchanged.

The minimized form of the automaton is thus :

- $n = 2$
- $S_I = \{s_0, s_1, s_3, s_4, s_5\}$
- $S_E = \emptyset$
- s_0 is the initial state
- δ is given by :

δ	s_0	s_1	s_3	s_4	s_5
$+1$	s_3	s_5	s_5		
-1	s_4	s_4	s_4		
$+2$	s_1				
-2	s_4				

- VS and col are unchanged.

Figure 6.7 shows a graph representation of the minimized IRVA.

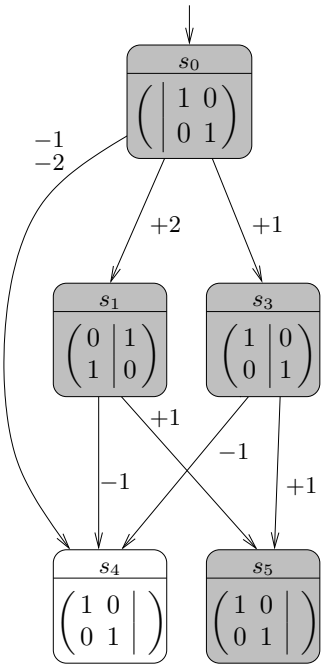


Figure 6.7: Resulting minimized IRVA.

Chapter 7

Boolean Combinations of IRVA

7.1 Motivation

In this chapter, we address the problem of computing an IRVA representing a Boolean combination of sets represented by IRVA. The motivation behind this operation is to be able to construct complex polyhedra as the result of combining simple ones. We start by showing how to construct IRVA corresponding to elementary sets, and then develop an algorithm for performing various Boolean combinations of IRVA representing arbitrary polyhedra or polyhedral partitions.

7.2 Primitive Elements

7.2.1 Half-Spaces

Every polyhedron, hence every pyramid, can be expressed as a finite disjunction of finite conjunctions of linear inequalities (see Definition 3.1).

Except for the particular case of $\vec{0} \cdot \vec{x} < \lambda$, with $\lambda \in \mathbb{R}$, the set of solutions of every linear inequality is a set of points bounded by a hyperplane. In the general context of pyramidal partitions, we can see a linear inequality as a coloring function. Indeed, it is possible to associate a color to all points that satisfy the inequation, another one to points that do not, and the last one to points that belong to the hyperplane.

Given the inequation $\vec{a} \cdot \vec{x} \# 0$, with $\vec{a} \neq \vec{0}$ and $\# \in \{\leq, <, >, \geq\}$ and three colors c_1, c_2 and c_3, let us see how to build an IRVA representing the set of solutions of this inequation.

The represented pyramidal partition is composed of three components : the delimiting hyperplane H, and two $n-$spaces P and M, each corresponding to

one side of the hyperplane. Formally :

$$H : \vec{a} \cdot \vec{x} = 0$$
$$P : \vec{a} \cdot \vec{x} > 0$$
$$M : \vec{a} \cdot \vec{x} < 0$$

Note that H is a vector space.

The IRVA representing this set is thus composed of three implicit states s_h, s_p and s_m, with s_h being the initial state.

The vector spaces associated to those states are respectively H, $\mathbb{R}^n$ and $\mathbb{R}^n$.

The transition function is very simple to define, since the exit space from s_h has dimension 1. From s_h, a single symbol $+1$ or -1 suffices to classify any direction. The destination by $+1$ from s_h is s_p and the one by -1 is s_m. The three implicit states are then colored as needed.

Example 7.1 *We show the construction of an IRVA based on the following linear inequallity on* $\mathbb{R}^3$ *:*

$$-\frac{x_1}{2} + x_3 \geq 0$$

and three colors, `gray` *for points satisfying the inequation,* `gray` *for points inside the bounding hyperplane and* `white` *for points that do not satisfy the inequation. Figure 7.1 shows the corresponding IRVA.* ◇

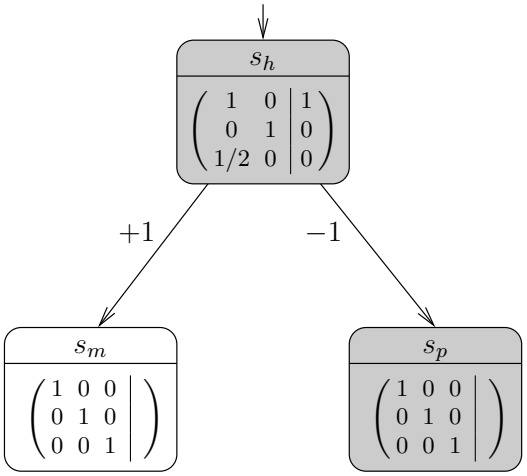


Figure 7.1: IRVA corresponding to $-\frac{x_1}{2} + x_3 \geq 0$.

7.2.2 Vector Spaces

Here, we show how to build an IRVA representing a pyramidal partition that corresponds to a given vector space V of color c_1. Its exterior points having the color c_2.

If $V = \mathbb{R}^n$, then the IRVA is composed of a single implicit state s with $\mathrm{VS}(s) = \mathbb{R}^n$ that has the color c_1.

Otherwise, the IRVA has two implicit states : s_1 such that $\mathrm{VS}(s_1) = V$ and s_2 such that $\mathrm{VS}(s_2) = \mathbb{R}^n$. The colors of s_1 and s_2 are respectively c_1 and c_2. The implicit state s_1 is the initial state of the IRVA.

The transition relation is very simple, since all outgoing paths from s_1, each labeled by a symbol in $\Sigma(s_1)$, lead to s_2.

Example 7.2 *Figure 7.2 shows the IRVA constructed from the vector space*

$$\{1, 1/2, 1/2\}$$

and two colors `gray` *for points inside the vector space and* `white` *for points outside of it.* ◇

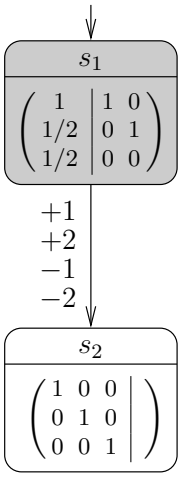


Figure 7.2: IRVA corresponding to the vector space $\{1, 1/2, 1/2\}$.

7.3 Coloring Boolean Combinations

We have observed in Section 3.2 that a polyhedral partition Π can be seen as a function of the form $\mathbb{R}^n \to \{1, 2, \ldots, m\}$ that associates to each point of $\mathbb{R}^n$ a color out of m possible choices. We have also seen in Section 5.3.4 that, unsurprisingly, an IRVA representing a pyramidal partition can also be seen as such a coloring function.

As a consequence, in order to define a Boolean combination of two pyramidal partitions Π_1 and Π_2, we need a coloring function of the form

$$\{1, 2, \ldots, m_1\} \times \{1, 2, \ldots, m_2\} \to \{1, 2, \ldots, m\}$$

where m_1 and m_2 are the number of colors of, respectively, Π_1 and Π_2 and m is the number of colors of the result. We obtain the following definition.

Definition 7.3 *Over $\mathbb{R}^n$, the combination of two pyramidal partitions Π_1 and Π_2 induced by the coloring function c is the pyramidal partition $\Pi = \Pi_1 \ltimes_c \Pi_2$ such that, for every $\vec{x} \in \mathbb{R}^n$, $\Pi(\vec{x}) = c(\Pi_1(\vec{x}), \Pi_2(\vec{x}))$.* □

Remark that in the particular case where only two colors are used, classical Boolean operations on polyhedra such as union, intersection or difference can be defined by an appropriate choice of coloring function.

(a) (b)

Figure 7.3: The operands of Example 7.4 : (a) The pyramid $\Pi_1 \subset \mathbb{R}^2$, and (b) $\Pi_2 \subset \mathbb{R}^2$.

Example 7.4 *We illustrate two combinations, of pyramidal partitions Π_1 and Π_2, in Figures 7.3(a) and (b). In those two pyramidal partitions, only two colors are used :* `white` *and* `gray`.

We consider two coloring functions $c_\cup$ and $c_\cap$ of the form

$$\{\texttt{white}, \texttt{gray}\} \times \{\texttt{white}, \texttt{gray}\} \to \{\texttt{white}, \texttt{gray}\}$$

such that

$$c_\cup(c_1, c_2) = \begin{cases} \texttt{gray} & \textit{if } (c_1 = \texttt{gray}) \vee (c_2 = \texttt{gray}) \\ \texttt{white} & \textit{otherwise} \end{cases}$$

$$c_\cap(c_1, c_2) = \begin{cases} \texttt{gray} & \textit{if } (c_1 = \texttt{gray}) \wedge (c_2 = \texttt{gray}) \\ \texttt{white} & \textit{otherwise} \end{cases}$$

The combinations $\Pi_1 \ltimes_{c_\cup} \Pi_2$ and $\Pi_1 \ltimes_{c_\cap} \Pi_2$ are in Figure 7.4. ◇

Figure 7.4: Result of the combinations (a) $\Pi_1 \ltimes_{c_\cup} \Pi_2$ (b) $\Pi_1 \ltimes_{c_\cap} \Pi_2$.

7.4 Boolean Combination Algorithm

In this section, we present how to compute Boolean combinations of IRVA.

In order to keep the algorithm as general as possible, we will compute the product of two IRVA without making any hypothesis on the final colors, and then apply the coloring function to decide the color that has to be associated to each implicit state of the product IRVA.

This product operation over IRVA shares some similarities with the product of finite-state automata, in the sense that the product of two IRVA is an IRVA that simulates the concurrent behavior of the operands.

The product becomes a combination operation when the coloring function is used in order to assign a color to the implicit states of the output.

But first, we recall the notion of minimum covered component.

7.4.1 Minimum Covered Component

A key operation in the product algorithm developed in Section 7.4.2 is the search for a minimum covered element in an IRVA $\mathcal{A}$, from an implicit state s by a region R. This operation is noted :

$$\textbf{minel}(\mathcal{A}, R, s)$$

See Sections 3.3 for the presentation of the problem applied to polyhedra and pyramidal partitions pyramidal partitions and Section 5.4.1 for a description of an algorithm that solves the problem applied to the represented pyramid of a given IRVA.

7.4.2 Product Algorithm

We will now develop an algorithm that calculates the IRVA $\mathcal{P}$ representing $\Pi = \Pi_1 \ltimes_c \Pi_2$, given two IRVA $\mathcal{A}_1$ and $\mathcal{A}_2$ representing respectively Π_1 and Π_2. Each IRVA is defined as :

- $\mathcal{P} = (n,\ S_I,\ S_E,\ (s_{01}, s_{02}),\ \delta,\ \mathrm{VS},\ \mathrm{col}\ )$
- $\mathcal{A}_1 = (n, S_{I1}, S_{E1}, s_{01}, \delta_1, \mathrm{VS}_1, \mathrm{col}_1)$
- $\mathcal{A}_2 = (n, S_{I2}, S_{E2}, s_{02}, \delta_2, \mathrm{VS}_2, \mathrm{col}_2)$

Let us first make some observations. Any point $\vec{x} \in \mathbb{R}^n$ belongs to one component of Π_1 and one component of Π_2, which we name respectively C_1 and C_2. Two points associated to the same pair of components will belong to the same polyhedral component C of Π, since their situations are identical. Furthermore, the component is such that $C = C_1 \cup C_2$. In other words, the product associates a pair of components to each point of space. The color of the components of Π are obtained by applying the coloring function c to the corresponding pair of components.

Having that in mind, each implicit state of $\mathcal{P}$ corresponds to a composite state (s_1, s_2) of implicit states with $s_1 \in S_{I1}$ and $s_2 \in S_{I2}$. If C_1 and C_2 are respectively the polyhedral components associated to s_1 and s_2, then the polyhedral component represented by (s_1, s_2) is composed of the points $\vec{x} \in \mathbb{R}^n$ such that $\vec{x} \in C_1 \cap C_2$. From that, we obtain $\mathrm{VS}((s_1, s_2)) = \mathrm{VS}_1(s_1) \cap \mathrm{VS}_2(s_2)$.

The IRVA $\mathcal{P}$ can be constructed incrementally. Each major step of the algorithm consists in taking a composite implicit state (s_1, s_2) out of a list $\mathcal{L}$, building its outgoing decision structure and creating new implicit states when needed. The list $\mathcal{L}$ is maintained during the execution of the algorithm in order to identify the states for which the outgoing decision structure still needs to be built.

The algorithm begins with the creation of the initial state (s_{01}, s_{02}) with

$$\mathrm{VS}((s_{01}, s_{02})) := \mathrm{VS}_1(s_{01}) \cap \mathrm{VS}_2(s_{02})$$
$$\mathrm{col}((s_{01}, s_{02})) := c(\mathrm{col}(s_{01}), \mathrm{col}(s_{02}))$$

If $\dim(\mathrm{VS}((s_{01}, s_{02}))) = n$, then the algorithm terminates. Otherwise, (s_{01}, s_{02}) is added to the list of states that need to be processed.

Let us now study in detail how the decision structures are built and how implicit states are created during the product computation.

Let (s_1, s_2) be an implicit state for which the outgoing decision structure needs to be developed and ρ_1, ρ_2 be the pyramidal partitions represented respectively from s_1 and s_2. One enumerates all the possible prefixes of words $w \in \pm\mathbb{N}\{0,1\}^*$ labeling paths from (s_1, s_2) in breadth-first order. This enumeration begins with paths of length one, then two, and so on. For every such prefix w, one checks whether it leads to an implicit state of $\mathcal{P}$.

This test is carried out by first computing the path region $R_{\mathrm{VS}((s_1,s_2)),w}$. Then one searches for the minimum covered components ${C_1}'$ of ρ_1 and ${C_2}'$ of ρ_2 by $R_{\mathrm{VS}((s_1,s_2)),w}$. If one of the components is (or both are) undefined, it means that the prefix w is too short to make a decision in both operands and must be expanded further. Otherwise, m_1 and m_2 be the implicit states associated respectively to ${C_1}'$ and ${C_2}'$. A potential implicit state associated to $C_1 \cap C_2$ can be the destination of the prefix w, provided that two conditions are satisfied. First the dimension of the affine closure of $C_1 \cap C_2$ must be greater than the dimension of the component of Π associated to (s_1, s_2). This can be checked by testing the constraint

$$\dim(\mathrm{VS}((s_1, s_2))) < \dim(\mathrm{VS}_1(m_1) \cap \mathrm{VS}_2(m_2)).$$

Additionally, the polyhedral component ${C_1}' \cap {C_2}'$ must have a non empty intersection with $R_{\mathrm{VS}((s_1,s_2)),w}$. If any of those two conditions is not met, then the prefix w must be expanded further. To clarify, those conditions represent the fact that, in order to be a valid destination from (s_1, s_2) by a prefix w in $\mathcal{P}$, an implicit state (m_1, m_2) must represent the minimum covered component of Π by $R_{\mathrm{VS}((s_1,s_2)),w}$. If both conditions are satisfied and if the implicit state (m_1, m_2) exists in $\mathcal{P}$, then the destination of w becomes (m_1, m_2). If such a state does not yet exist, a new implicit state (m_1, m_2) is created and added to S_I. The associated vector space of (m_1, m_2) is $\mathrm{VS}_1(m_1) \cap \mathrm{VS}_2(m_2)$ and its color is $c(\mathrm{col}_1(m_1), \mathrm{col}_2(m_2)$. In both cases, all the necessary explicit states needed to reach (m_1, m_2) from (s_1, s_2) by w are created. See Algorithm 2 for a formal definition. Note that $\mathcal{P}$ is generally not minimal.

7.4.3 Complexity

Analyzing completely the theoretical worst case complexity of the product algorithm makes little sense for two reasons :

- The algorithm relies on heavy use of path regions and algebraic manipulations. The complexity of the manipulation operators influences the effective complexity of the product algorithm itself. This complexity can vary greatly with design choices, such as whether or not exact arithmetic is used.

- We expect this worse-case complexity to be avoidable in most practical situations, as in the case of RVA [WB00].

The product operation is the incremental construction of $\mathcal{P}$, the product IRVA of two operands $\mathcal{A}_1$ and $\mathcal{A}_2$, from its origin. Obviously, enumerating all the states of $\mathcal{P}$, $\mathcal{A}_1$ and $\mathcal{A}_2$ is not avoidable.

More precisely, the product algorithm builds implicit states and develops their outgoing decision structures. Paths inside this structure are as short as

input : Two IRVA $\mathcal{A}_1 = (n, S_{I1}, S_{E1}, s_{01}, \delta_1, \mathrm{VS}_1, \mathrm{col}_1)$ and
$\mathcal{A}_2 = (n, S_{I2}, S_{E2}, s_{02}, \delta_2, \mathrm{VS}_2, \mathrm{col}_2)$, a combination function c
output: An IRVA $\mathcal{P} = (n, S_I, S_E, s_0, \delta, \mathrm{VS}, \mathrm{col})$

$s_0 := (s_{01}, s_{02})$;
$S_I := \{s_0\}$;
$S_E := \emptyset$;
$\mathrm{VS}(s_0) := \mathrm{VS}_1(s_{01}) \cap \mathrm{VS}_2(s_{02})$;
$\mathrm{col}(s_0) := c(\mathrm{col}_1(s_{01}), \mathrm{col}_2(s_{02}))$;
$stack := \emptyset$;
`push` $(stack, (s_{01}, s_{02}, s_0, s_0, \varepsilon))$;
while `not`(`empty?`$(stack)$) **do**
  (s_1, s_2, s, s_I, w) := `pop`$(stack)$;
  foreach $a \in$ `succ`(s) **do**
    $m_1 :=$ **minel**$(\mathcal{A}_1, R_{\mathrm{VS}(s_I),wa}, s_1)$;
    $m_2 :=$ **minel**$(\mathcal{A}_2, R_{\mathrm{VS}(s_I),wa}, s_2)$;
    if $m_1 \neq \perp$ `and` $m_2 \neq \perp$ **then**
      $R := \mathrm{VS}_1(m_1) \cap \mathrm{VS}_2(m_2)$;
      if $R \cap R_{\mathrm{VS}(s_I),wa} = \emptyset$ `or` $\dim(\mathrm{VS}(s_I)) \geq \dim(R)$ **then**
        $s_N :=$ `new`();
        $S_E := S_E \cup \{s_N\}$;
        $\delta(s, a) := s_N$;
        `push`$(stack, (m_1, m_2, s_N, s_I, wa))$;
      else
        if $(m_1, m_2) \notin S_I$ **then**
          $S_I := S_I \cup \{(m_1, m_2)\}$;
          $\mathrm{VS}((m_1, m_2)) := R$;
          $\mathrm{col}((m_1, m_2)) := c(\mathrm{col}_1(m_1), \mathrm{col}_2(m_2))$;
          `push`$(stack, (m_1, m_2, (m_1, m_2), (m_1, m_2), \varepsilon))$;
        end
        $\delta(s, a) := (m_1, m_2)$;
      end
    else
      if $m_1 = \perp$ **then** $m_1 := s_1$;
      if $m_2 = \perp$ **then** $m_2 := s_2$;
      $s_N :=$ `new`();
      $S_E := S_E \cup \{s_N\}$;
      $\delta(s, a) := s_N$;
      `push`$(stack, (m_1, m_2, s_N, s_I, wa))$;
    end
  end
end

Algorithm 2: Computation of $\mathcal{A}_1 \ltimes_c \mathcal{A}_2$

possible. This means that as soon as a path can identify an implicit state as destination, it leads to this implicit state.

Each implicit state (s_1, s_2) of the product IRVA is constructed by combining the implicit state s_1 from operand $\mathcal{A}_1$ and s_2 from operand $\mathcal{A}_2$. When trying to figure out the destination of a path w from (s_1, s_2) in the product automaton, the path region $R = R_{VS((s_1,s_2),w}$ is used to find the minimum covered element, if it is defined, from s_1 in $\mathcal{A}_1$ and from s_2 in $\mathcal{A}_2$.

We see that the cost of the product algorithm depends greatly on the search for the minimum covered element.

When $\textbf{minel}(\mathcal{A}_1, R, s_1) = \perp$, the destination of the path w from s is an explicit state in the product automaton. Moreover, when a minimal element is not defined, it is due to the fact that the set of minimal states $\{m_1, \ldots, m_k\}$, computed by the procedure $\textbf{minimals}(\mathcal{A}_1, s_1, R)$, contains more than one element.

A future step of the product algorithm will eventually develop this explicit state by exploring its successors. For each successor, this results in adding a symbol σ to the path w outgoing from (s_1, s_2) and the path region $R' = R_{\mathrm{VS}((s_1,s_2)),w\sigma}$ will be computed. Note that, by definition, $R' \subset R$. The procedure $\textbf{minel}(\mathcal{A}_1, R', s_1)$ will be executed.

This new call to **minel** will call $\textbf{minimals}(\mathcal{A}_1, s_1, R')$ and does not exploit the fact the result of $\textbf{minimals}(\mathcal{A}_1, s_1, R)$ is already known. Indeed, since we know that

$$\textbf{minimals}(\mathcal{A}_1, s_1, R) = \{m_1, \ldots, m_k\}$$

we also know that

$$\textbf{minimals}(\mathcal{A}_1, s_1, R') \subseteq \{m_1, \ldots, m_k\}.$$

An incremental version of the **minimals** procedure that restarts with previous results in the kind of scenario we have explained could avoid unnecessary calculations.

Technically, the situation is a little more complex. It is tempting to call $\textbf{minimals}(\mathcal{A}_1, m_i, R')$ for each $m_i \in \{m_1, \ldots, m_k\}$ and merge the results into a set that keeps only minimal states. This is unfortunately incorrect due to the fact that in the **minimals** algorithm, each implicit state m_i of $\{m_1, \ldots, m_k\}$ that has been selected was considered by considering its vector space through a *combined path region* (see Definition 5.15) determined by the path linking s_1 to m_i.

The fact that we need this combined path region to avoid wrong results can be understood because of the fact that the vector spaces associated to implicit states of an IRVA are no representations of polyhedral components, but the affine closure of polyhedral components.

In order to define an incremental version of the **minimals** algorithm, we need to keep the previous results in the form of a set of minimal implicit states, each one associated to a set of combined path region in which it can be considered. From there, it is possible to call **minimals** for each combined

path region R_{ij} of each implicit state s_i of a previous result, by calls of the form **minimals**$(\mathcal{A}_1, m_i, R' \cap R_{ij})$ and merging of all the results into a set that keeps only minimal states.

Such tricky techniques are out of the scope of this work, but should nevertheless be investigated should the data structure be used in a practical environment relying on an efficient product computation algorithm.

7.4.4 Example of Product Computation

In this section, we illustrate in detail the computation of the Boolean combination of two IRVA.

We want to compute the intersection of two half-spaces π_1 and π_2 in $\mathbb{R}^2$. They are defined by those inequations :

$$\Pi_1 : x_1 \leq 2$$
$$\Pi_2 : x_2 \leq 1$$

The polyhedron Π_1 defines three components : the boundary a of the half-space, and the 2-spaces *in* and *out*, containing respectively the points that belong and do not belong to Π_1. The components a and *in* are `gray` and *out* is `white`.

The polyhedron Π_2 also defines three polyhedral components : the boundary b of the half-space, and again the 2-spaces *in* and *out*, containing respectively the points that belong and do not belong to Π_2. The components b and *in* are `gray` and *out* is `white`.

Figure 7.5 shows a geometrical view of Π_1 and Π_2.

To compute the intersection of Π_1 and Π_2, we will use the following coloring function :

$$c_\cap(c_1, c_2) = \begin{cases} \texttt{gray} & \text{if } (c_1 = \texttt{gray}) \wedge (c_2 = \texttt{gray}) \\ \texttt{white} & \text{otherwise} \end{cases}$$

Since they are not pyramidal with respect to $\vec{0}$, we compute their representing pyramids in $\mathbb{R}^3$. We obtain :

$$\Pi_1 : -\frac{x_1}{2} + x_3 \geq 0$$
$$\Pi_2 : -x_2 + x_3 \geq 0$$

Let $\mathcal{A}_1$ and $\mathcal{A}_2$ represent respectively Π_1 and Π_2. They are defined by the following tuples :

$$\mathcal{A}_1 = (3, \{s_a, s_{in}, s_{out}\}, \emptyset, s_a, \delta_1, \mathrm{VS}_1, \mathrm{col}_1)$$
$$\mathcal{A}_2 = (3, \{s_b, s_{in}, s_{out}\}, \emptyset, s_b, \delta_2, \mathrm{VS}_2, \mathrm{col}_2)$$

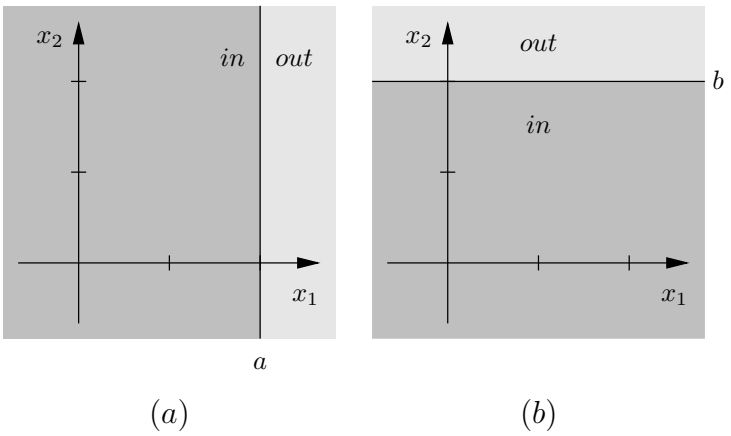


Figure 7.5: Geometrical view of (a) π_1 and (b) π_2.

Their complete definition is straightforward and are summarized by two graphs in Figures 7.6 and 7.7. The states s_{in} and s_{out} represents the polyhedral components *in* and *out*. The states s_a and s_b represent respectively the components a and b.

The IRVA $\mathcal{P} = (3, S_I, S_E, (s_a, s_b), \delta, \mathrm{VS}, \mathrm{col})$ represents the result of $\Pi_1 \ltimes_{c_i} \Pi_2$ and is built by the combination algorithm as follows.

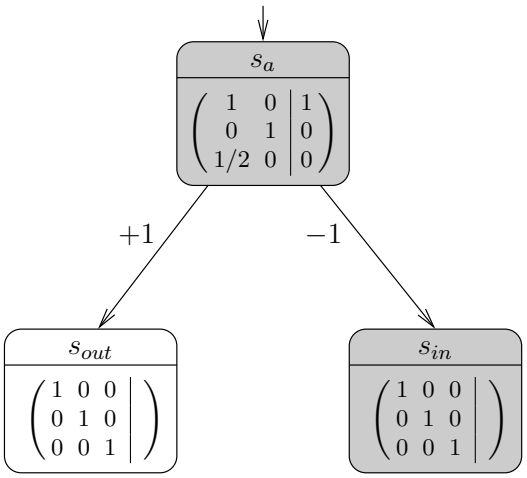


Figure 7.6: Graph representation of $\mathcal{A}_1$.

First, the initial state of $\mathcal{P}$, (s_a, s_b) is created and is added to S_I and to $\mathcal{L}$. Its associated vector space and color are :

$$\mathrm{VS}(s_c) := \mathrm{VS}_1(s_a) \cap \mathrm{VS}_2(s_b) = \{(1, 1/2, 1/2)\}$$
$$\mathrm{col}(s_c) := c_i(\mathrm{col}_1(s_a), \mathrm{col}_2(s_b)) = \texttt{gray}$$

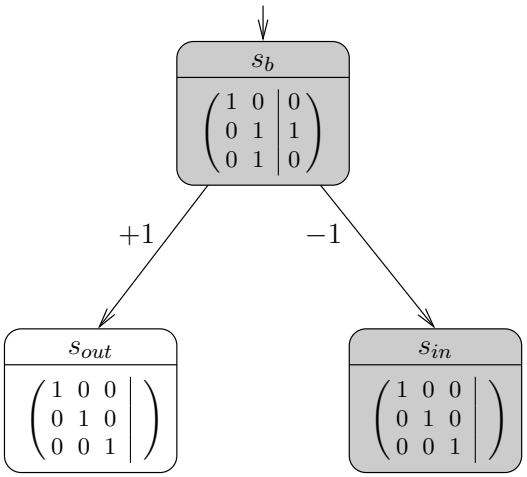


Figure 7.7: Graph representation of $\mathcal{A}_2$.

The algorithm proceeds as follows :

Step 1 : State (s_a, s_b) is removed from $\mathcal{L}$ and is developed.

Step 1.1 :

state	word	path region	min in $\mathcal{A}_1$	min in $\mathcal{A}_2$
(s_a, s_b)	$+1$	$R_{\mathrm{VS}((s_a,s_b)),+1}$	s_{out}	s_b

New implicit state : (s_{out}, s_b)
VS : $\{(1,0,0),(0,1,1)\}$
col : `white`
δ : $\delta^*((s_a, s_b), +1) := (s_{out}, s_b)$
(s_{out}, s_b) is added to $\mathcal{L}$.

Step 1.2 :

state	word	path region	min in $\mathcal{A}_1$	min in $\mathcal{A}_2$
(s_a, s_b)	$+2$	$R_{\mathrm{VS}((s_a,s_b)),+2}$	s_a	s_{out}

New implicit state : (s_a, s_{out})
VS : $\{(1,0,1/2),(0,1,0)\}$
col : `white`
δ : $\delta^*((s_a, s_b), +2) := (s_a, s_{out})$
(s_a, s_{out}) is added to $\mathcal{L}$.

Step 1.3 :

state	word	path region	min in $\mathcal{A}_1$	min in $\mathcal{A}_2$
(s_a, s_b)	-1	$R_{\mathrm{VS}((s_a,s_b)),-1}$	s_{in}	s_b

New implicit state : (s_{in}, s_b)
VS : $\{(1,0,0),(0,1,1)\}$
col : `gray`
δ : $\delta^*((s_a, s_b), -1) := (s_{in}, s_b)$
(s_{in}, s_b) is added to $\mathcal{L}$.

Step 1.4 :

state	word	path region	min in $\mathcal{A}_1$	min in $\mathcal{A}_2$
(s_a, s_b)	-2	$R_{\mathrm{VS}((s_a,s_b)),-2}$	s_a	s_{in}

New implicit state : (s_a, s_{in})
VS : $\{(1,0,1/2),(0,1,0)\}$
col : `gray`
δ : $\delta^*((s_a, s_b), -2) := (s_a, s_{in})$
(s_a, s_{in}) is added to $\mathcal{L}$.

Step 2 : State (s_{out}, s_b) is removed from $\mathcal{L}$ and is developed.

Step 2.1 :

state	word	path region	min in $\mathcal{A}_1$	min in $\mathcal{A}_2$
(s_{out}, s_b)	$+1$	$R_{\mathrm{VS}((s_{out},s_b)),+1}$	s_{out}	s_{out}

New implicit state : (s_{out}, s_{out})
VS : $\{(1,0,0),(0,1,0),(0,0,1)\}$
col : `white`
δ : $\delta^*((s_{out}, s_b), +1) := (s_{out}, s_{out})$

Step 2.2 :

state	word	path region	min in $\mathcal{A}_1$	min in $\mathcal{A}_2$
(s_{out}, s_b)	-1	$R_{\mathrm{VS}((s_{out},s_b)),-1}$	s_{out}	s_{in}

New implicit state : (s_{out}, s_{in})
VS : $\{(1,0,0),(0,1,0),(0,0,1)\}$
col : `white`
δ : $\delta^*((s_{out}, s_b), -1) := (s_{out}, s_{in})$

Step 3 : State (s_a, s_{out}) is removed from $\mathcal{L}$ and is developed.

Step 3.1 :

state	word	path region	min in $\mathcal{A}_1$	min in $\mathcal{A}_2$
(s_a, s_{out})	$+1$	$R_{\mathrm{VS}((s_a,s_{out})),+1}$	s_{out}	s_{out}

The implicit state (s_{out}, s_{out}) already exists.
δ : $\delta^*((s_a, s_{out}), +1) := (s_{out}, s_{out})$.

Step 3.2 :

state	word	path region	min in $\mathcal{A}_1$	min in $\mathcal{A}_2$
(s_a, s_{out})	-1	$R_{\mathrm{VS}((s_a,s_{out})),-1}$	s_{in}	s_{out}

New implicit state : (s_{in}, s_{out})
VS : $\{(1,0,0),(0,1,0),(0,0,1)\}$
col : `white`
δ : $\delta^*((s_a, s_{out}), -1) := (s_{in}, s_{out})$

Step 4 : State (s_{in}, s_b) is removed from $\mathcal{L}$ and is developed.

Step 4.1 :

state	word	path region	min in $\mathcal{A}_1$	min in $\mathcal{A}_2$
(s_{in}, s_b)	$+1$	$R_{\mathrm{VS}((s_{in},s_b)),+1}$	s_{in}	s_{out}

The implicit state (s_{in}, s_{out}) already exists.
δ : $\delta^*((s_{in}, s_b), +1) := (s_{in}, s_{out})$.

Step 4.2 :

state	word	path region	min in $\mathcal{A}_1$	min in $\mathcal{A}_2$
(s_{in}, s_b)	-1	$R_{\mathrm{VS}((s_{in},s_b)),-1}$	s_{in}	s_{in}

New implicit state : (s_{in}, s_{in})
VS : $\{(1,0,0),(0,1,0),(0,0,1)\}$
col : `gray`
δ : $\delta^*((s_{in}, s_b), -1) := (s_{in}, s_{in})$

Step 5 : State (s_a, s_{in}) is removed from $\mathcal{L}$ and is developed.

Step 5.1 :

state	word	path region	min in $\mathcal{A}_1$	min in $\mathcal{A}_2$
(s_a, s_{in})	$+1$	$R_{\mathrm{VS}((s_a,s_{in})),+1}$	s_{out}	s_{in}

The implicit state (s_{out}, s_{in}) already exists.
δ : $\delta^*((s_a, s_{in}), +1) := (s_{out}, s_{in})$.

Step 5.2 :

state	word	path region	min in $\mathcal{A}_1$	min in $\mathcal{A}_2$
(s_a, s_{in})	-1	$R_{\mathrm{VS}((s_a,s_{in})),-1}$	s_{in}	s_{in}

The implicit state (s_{in}, s_{in}) already exists.
δ : $\delta^*((s_a, s_{in}), +1) := (s_{in}, s_{in})$.

The resulting IRVA $\mathcal{P}$ is illustrated Figure 7.8. The subsequent minimized version of $\mathcal{P}$ is presented in Figure 7.9. Finally Figure 7.10 shows geometrically the result of the combination of π_1 and π_2 and the minimized version of $\mathcal{P}$ represents the associated pyramid of $\pi_1 \ltimes_{c_i} \pi_2$. In other words, we can see Figure 7.10 as a section of $\Pi_1 \ltimes_{c_i} \Pi_2$ with the plane $Z = 1$.

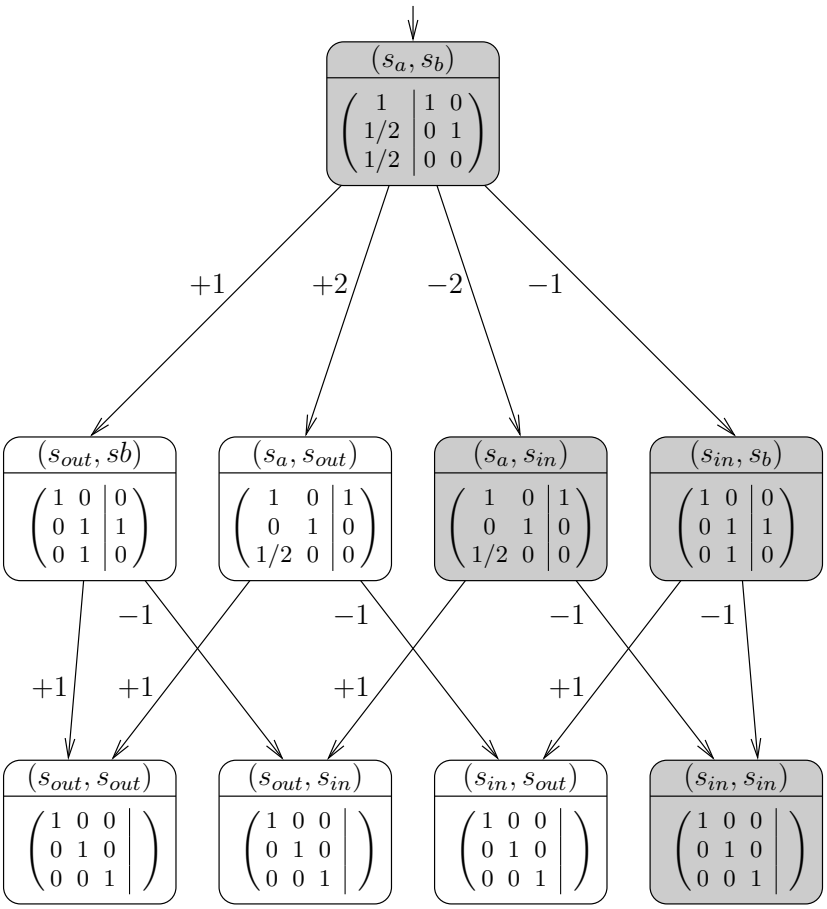


Figure 7.8: Product IRVA $\mathcal{P}$.

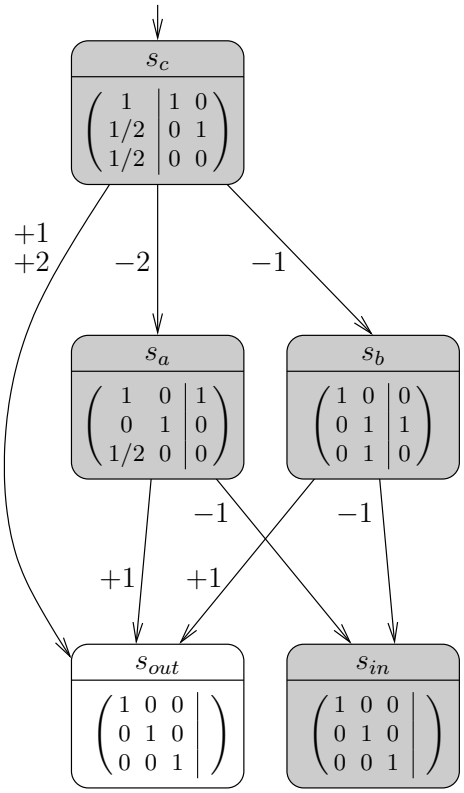


Figure 7.9: Minimized product IRVA $\mathcal{P}$.

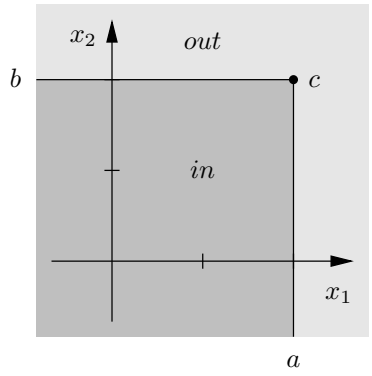


Figure 7.10: The intersection of π_1 and π_2.

Chapter 8

Projection

In this chapter, we develop an algorithm for computing projections of a set represented by an IRVA. Projection is defined as a linear transformation ρ that maps points $\vec{x} \in \mathbb{R}^n$ onto an affine space that is smaller than $\mathbb{R}^n$, and such that $\rho(\rho(\vec{x})) = \rho(\vec{x})$. We first address in Section 8.1 a particular case of projection that simply consists in getting rid of some component in the coordinates of points, and then move to the general case in Section 8.2.

8.1 Aligned Projection

In the following definitions, we use

$$\mathcal{B} = \{\vec{e}_1, \vec{e}_2, \ldots, \vec{e}_n\}$$

as the unit vectors of the Cartesian coordinate system of $\mathbb{R}^n$, with $\vec{e}_1 = (1, 0, \ldots, 0)$, $\vec{e}_2 = (0, 1, \ldots, 0)$, ..., $\vec{e}_n = (0, \ldots, 0, 1)$.

Recall the definition of the aligned projection of a vector :

Definition 8.1 *Let $\vec{p} = (p_1, p_2, \ldots, p_n)$ be a point of $\mathbb{R}^n$. The **aligned projection** of $\vec{p}$ onto the coordinate component different from i, or, in short, w.r.t. i, noted $\vec{p}_{|\neq i}$ is the point*

$$\vec{p}_{|\neq i} = (p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n)$$

□

Definition 8.2 *Let $\pi \subseteq \mathbb{R}^n$ be a polyhedron. The aligned projection of π w.r.t. $i \in \{1, \ldots, n\}$, noted $\pi_{|\neq i}$, is the polyhedron $\pi_{|\neq i} \subseteq \mathbb{R}^{n-1}$ defined as*

$$\pi_{|\neq i} = \{\vec{u} \mid (\exists \vec{x} \in \pi)(\vec{u} = \vec{x}_{|\neq i})\}$$

□

Definition 8.3 *Let $\vec{v}_{|\neq i}$ be a projection w.r.t. i of a point $\vec{v}$. The **kernel** of the projection is the set of points that are projected onto $\vec{0}$, i.e. the set*

$$\{\vec{x} \in \mathbb{R}^n \mid \vec{x}_{|\neq i} = \vec{0}\}.$$

*This corresponds to the vector space generated by $\vec{e}_i$. The **range** of the projection is the smallest vector space that contains the image of $\mathbb{R}^n$, i.e. the set*

$$\{\vec{x} \in \mathbb{R}^n \mid (\exists \vec{y} \in \mathbb{R}^n)(\vec{y}_{|\neq i} = \vec{x})\}.$$

□

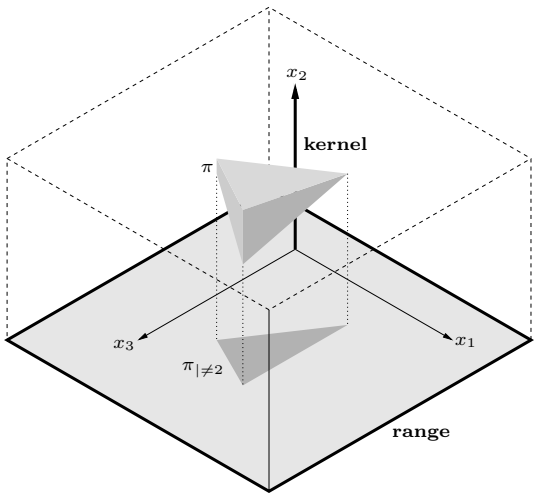


Figure 8.1: Example of a projection of a polyhedron π w.r.t. i.

Example 8.4 *Figure 8.1 shows a polyhedron π and its projection w.r.t. i as well as the range and kernel of that projection.* ◇

8.1.1 Coloring Function

Consider a polyhedron $\pi \subseteq \mathbb{R}^n$ and its projected image $\pi_{|\neq i}$ w.r.t. i.

Let $\vec{x}$ be a point belonging to a component C of $\pi_{|\neq i}$. The point $\vec{x}$ is the projection of an infinite number of points of $\mathbb{R}^n$, i.e. the set

$$X = \{\vec{v} \in \mathbb{R}^n \mid (\exists \lambda \in \mathbb{R})(\vec{v} = \vec{x} + \lambda \vec{x}_i)\}.$$

Without loss of generality, let S be the set of components of Π that contain at least a point of X. We deduce that C is a combination of the elements of S. Hence, the polarity of C is a function of the polarities of the elements of S.

So, in order to assign colors to the components of a projected pyramidal partition, one uses a *coloring function* that maps a set of colors to a single one. The choice of coloring function depends on the intended application.

8.1.2 Projecting a Vector Space

Let $V \subseteq \mathbb{R}^n$ be a vector space generated by the basis $\{\vec{v}_1,\vec{v}_2,\ldots,\vec{v}_m\}$. The projection of V w.r.t. i, noted $V_{|\neq i}$ is the vector space generated by : $\{\vec{v}_{1|\neq i}$, $\vec{v}_{2|\neq i}$, ..., $\vec{v}_{m|\neq i}\}$. Remark that this set generally contains linearly dependent vectors.

Lemma 8.5 *Let $V \subseteq \mathbb{R}^n$ be a vector space such that* $\dim(V) = m$. *The vector space $V_{|\neq i} \subseteq \mathbb{R}^{n-1}$ is such that :*

$$\dim(V_{|\neq i}) = \begin{cases} m-1 & \text{if } \vec{e}_i \in V \\ m & \text{otherwise.} \end{cases}$$

8.1.3 Inverse Projection

An operation needed for developing an algorithm for projecting IRVA is the reverse operator of projection, defined as follows.

Definition 8.6 *For a set $S \subseteq \mathbb{R}^{n-1}$ and $i \in \{1, \ldots, n\}$, the **inverse projection** of S w.r.t. i, noted $S{\uparrow_i}$, is the set*

$$\{\vec{x} \in \mathbb{R}^n \mid \vec{x}_{|\neq i} \in S\}.$$

□

In the next section, we present a projection algorithm that computes the projection of a pyramidal partition $\Pi \subseteq \mathbb{R}^n$, represented by an IRVA $\mathcal{A}$, into a pyramidal partition $\Pi_{|\neq i} \subseteq \mathbb{R}^{n-1}$.

The range of the projection is the hyperplane p_i having for equation $x_i = 0$.

In this algorithm we apply the inverse projection operation to a path region $R_{V,w} \subset \mathbb{R}^{n-1}$, defined by a vector space V of $\mathbb{R}^{n-1}$ and a word $w \in \pm\mathbb{N}\{0,1\}^*$ that encodes a prefix of a direction in the vicinity of V.

Since this region is in $\mathbb{R}^{n-1}$, we adapt it in order for it to be included inside p_i. It is adapted as follows :

$$R = \{\vec{p} \in \mathbb{R}^n \mid (\exists \vec{u} \in R_{V,w})(\vec{p} = (u_1, u_2, \ldots, u_{i-1}, 0, u_i, \ldots, u_m))\}.$$

Since the set $R{\uparrow_i}$ is equivalent to $R_{V{\uparrow_i},w}$, we use $R_{V{\uparrow_i},w}$ immediately.

8.1.4 Projection Algorithm

Let $\mathcal{A} = (n, S_I, S_E, s_0, \delta, \mathrm{VS}, \mathrm{col})$ be an IRVA representing a pyramidal partition $\Pi \subseteq \mathbb{R}^n$ and $i \in \{1 \ldots, n\}$ be a vector component index. In this section we will develop a projection algorithm that computes an IRVA $\mathcal{A}_{|\neq i} = (n-1$, ${S_I}'$, ${S_E}'$, ${s_0}'$, δ', VS', $\mathrm{col}')$ that represents $\Pi_{|\neq i}$.

We have seen in Section 8.1.1 that a polyhedral component of $\Pi_{|\neq i}$ corresponds to a set of components of Π. As a consequence, each implicit state of $\mathcal{A}_{|\neq i}$ will be a composite set of implicit states of $\mathcal{A}$.

The vector space associated to an implicit state corresponds to the affine closure of the polyhedral component that correspond to this state. The projection algorithm is based on the principle that the projection of the affine closure of a polyhedral component corresponds to the affine closure of the projection of this component. Hence, every vector space associated to an implicit state is projected. Some of these projected vector spaces can overlap, as explained in Section 8.1.1. Hence, each polyhedral component of $A_{|\neq i}$ corresponds to a set of polyhedral component of $\mathcal{A}$.

Projection can be carried out somehow similarly to the computation of the product of two IRVA. The projected IRVA is built incrementally from its origin.

The first phase of the algorithm consists in creating the initial state of $\mathcal{A}_{|\neq i}$.

We know that, among the vector spaces associated to the implicit states of $\mathcal{A}$, $\mathrm{VS}(s_0)$ is the smallest. By Lemma 8.5, we know that the dimension of the projection $V_{|\neq i}$ w.r.t. i of a vector space V is either $\dim(V)$ or $\dim(V) - 1$. It follows that $\dim(\mathrm{VS}(s_0)_{|\neq i})$ is the smallest dimension among all the projections of the vector spaces associated to the implicit states of $\mathcal{A}$ and that the initial implicit state of $\mathcal{A}_{|\neq i}$ has $\mathrm{VS}(s_0)_{|\neq i}$ as associated vector space.

So, in order to build the initial state of $\mathcal{A}_{|\neq i}$, we must identify the implicit states with a vector space whose projections share the same dimension as $\mathrm{VS}(s_0)_{|\neq i}$. Those states can only belong to the implicit states that are reachable directly from s_0, i.e. that belong to $\delta_I(s_0)$. Indeed, since every implicit state reachable from another one has an associated vector space of greater dimension, when an implicit state s' is reachable from another implicit state s, which is itself reachable from s_0, we have $\dim(\mathrm{VS}(s_0)) < \dim(\mathrm{VS}(s)) < \dim(\mathrm{VS}(s'))$. This set of implicit states is the initial state of $\mathcal{A}_{|\neq i}$.

The vector space of this initial state is $\mathrm{VS}(s_0)_{|\neq i}$.

During the second phase of the algorithm, a stack is used to keep track of the states of $\mathcal{A}_{|\neq i}$ whose successors need to be explored. Every time a state is created, either explicit or implicit, it is pushed on the stack, along with additional information needed to perform the computation further.

The second phase consists in developing the successors of all states that are present in the stack. Once the stack is empty, the algorithm terminates. The second phase begins after the initial state is created, and pushed on the stack.

When developing a state s, we develop all of its successors in its outgoing decision structure. In order to do that, we must know the last initial state leading to the current state and the labels of the corresponding path. If current state is implicit, then the last implicit state is the current state itself and the path is empty.

For each symbol, we identify the path region R that corresponds to reading

the labels of the path from the last implicit state. Recall that to each implicit state of the projected IRVA corresponds the set I of implicit states of $\mathcal{A}$. We then compute the set of minimal covered elements of $\mathcal{A}$ from s_i by the path region R for every s_i in I. The sets that are obtained are then combined into a single one containing only states that are not reachable from one another. If the intersection of the vector spaces of those implicit states has a non empty intersection with R, then the destination of the symbol is an implicit state. If this implicit state does not exist, then it is created with an associated vector space equal to this intersection. If the intersection is empty, then the destination is an explicit state that has to be created. Every time a state is created, it is placed on the stack in order to expand its successors further.

The color of each implicit state $\{s_1, \ldots, s_k\}$ of $\mathcal{A}_{|\neq i}$ is the color mapped to the following set

$$\{\mathrm{col}(s_1), \ldots, \mathrm{col}(s_k)\}$$

by a given coloring function.

A formal version is given in Algorithm 3.

8.1.5 Example of Projection

In this section, we illustrate a complete run of the IRVA projection algorithm.

The goal is to compute the projection w.r.t. 2 of a polyhedron $\pi \subset \mathbb{R}^3$, defined as the following intersection of constraints :

$$\pi : \Big(x_1 + x_2 - 3x_3 \leq 0\Big) \wedge \Big(-3x_1 + x_2 + x_3 \leq 0\Big) \wedge \Big(x_1 - 3x_2 + x_3 \leq 0\Big)$$

The polyhedron π defines nine polyhedral components : the origin $\vec{0}$ of $\mathbb{R}^3$; the three half-lines A, B and C forming the intersections of the three half-spaces; the three faces AB, BC and AC of π and the 3-spaces *in* and *out*, each containing respectively the points that belong and do not belong to π. The components $\vec{0}, A, B, C, AB, BC, AC$ and *in* are colored `gray` and *out* is colored `white`.

The polyhedron is depicted in Figure 8.2.

To compute the projection of π, we will use the following coloring function :

$$f(\{s_1, \ldots, s_k\}) = \begin{cases} \texttt{gray} & \text{if } (\exists s_i \in \{s_1, \ldots, s_k\})(\mathrm{col}(s_i) = \texttt{gray}) \\ \texttt{white} & \text{otherwise} \end{cases}$$

The IRVA $\mathcal{A}$ representing π it is defined by the following tuple :

$$\mathcal{A} = (3, S_I, S_E, s_0, \delta, \mathrm{VS}, \mathrm{col})$$

Its complete definition is straightforward and is given in Figure 8.3. The states s_{in} and s_{out} represent respectively the polyhedral components *in* and *out*. The states s_0, s_a, s_b, s_c, s_{ab}, s_{bc} and s_{ac} represent respectively the components $\vec{0}$, A, B, C, AB, BC and AC.

input : An IRVA $\mathcal{A} = (n, S_I, S_E, s_0, \delta, \mathrm{VS}, \mathrm{col})$, an axis of projection $\vec{x}_i$ and a coloring function f.
output: An IRVA $\mathcal{A}_{|\neq i} = (n-1, {S_I}', {S_E}', {s_0}', \delta', \mathrm{VS}', \mathrm{col}')$.

foreach $q \in S_I$ **do**
  $\mathrm{PVS}(q) := \mathrm{VS}(q)_{|\neq i}$
end
${s_0}' := \{s_0\} \cup \{q \in \delta_I(s_0) \mid \dim(\mathrm{PVS}(q)) = \dim(\mathrm{PVS}(s_0))\}$;
${S_I}' := \{{s_0}'\}$;
$\mathrm{VS}'({s_0}') := \mathrm{PVS}(s_0)$;
$\mathrm{col}'({s_0}') := f({s_0}')$;
$stack := \emptyset$;
`push` $(stack, (s'_0, s'_0, \varepsilon))$;
while `not(empty?(`$stack$`))` **do**
  (S, s, w) := `pop(`$stack$`)`;
  foreach $\sigma \in$ `succ(`s`)` **do**
    $w' := w\sigma$;
    $R := R_{\mathrm{VS}'(S)\uparrow_i, w'}$;
    $F := \emptyset$;
    foreach $s_i \in S$ **do**
      $F := \mathrm{merge}(F, \mathbf{minimals}(\mathcal{A}, s_i, R))$;
    end
    $I := \bigcap_{s_i \in F} \mathrm{PVS}(s_i)$;
    if $I \cap R_{\mathrm{VS}'(S), w'} = \emptyset$ **then**
      ${s_N}$:= `new()`;
      ${S_E}'$:= ${S_E}' \cup \{s_N\}$;
      $\delta'(s, \sigma)$:= s_N;
      `push` $(stack, (S, s_N, w'))$;
    else
      if $F \notin {S_I}'$ **then**
        ${S_I}' := {S_I}' \cup \{F\}$;
        $\mathrm{VS}'(F) = I$;
        $\mathrm{col}'(F) := f(F)$;
        `push` $(stack, (F, F, \varepsilon))$;
      end
      $\delta'(s, \sigma) := F$;
    end
  end
end

Algorithm 3: Computation of $\mathcal{A}_{|\neq i}$

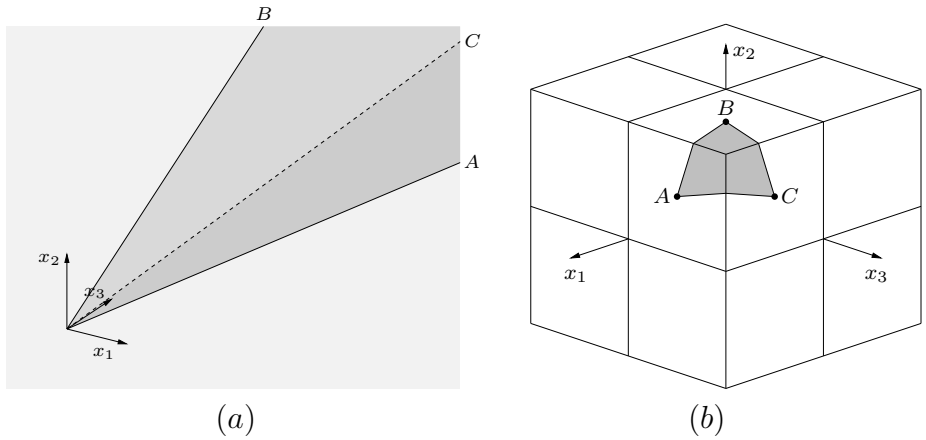


Figure 8.2: (a) Example polyhedron π, (b) intersection of π with normalization cube.

The IRVA $\mathcal{A}_{|\neq 2} = (2, {S_I}', {S_E}', \{s_0\}, \delta', \mathrm{VS}', \mathrm{col}')$ is built by the projection algorithm as follows.

First, the associated vector space of each implicit state is projected w.r.t. 2 and stored in an associative table *PVS*.

$\mathrm{PVS}(s_0)$	$\emptyset$
$\mathrm{PVS}(s_a)$	$\{(1, 1/2)\}$
$\mathrm{PVS}(s_b)$	$\{(1, 1)\}$
$\mathrm{PVS}(s_c)$	$\{(1, 2)\}$
$\mathrm{PVS}(s_{ab})$	$\{(1, 0), (0, 1)\}$
$\mathrm{PVS}(s_{bc})$	$\{(1, 0), (0, 1)\}$
$\mathrm{PVS}(s_{ac})$	$\{(1, 0), (0, 1)\}$
$\mathrm{PVS}(s_{in})$	$\{(1, 0), (0, 1)\}$
$\mathrm{PVS}(s_{out})$	$\{(1, 0), (0, 1)\}$

Second, the initial state of $\mathcal{A}_{|\neq 2}$ is created. It corresponds to the set $\{s_0\}$, as no other implicit state has an empty associated vector space after projection. Its color is `gray`.

In order to keep this presentation concise, we call the function **minimals** (see Algorithm 1 for a definition) with a set of implicit states for second argument in the following steps, instead of a single implicit state. The expression

$$\mathbf{minimals}(\mathcal{A}, \{s_1, s_2, s_3\}, R)$$

is the merged result of function **minimals** for each implicit state of the set given as argument :

$$\mathbf{merge}(\mathbf{merge}(\mathbf{minimals}(\mathcal{A}, s_1, R), \mathbf{minimals}(\mathcal{A}, s_2, R)), \mathbf{minimals}(\mathcal{A}, s_3, R))$$

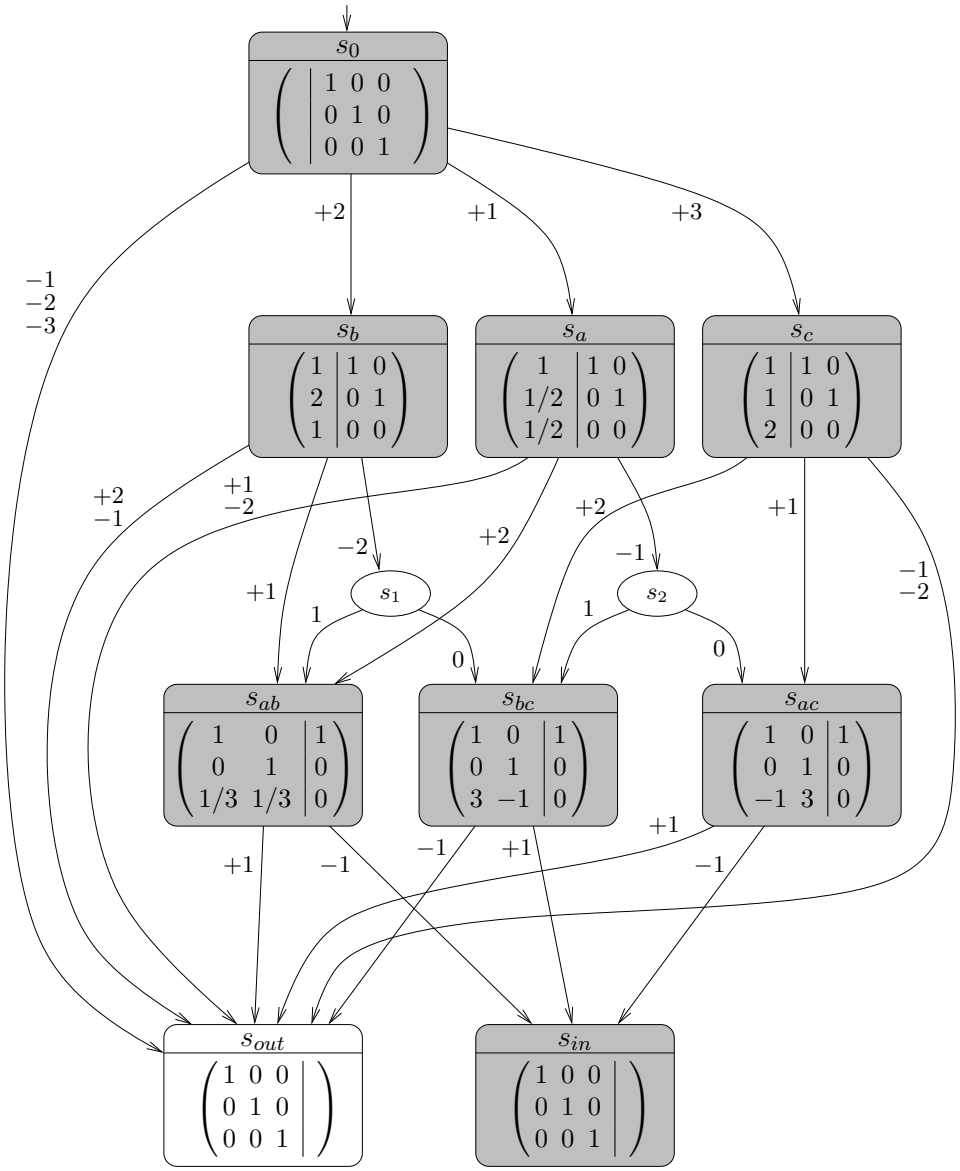


Figure 8.3: Graph representation of $\mathcal{A}$.

Step 1 : Pop $(\{s_0\}, \{s_0\}, \varepsilon)$ from the stack.

Step 1.1 :

state	symbol	path region R	**minimals**$(\mathcal{A}, \{s_0\}, R)$
$\{s_0\}$	$+1$	$R_{\mathrm{VS}'(\{s_0\})\uparrow_2,+1}$	$\{s_a, s_b\}$

There is no intersection with $R_{\mathrm{VS}'(\{s_0\}),+1}$
The explicit state s_1 is created, $\delta'(\{s_0\}, +1) = s_1$.
The tuple $(\{s_0\}, s_1, +1)$ is pushed on stack.

Step 1.2 :

state	symbol	path region R	**minimals**$(\mathcal{A}, \{s_0\}, R)$
$\{s_0\}$	$+2$	$R_{\mathrm{VS}'(\{s_0\})\uparrow_2,+2}$	$\{s_b, s_c\}$

There is no intersection with $R_{\mathrm{VS}'(\{s_0\}),+2}$
The explicit state s_2 is created, $\delta'(\{s_0\}, +2) = s_2$.
The tuple $(\{s_0\}, s_2, +2)$ is pushed on stack.

Step 1.3 :

state	symbol	path region R	**minimals**$(\mathcal{A}, \{s_0\}, R)$
$\{s_0\}$	-1	$R_{\mathrm{VS}'(\{s_0\})\uparrow_2,-1}$	$\{s_{out}\}$

There is an intersection with $R_{\mathrm{VS}'(\{s_0\}),-1}$
The implicit state $\{s_{out}\}$ is created, $\delta'(\{s_0\}, -1) = \{s_{out}\}$.
$\mathrm{VS}'(\{s_{out}\}) = \{(1,0),(0,1)\}$.
$\mathrm{col}'(\{s_{out}\}) = \texttt{white}$.
The tuple $(\{s_{out}\}, \{s_{out}\}, \varepsilon)$ is pushed on stack.

Step 1.4 :

state	symbol	path region R	**minimals**$(\mathcal{A}, \{s_0\}, R)$
$\{s_0\}$	-2	$R_{\mathrm{VS}'(\{s_0\})\uparrow_2,-2}$	$\{s_{out}\}$

There is an intersection with $R_{\mathrm{VS}'(\{s_0\}),-2}$
The implicit state $\{s_{out}\}$ exists, $\delta'(\{s_0\}, -2) = \{s_{out}\}$.

Step 2 : Pop $(\{s_0\}, s_1, +1)$ from the stack.

Step 2.1 :

state	symbol	path region R	**minimals**$(\mathcal{A}, \{s_0\}, R)$
s_1	0	$R_{\mathrm{VS}'(\{s_0\})\uparrow_2,+1\ 0}$	$\{s_{out}\}$

There is an intersection with $R_{\mathrm{VS}'(\{s_0\}),+1\ 0}$
The implicit state $\{s_{out}\}$ exists, $\delta'(s_1, 0) = \{s_{out}\}$.

Step 2.2 :

state	symbol	path region R	**minimals**$(\mathcal{A}, \{s_0\}, R)$
s_1	1	$R_{\mathrm{VS}'(\{s_0\})\uparrow_2,+1\ 1}$	$\{s_a, s_b\}$

There is no intersection with $R_{\mathrm{VS}'(\{s_0\}),+1\ 1}$
The explicit state s_3 is created, $\delta'(s_1, 1) = s_3$.
The tuple $(\{s_0\}, s_3, +1\ 1)$ is pushed on stack.

Step 3 : Pop $(\{s_0\}, s_2, +2)$ from the stack.

Step 3.1 :

state	symbol	path region R	**minimals**$(\mathcal{A}, \{s_0\}, R)$
s_2	0	$R_{\mathrm{VS}'(\{s_0\})\uparrow_2,+2\ 0}$	$\{s_{out}\}$

There is an intersection with $R_{\mathrm{VS}'(\{s_0\}),+2\ 0}$
The implicit state $\{s_{out}\}$ exists, $\delta'(s_2, 0) = \{s_{out}\}$.

Step 3.2 :

state	symbol	path region R	**minimals**$(\mathcal{A}, \{s_0\}, R)$
s_2	1	$R_{\mathrm{VS}'(\{s_0\})\uparrow_2,+2\ 1}$	$\{s_b, s_c\}$

There is no intersection with $R_{\mathrm{VS}'(\{s_0\}),+2\ 1}$
The explicit state s_4 is created, $\delta'(s_2, 1) = s_4$.
The tuple $(\{s_0\}, s_4, +2\ 1)$ is pushed on stack.

Step 4 : Pop ($\{s_0\}$, s_3, +1 1) from the stack.

Step 4.1 :

state	symbol	path region R	**minimals**$(\mathcal{A}, \{s_0\}, R)$
s_3	0	$R_{\mathrm{VS}'(\{s_0\})\uparrow_2,+1\ 10}$	$\{s_a\}$

There is an intersection with $R_{\mathrm{VS}'(\{s_0\}),+1\ 10}$
The implicit state $\{s_a\}$ is created, $\delta'(s_3, 0) = \{s_a\}$.
$\mathrm{VS}'(\{s_a\}) = \{(1, 1/2)\}$.
$\mathrm{col}'(\{s_a\}) = \texttt{gray}$.
The tuple $(\{s_a\}, \{s_a\}, \varepsilon)$ is pushed on stack.

Step 4.2 :

state	symbol	path region R	**minimals**$(\mathcal{A}, \{s_0\}, R)$
s_3	1	$R_{\mathrm{VS}'(\{s_0\})\uparrow_2,+1\ 11}$	$\{s_a, s_b\}$

There is no intersection with $R_{\mathrm{VS}'(\{s_0\}),+1\ 11}$
The explicit state s_5 is created, $\delta'(s_3, 1) = s_5$.
The tuple $(\{s_0\}, s_5, +1\ 11)$ is pushed on stack.

Step 5 : Pop ($\{s_0\}$, s_4, +2 1) from the stack.

Step 5.1 :

state	symbol	path region R	**minimals**$(\mathcal{A}, \{s_0\}, R)$
s_4	0	$R_{\mathrm{VS}'(\{s_0\})\uparrow_2,+2\ 10}$	$\{s_c\}$

There is an intersection with $R_{\mathrm{VS}'(\{s_0\}),+2\ 10}$
The implicit state $\{s_c\}$ is created, $\delta'(s_4, 0) = \{s_c\}$.
$\mathrm{VS}'(\{s_c\}) = \{(1, 2)\}$.
$\mathrm{col}'(\{s_c\}) = \texttt{gray}$.
The tuple $(\{s_c\}, \{s_c\}, \varepsilon)$ is pushed on stack.

Step 5.2 :

state	symbol	path region R	**minimals**$(\mathcal{A}, \{s_0\}, R)$
s_4	1	$R_{\mathrm{VS}'(\{s_0\})\uparrow_2,+2\ 11}$	$\{s_b, s_c\}$

There is no intersection with $R_{\mathrm{VS}'(\{s_0\}),+2\ 11}$
The explicit state s_6 is created, $\delta'(s_4, 1) = s_6$.
The tuple $(\{s_0\}, s_6, +2\ 11)$ is pushed on stack.

Step 6 : Pop $(\{s_a\}, \{s_a\}, \varepsilon)$ from the stack.

Step 6.1 :

state	symbol	path region R	**minimals**$(\mathcal{A}, \{s_a\}, R)$
$\{s_a\}$	$+1$	$R_{\mathrm{VS}'(\{s_a\})\uparrow_2,+1}$	$\{s_{out}\}$

There is an intersection with $R_{\mathrm{VS}'(\{s_a\}),+1}$
The implicit state $\{s_{out}\}$ exists, $\delta'(\{s_a\}, +1) = \{s_{out}\}$.

Step 6.2 :

state	symbol	path region R	**minimals**$(\mathcal{A}, \{s_a\}, R)$
$\{s_a\}$	-1	$R_{\mathrm{VS}'(\{s_a\})\uparrow_2,-1}$	$\{s_{ab}, s_{ac}\}$

There is an intersection with $R_{\mathrm{VS}'(\{s_a\}),-1}$
The implicit state $\{s_{ab}, s_{ac}\}$ is created, $\delta'(\{s_a\}, -1) = \{s_{ab}, s_{ac}\}$.
$\mathrm{VS}'(\{s_{ab}, s_{ac}\}) = \{(1, 0), (0, 1)\}$.
$\mathrm{col}'(\{s_{ab}, s_{ac}\}) = \texttt{gray}$.
The tuple $(\{s_{ab}, s_{ac}\}, \{s_{ab}, s_{ac}\}, \varepsilon)$ is pushed on stack.

Step 7 : Pop $(\{s_0\}, s_5, +1\ 11)$ from the stack.

Step 7.1 :

state	symbol	path region R	**minimals**$(\mathcal{A}, \{s_0\}, R)$
s_5	0	$R_{\mathrm{VS}'(\{s_0\})\uparrow_2,+1\ 110}$	$\{s_a\}$

There is an intersection with $R_{\mathrm{VS}'(\{s_0\}),+1\ 110}$
The implicit state $\{s_a\}$ exists, $\delta'(s_5, 0) = \{s_a\}$.

Step 7.2 :

state	symbol	path region R	**minimals**$(\mathcal{A}, \{s_0\}, R)$
s_5	1	$R_{\mathrm{VS}'(\{s_0\})\uparrow_2,+1\ 111}$	$\{s_b, s_{ac}\}$

There is an intersection with $R_{\mathrm{VS}'(\{s_0\}),+1\ 111}$
The implicit state $\{s_b, s_{ac}\}$ is created, $\delta'(s_5, 1) = \{s_b, s_{ac}\}$.
$\mathrm{VS}'(\{s_b, s_{ac}\}) = \{(1, 1)\}$.
$\mathrm{col}'(\{s_b, s_{ac}\}) = \texttt{gray}$.
The tuple $(\{s_b, s_{ac}\}, \{s_b, s_{ac}\}, \varepsilon)$ is pushed on stack.

Step 8 : Pop ($\{s_c\}$, $\{s_c\}$, ε) from the stack.

Step 8.1 :

state	symbol	path region R	**minimals**$(\mathcal{A}, \{s_c\}, R)$
$\{s_c\}$	$+1$	$R_{\mathrm{VS}'(\{s_c\})\uparrow_2,+1}$	$\{s_{bc}, s_{ac}\}$

There is an intersection with $R_{\mathrm{VS}'(\{s_c\}),+1}$
The implicit state $\{s_{bc}, s_{ac}\}$ is created, $\delta'(\{s_c\}, +1) = \{s_{bc}, s_{ac}\}$.
$\mathrm{VS}'(\{s_{bc}, s_{ac}\}) = \{(1,0),(0,1)\}$.
$\mathrm{col}'(\{s_{bc}, s_{ac}\}) = \texttt{gray}$.
The tuple $(\{s_{bc}, s_{ac}\}, \{s_{bc}, s_{ac}\}, \varepsilon)$ is pushed on stack.

Step 8.2 :

state	symbol	path region R	**minimals**$(\mathcal{A}, \{s_c\}, R)$
$\{s_c\}$	-1	$R_{\mathrm{VS}'(\{s_c\})\uparrow_2,-1}$	$\{s_{out}\}$

There is an intersection with $R_{\mathrm{VS}'(\{s_c\}),-1}$
The implicit state $\{s_{out}\}$ exists, $\delta'(\{s_c\}, -1) = \{s_{out}\}$.

Step 9 : Pop ($\{s_0\}$, s_6, +2 11) from the stack.

Step 9.1 :

state	symbol	path region R	**minimals**$(\mathcal{A}, \{s_0\}, R)$
s_6	0	$R_{\mathrm{VS}'(\{s_0\})\uparrow_2,+2\ 110}$	$\{s_c\}$

There is an intersection with $R_{\mathrm{VS}'(\{s_0\}),+2\ 110}$
The implicit state $\{s_c\}$ exists, $\delta'(s_6, 0) = \{s_c\}$.

Step 9.2 :

state	symbol	path region R	**minimals**$(\mathcal{A}, \{s_0\}, R)$
s_6	1	$R_{\mathrm{VS}'(\{s_0\})\uparrow_2,+2\ 111}$	$\{s_b, s_{ac}\}$

There is an intersection with $R_{\mathrm{VS}'(\{s_0\}),+2\ 111}$
The implicit state $\{s_b, s_{ac}\}$ exists, $\delta'(s_6, 1) = \{s_b, s_{ac}\}$.

Step 10 : Pop ($\{s_b, s_{ac}\}$, $\{s_b, s_{ac}\}$, ε) from the stack.

Step 10.1 :

state	symbol	path region R	**minimals**$(\mathcal{A}, \{s_b, s_{ac}\}, R)$
$\{s_b, s_{ac}\}$	$+1$	$R_{\mathrm{VS}'(\{s_b, s_{ac}\})\uparrow_2,+1}$	$\{s_{ab}, s_{ac}\}$

There is an intersection with $R_{\mathrm{VS}'(\{s_b, s_{ac}\}),+1}$
The implicit state $\{s_{ab}, s_{ac}\}$ exists, $\delta'(\{s_b, s_{ac}\}, +1) = \{s_{ab}, s_{ac}\}$.

Step 10.2 :

state	symbol	path region R	**minimals**$(\mathcal{A}, \{s_b, s_{ac}\}, R)$
$\{s_b, s_{ac}\}$	-1	$R_{\mathrm{VS}'(\{s_b, s_{ac}\})\uparrow_2,-1}$	$\{s_{bc}, s_{ac}\}$

There is an intersection with $R_{\mathrm{VS}'(\{s_b, s_{ac}\}),-1}$
The implicit state $\{s_{bc}, s_{ac}\}$ exists, $\delta'(\{s_b, s_{ac}\}, -1) = \{s_{bc}, s_{ac}\}$.

The computed IRVA $\mathcal{A}_{|\neq 2}$ is given in Figure 8.4. The minimized IRVA is depicted in Figure 8.5. The states have been relabeled to match the original names of $\mathcal{A}$ in order to emphasize the fact that this minimized IRVA represents the polyhedron $\pi_{|\neq 2}$.

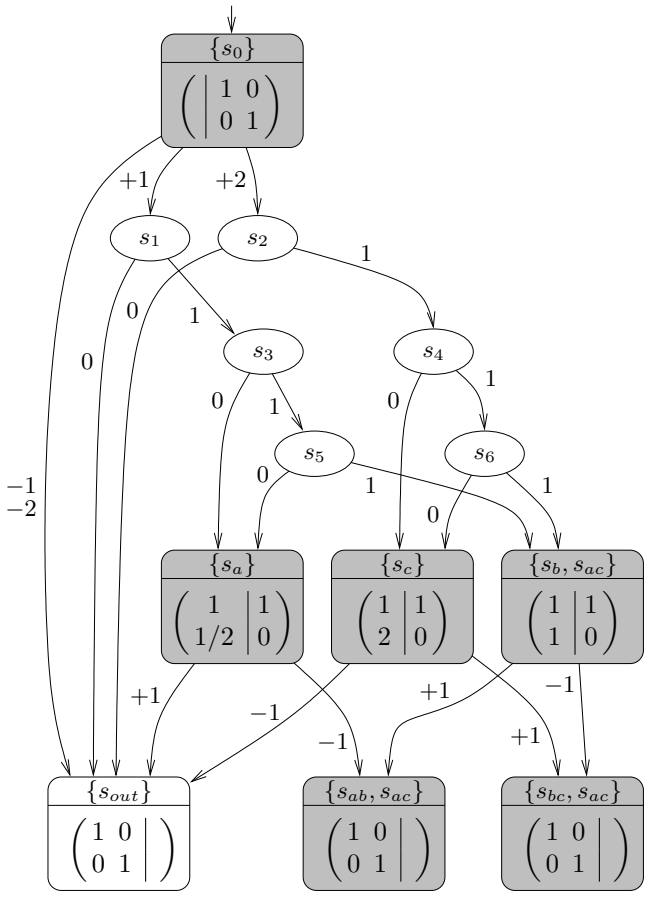


Figure 8.4: The computed IRVA $\mathcal{A}_{|\neq 2}$.

8.2 Generalized Projection

In Section 8.1, we developed an algorithm for computing aligned projections of IRVA. We now show that this operation can be extended to a more general notion of projection.

This generalized projection consists in using arbitrary vector spaces as kernel and range, with the restriction that the vector basis $\{\vec{k}_1, \ldots, \vec{k}_p\}$ generating

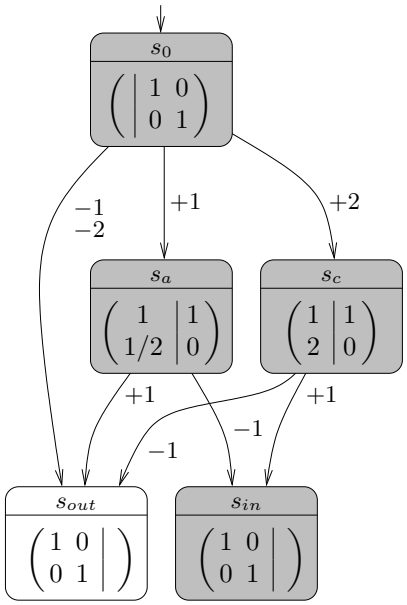


Figure 8.5: Minimized version of $\mathcal{A}_{|\neq 2}$.

the kernel and the vector basis $\{\vec{r}_1, \ldots, \vec{r}_{n-p}\}$ generating the range are mutually independent. In other words, the vectors in $\{\vec{k}_1, \ldots, \vec{k}_p, \vec{r}_1, \ldots, \vec{r}_{n-p}\}$ are linearly independent and form a vector basis generating $\mathbb{R}^n$.

Definition 8.7 *Let $S \subseteq \mathbb{R}^n$ be a set, $K \subseteq \mathbb{R}^n$ be a vector space of basis $\{\vec{k}_1, \ldots, \vec{k}_p\}$ and $R \subseteq \mathbb{R}^n$ a vector space of basis $\{\vec{r}_1, \ldots, \vec{r}_{n-p}\}$ such that $\{\vec{k}_1, \ldots, \vec{k}_p, \vec{r}_1, \ldots, \vec{r}_{n-p}\}$ are linearly independent and form a vector basis generating $\mathbb{R}^n$. The **generalized projection** of S w.r.t. K and R, noted $S{\downarrow_{K,R}}$, is the set :*

$$(S + K) \cap R.$$

□

Notice that this set is by construction included in R and could be expressed in the coordinate system $\{\vec{r}_1, \ldots, \vec{r}_{n-p}\}$, should one need the projection of S expressed in $\mathbb{R}^{n-p}$.

Definition 8.8 *Let $S \subseteq \mathbb{R}^n$ be a set and $K \subseteq \mathbb{R}^n$ be a vector space. The **generalized inverse projection** of S w.r.t. K, noted $S{\uparrow_K}$, is the set :*

$$S + K.$$

□

With minor adjustments, Algorithm 3 can be adapted in order to compute generalized projection of IRVA. We now study in detail the necessary modifications.

Let n be a dimension, K be a vector space used as kernel generated by $\{\vec{k}_1, \ldots \vec{k}_p\}$ and R be a vector space used as range generated by $= \{\vec{r}_1, \ldots, \vec{r}_{n-p}\}$. The projection of a set S w.r.t. K into R maps points in $\mathbb{R}^n$ to points in $\mathbb{R}^{n-p}$.

The first adaptation concerns the projection of vector spaces, done when computing the projection of the vector spaces associated to implicit states. In the case of aligned projection, this operation simply amounts to removing a row of the vector basis generating the vector space to project. This row corresponds to the coordinate component that is projected away. A precaution is to verify that the result is composed of linearly independent vectors.

In the general case, in order to project a vector space V generated by $\{\vec{v}_1, \ldots, \vec{v}_m\}$ into R w.r.t. K, one has to express those m vectors into the coordinate system

$$\{\vec{k}_1, \ldots, \vec{k}_p, \vec{r}_1, \ldots, \vec{r}_{n-p}\}$$

and remove the m first vector components.

The second adaptation concerns the inverse projection operation of a path region R presented in Section 8.1.3. In the case of aligned projection, we have assumed that the operation consists on the inverse projection of the points of R w.r.t. to axis i, in order to characterize the set of points that have an image by projection inside this path region. In the general case, as the projection have a larger kernel, the inverse operation is applied w.r.t. each vector of K. So, instead of having

$$R = R_{\mathrm{VS}'(S)\uparrow_i, w'}$$

we have

$$R = R_{\mathrm{VS}'(S)\uparrow_K, w'}$$

with K being the vector basis of the kernel.

The other parts of the algorithm are unchanged.

8.2.1 Generalized Coloring Functions

An interesting possibility offered by our approach is to define coloring functions that implement features such as, for instance, simulating transparency when several projected elements are stacked together. Furthermore, one can replace coloring functions by a more general mechanism for assigning colors, taking into account additional information such as the distance between stacked components, or the order of this stacking. For example, opacity could be simulated by defining the color of a stacking as being equal to that of its uppermost element. This generalized notion of coloring function paves the way to a visualization method for IRVA, by first projecting the structure onto the visualization window using a color assignment scheme that enhances the perception of the geometry, and then solving the point classification problem for each displayed pixel, which can be done efficiently as discussed in Section 5.1.4.

Chapter 9

Implementation and Experimental Results

An implementation has been developed in the C language in order to be able to evaluate the data structure and the algorithms presented in this work. The main purpose of this implementation is to test the presented concepts and algorithms with larger data structures than human-sized examples. Indeed, although the properties of the algorithms have been studied with the help of some handmade case studies, being able to explore larger problems offers valuable insight on the behaviors of IRVA.

We consider this implementation as a first step toward a usable IRVA package. As a consequence, it has been developed as a proof of concept in which correctness and readability take priority over performance. This implementation made it possible to evaluate IRVA on several case studies, and to identify mechanisms that could benefit further optimization.

In this chapter, we will first present the scope of our implementation. We then discuss design choices, explain in detail the implementation issues behind the essential mechanisms. Then we introduce several case studies and discuss the experimental results, that have been obtained.

9.1 Features

We now describe the features that have been implemented in our IRVA package.

Creation of primitive elements is, of course, a first mandatory step for the incremental construction of any set. Here, we list the different elementary IRVA that can directly be built.

Vector spaces : it is possible to build an IRVA representing a given vector space, specified by a vector basis provided in matrix notation along with its color, as well as a color for its surrounding space.

Linear constraints : Representing linear constraints plays a crucial role when building volumes. Ideally, one should provide just $\vec{a}$, such that the represented set is $\vec{a} \cdot \vec{x} \# 0$, with $\# \in \{\leq, <, >, \geq\}$ along with three colors, one for the points satisfying $\vec{a} \cdot \vec{x} = 0$, one for the points inside $\vec{a} \cdot \vec{x} < 0$ and the last one for the points not satisfying $\vec{a} \cdot \vec{x} \leq 0$.

Manual definition : This operation consists in building an IRVA from a description of its states and transitions. In order to reduce the pain of creating manually an IRVA, a completion function permits not to specify elements of one particular color and let it fill the spatial gaps for the represented pyramidal partition to be complete. For example, when using only two colors `in` and `out` and representing a triangle, only implicit states defining the triangle can be specified along with the decisions between them. The completion function can create the `out` universal implicit state and more importantly all transitions leading to it.

In addition, we have also implemented operations for manipulating represented IRVA. Those are the following :

Boolean combination : Given two IRVA $\mathcal{A}$ and B and a coloring function c, this operation computes the IRVA representing $\mathcal{A} \ltimes_c \mathcal{B}$.

Minimization : This operation computes the canonical form of a given IRVA.

Projection : we have only implemented aligned projections of IRVA, as defined in Section 8.1. Given an IRVA $\mathcal{A}$ and $i \in \{1, 2, \ldots, n\}$, this operation computes $\mathcal{A}_{|\neq i}$.

Finally a script system has been added to facilitate the definition of test scenarios. Such scenarios are precious to create complex IRVA incrementally, or store or load them from a files. Also, an export module makes it possible to create a graph representation of an IRVA in *dot* format [GN00]. Finally a simple visualization tool is proposed, which is a very useful tool for debugging.

9.2 Data Structure Representation

In this section we detail the choices made for the operational representation of the IRVA data structure.

We followed the principles of [Hof89], by isolating the program from a library responsible of all calculations on real numbers. This clean separation make it possible to implement algorithms without having to worry about the fact that floating-point arithmetic is approximate and can induce errors [HHK89, McC98]. Our implementation relies on a symbolic representation of rational numbers with numerator and denominator of arbitrary sizes. Hence, all calculations are done in exact arithmetic, in line with our choice of privileging correctness over performance.

Using exact arithmetic introduces a non negligible overhead in computing power but leads to nice properties. First, it enables the program not to care about approximations and errors. This is a clear design advantage over the traditional approach of using floating-point arithmetic with a lot of effort to be invested in carefully detecting, trying to avoid and more improbably trying to correct errors induced by inaccuracies.

We represent vector spaces in matrix notation. More precisely, a vector space V generated by the basis $\{\vec{v}_1, \ldots, \vec{v}_m\}$ of $\mathbb{R}^n$ is represented by a matrix of n rows and m columns. The i^{th} column corresponds to the vector $\vec{v}_i$ of the basis. Such matrices can be represented in a canonical way by carrying out Gauss-Jordan elimination to produce a reduced row-echelon form [Mey00]. In the case of vector spaces associated to implicit states, it is useful to also precompute and to store another matrix, corresponding to the inverse of the canonical basis of the complementary vector space of V. This additional matrix is useful for speeding up the computation of encodings.

Colors associated to vectors spaces are represented as 32-bit integer numbers. Two basic colors are already defined, they correspond to *inside* and *outside*. They make it possible to represent pyramidal polyhedra out of the box. Representing more colors is however not a problem, since the implementation delegates the management of colors to the user via externally definable coloring functions that take integers as parameters. For the computation of Boolean combinations and projections, some classical color functions are already available by default.

The transition relation of IRVA is represented by a vector of records. One record consists of a type that differentiates implicit from explicit states, and the definition of the outgoing transitions.

In order for the transition relation to be binary, face decision symbols are encoded in base 2 and new explicit states are added, along with new transitions that recognize those encodings. Notice that path regions are defined as soon as the encoding of the face symbol is entirely read, and not sooner.

9.3 Key Mechanisms

We begin the description of the implementation of operations by defining a set of key mechanisms on which the algorithms rely on. For each of those mechanisms, we discuss the design choices that have been made.

9.3.1 Arbitrary Precision Arithmetic

We chose to use the *GNU Multiple Precision Arithmetic Library (gmp)* [Gra12] to manipulate integers of arbitrary precision, from which we obtain procedures for manipulating rational numbers, vectors and matrices.

9.3.2 Manipulating Path Regions

In most algorithms, we use path regions to represent sets of points characterized by the same prefix from an implicit state. Those path regions are, by design, non empty, convex and can have open boundaries. We also manipulate vector spaces, which are also convex sets. A tool for manipulating efficiently convex polyhedra with those properties is then of major importance. The two main operations applied to convex polyhedra are to test whether two regions are disjoint, and computing intersections. The former operation is employed when a vector space has an intersection with some path region, and the latter is used when identifying the vector space that has to be associated to a new implicit state. We use the *The Parma Polyhedra Library (PPL)* [BHZ08] for performing these operations.

9.3.3 Adjacency Information

In order to be able to quickly determine whether two implicit states represent incident polyhedral components, one has to check whether one state is reachable or not from the other inside the IRVA. This operation is very often required by different parts of algorithms. A data structure based on an adjacency matrix has been developed in order to speed up this search. Since we are only interested by reachability between implicit states, a square table of size $|S_I|^2$ is built. Each implicit state is characterized with an integer in interval $[1, |S_I|]$. A cell with coordinates (i, j) in the table has a Boolean value that is true iff the state associated to number j is reachable from the one associated to i. This adjacency matrix has two modes : an offline one that is precomputed for a static IRVA, typically an IRVA used as a source for a manipulation, without being altered; and a dynamic mode in which it is possible to add states as well as incidence information to an already existing adjacency matrix. This option is needed when reachability information is needed for an IRVA that is being constructed, for example during minimization.

9.3.4 Minimal Covered Elements

The implementation of the *minimals* function, that computes the set of minimal covered elements by a non empty convex region R, and from an implicit state s (c.f. Section 5.4.1) is based on a breadth-first search. This search is implemented by using a stack in which states that need to be explored are pushed, along with the last implicit state encountered, and the word leading from this last implicit state to the current state.

In Algorithm 1, for each state visited, one tests whether the cover region corresponding to each outgoing transition is disjoint from the given region (the region for which we search the set of minimal elements). This test aims to prune the search tree as early as possible. Those tests are done by first computing the pyramidal polyhedron corresponding to the path region $R_{VS(s),w}$

and calculating exactly its intersection with R. The resulting polyhedron is then tested for emptiness.

The Algorithm presented in Section 5.4.1 creates the set of minimal states in two main phases. Each call to *search_minimals* returns a set of minimal states corresponding to a particular subset of R, and different results are merged into new a set of minimal states (by the use of the *merge* function).

In the implementation, we simplified this procedure by defining a data structure representing a minimal set, that filters incrementally the states during their insertion. The data structure uses such a mechanism in order to be as efficient as possible :

- It is not possible to have multiple occurrences of the same state in the set.
- States are stored in an ordered manner to speed up set comparisons.
- When inserting a state s in a set, an adjacency matrix of the implicit states of $\mathcal{A}$ is used. If the set contains a state s' such that s is reachable from s', the set is left unchanged. Otherwise, the states of the set that are reachable from s in $\mathcal{A}$ are removed from the set, and s is finally added.

In practice, an instance of this data structure is shared between successive calls to the *search_minimals* function inside the computation of the set of minimal states covered by a region. This avoids the need to merge results.

9.4 Path Regions

In the definition of path regions in Section 5.3.1, we did not address the problem of representing and computing them. There are several ways of tackling the problem, with different levels of performance.

For the sake of generality, and because the exact nature of the required manipulations were not fully known by the time the development of this implementation started, we used the most general and flexible data structure possible which is based on manipulation of general convex polyhedra.

Indeed : one can create a polyhedron from an intersection of closed and opened linear constraints, the problem of determining whether two path regions have points in common can be reduced to testing emptiness of their intersection, etc.

We now explain more precisely how regions are constructed in our application.

We study the creation of a path region $R \subset \mathbb{R}^n$ based on a m-dimensional vector space represented by a basis $V = \{\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_m\}$ and a word $w \in \pm\mathbb{N}\{0,1\}^*$.

Let $Z = \{\vec{z}_1, \vec{z}_2, \ldots, \vec{z}_{n-m}\}$ be the canonical basis of the complementary vector space of V. By definition, we know that Z is a selection elements of $E = \{(1,0,\ldots,0),(0,1,\ldots,0), \ \ldots \ ,(0,\ldots,0,1)\}$.

As explained in Sections 5.2.2 and 5.2.3, the encoding of a direction is dependent to Z, since the word w is a prefix of encodings of directions in the outgoing space spanned by Z. It is then mandatory to know the vector basis Z in order to interpret the symbols of w and this interpretation is independent to the orientation of V.

First, we construct the region inside the space spanned by Z. The word w is composed of a face selection symbol followed by symbols of $\{0,1\}$ that subdivide this face. The face itself is a $(n-m-1)$-dimensional component. We decompose w in three parts $\#iw'$ where $\# \in \{+,-\}$ is the sign of the face, $i \in \{1,\ldots,n-m\}$ is a face number and $w' \in \{0,1\}^*$ is a suffix. Let $\vec{z}_i$ be the vector in Z that is orthogonal to the face. The word w' is a serialized encoding of words of the form $(\{0,1\}^*)^{n-m-1}$. For each word w_j in the serialized encoding, w_j delimits an interval $[t_j, r_j]$ inside the space spanned by a vector $\vec{z}_j \in Z$.

The region corresponding to the set of points of the space spanned by Z that have an encoding prefixed by w is the smallest cone that has $\vec{0}$ as apex, and that contains the interval $[t_j, r_j]$ for all $j \in \{0, \ldots, n-m-1\}$.

This cone C is generated by $(n-m-1)$ pairs of linear constraints, one for each element of Z not orthogonal to the face. If the face is identified by $+i$, we have :

$$C = \left\{ (p_1,\ldots,p_{n-m}) \in \mathbb{R}^{n-m} \mid \bigwedge_{j=0, j\neq i}^{n-m} \left(t_j p_i - \frac{1}{2} p_j \geq 0 \ \wedge \ r_j p_i - \frac{1}{2} p_j \leq 0 \right) \right\}$$

Otherwise, if the face is identified by $-i$, we have :

$$C = \left\{ (p_1,\ldots,p_{n-m}) \in \mathbb{R}^{n-m} \mid \bigwedge_{j=0, j\neq i}^{n-m} \left(t_j p_i - \frac{1}{2} p_j \leq 0 \ \wedge \ r_j p_i - \frac{1}{2} p_j \geq 0 \right) \right\}$$

To create the path region $R_{V,w}$ from the conical set C, one has to express this conjunction of linear constraints into the Cartesian coordinate system generating $\mathbb{R}^n$.

First, we remark that the inequation

$$ap_i + bp_j \geq 0$$

is equivalent to

$$0p_1 + \ldots + ap_i + \ldots + bp_j + \ldots + 0p_{n-m} \geq 0$$

and that its orientation is characterized by the vector

$$(0,\ldots,a,\ldots,b,\ldots,0) \in Z$$

The same vector expressed in $V + Z = \{\vec{v}_1,\ \vec{v}_2,\ \ldots,\ \vec{v}_m,\ \vec{z}_1,\ \vec{z}_2,\ \ldots,\ \vec{z}_{n-m}\}$ becomes a similar vector augmented with m zeros for the first m variables. We obtain :

$$\vec{d} = (\underbrace{0, \ldots, 0}_{m}, 0, \ldots, a, \ldots, b, \ldots, 0)$$

This also characterizes the orientation of a linear constraint, but in order to express it in $E = \{(1, 0, \ldots, 0), (0, 1, \ldots, 0),\ \ldots\ , (0, \ldots, 0, 1)\}$, we have to rotate the vector in order to align it with the m vectors of E that not belong to Z. The following matrix defines a coordinate change from E to $V + Z$:

$$\mathcal{M} = \begin{pmatrix} \vec{v}_1[1] & \vec{v}_2[1] & \ldots & \vec{v}_m[1] & \vec{z}_1[1] & \vec{z}_2[1] & \ldots & \vec{z}_{n-m}[1] \\ \vec{v}_1[2] & \vec{v}_2[2] & \ldots & \vec{v}_m[2] & \vec{z}_1[2] & \vec{z}_2[2] & \ldots & \vec{z}_{n-m}[2] \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \vec{v}_1[n] & \vec{v}_2[n] & \ldots & \vec{v}_m[n] & \vec{z}_1[n] & \vec{z}_2[n] & \ldots & \vec{z}_{n-m}[n] \end{pmatrix}$$

Since this is the inverse of the needed operation, the linear constraint we are searching is characterized by the vector $\mathcal{M}^{-1}\vec{d}$. Note that the matrix is not singular, as its columns are linearly independent vectors, by construction.

9.5 Operations

9.5.1 Minimization

The module responsible for the minimization of IRVA uses similar mechanisms as the Algorithm depicted in Section 6.3, with the exception of the fact that the IRVA undergoing the minimization operation is not modified. Instead, a new IRVA is constructed, corresponding to the canonical form of the input IRVA.

Let $\mathcal{A} = (n, S_I, S_E, s_0, \delta, \mathrm{VS}, \mathrm{col})$ be the input IRVA, and $\mathcal{M} = (n,\ {S_I}',\ {S_E}',\ {s_0}',\ \delta',\ \mathrm{VS}',\ \mathrm{col}')$ be the result of the minimization.

The procedure uses some additional data structures in order to simplify or speed up some computations.

In order to process the states in a correct order, a table associates a depth to each state of $\mathcal{A}$ determined by the computation of a topological ordering of the states of $\mathcal{A}$.

An adjacency matrix of implicit states of $\mathcal{A}$ is created. This matrix makes it possible to know, for a pair of states, if the second is reachable from the first.

During minimization, each processed state s is either kept or merged with an existing one. Keeping s means that a new state will have to be inserted inside $\mathcal{M}$. Merging s with an existing one means that nothing will be added to $\mathcal{M}$. In both cases, when another state of $\mathcal{A}$ that has s as a destination of one of its transitions is later examined, its destinations should be redirected to point either the new state of $\mathcal{M}$ if s has been kept, or to the corresponding

state of M in the case of a merge. An associative table $T : S_I \cup S_E \rightarrow {S_I}' \cup {S_E}'$ is used in order to keep track of the correspondence between states of $\mathcal{M}$ and $\mathcal{A}$.

The first minimization rule for explicit and implicit states (c.f. Section 6.3) locates already processed states in order to detect redundancies. For both explicit and implicit states, the procedure searches whether a state of the same type with the same successors is present or not in $\mathcal{M}$. Since the rule is applied to a state of $\mathcal{A}$, the successors of which are also states of $\mathcal{A}$, the table T is used in order to know the already processed states of $\mathcal{M}$ to which the destinations of this state correspond.

The implementation of the second minimization rule for implicit states requires two data structures. To know if an implicit state s will be added to $\mathcal{M}$, one has to search among its direct implicit successors (the set $\delta_I(s)$). In order to avoid to perform a depth-first search to construct $\delta_I(s)$, a list of implicit state is associated to each explicit state. This list is built incrementally in the following manner : at each step of the minimization procedure, when processing a state s having the destinations, s_1 and s_2, if s is implicit, then its associated list is $\{s\}$. If s is explicit, then its associated list is the union of the associated lists of s_1 and s_2. We use a data structure based on linked lists in order to merge lists in one simple operation. Lists can share elements and are just a pair of head and tail pointers into a pool of linked elements. Once the set $\delta_I(s)$ has been built, the procedure checks whether a minimal element exists or not by using an adjacency matrix of implicit states of $\mathcal{M}$. This matrix is maintained dynamically and is updated when new implicit states are added to $\mathcal{M}$.

The second minimization rule for explicit states requires a slightly different approach. The idea of the rule is to detect, for an explicit state s, if it is redundant and can be removed. It is the case when there exists a minimal element in the set of reachable implicit states from s. In the algorithm depicted in Section 6.3, we use this rule literally for the sake of simplicity. However, for the implementation, we are able to restrict the search thanks to the following property. If S is the set of reachable implicit states of s, since all successors of s have already been processed in a previous step, we know that the minimum element, if it is defined, will be a successor of s. This is due to the fact that if all explicit states created in $\mathcal{M}$ are not useless, hence a new explicit state that has two explicit states as destinations is bound to be useful, too. Moreover, if the processed explicit states are useful, then it means that for each of them, no minimum element is defined among their reachable set of implicit states. Merging two sets of implicit states that both have no minimum element defined, cannot produce a set with a minimum element. On the other hand, when an implicit state is a successor of the currently processed explicit state, adding this state to the set of reachable implicit states can change the situation, and a minimum element may be defined. This minimum element is bound to be the newly considered implicit state.

As a consequence, we conclude that the second minimization rule for explicit states only applies when one of the successors is implicit, and that it suffices to test whether all implicit states reachable from the other successor of s are also reachable from s'. If it is the case, then s' is the minimum element, and s can be considered useless. Again, the test of reachability is performed by using the already constructed adjacency matrix of implicit states of $\mathcal{A}$.

When an implicit state is considered to be useless by the first minimization rule for implicit states, it is possible that some already created explicit state of $\mathcal{M}$ becomes unreachable. Such states can be eliminated by a simple post processing that only keeps reachable states, for instance by using a depth-first search from the initial state.

9.5.2 Boolean Combination

The product is defined upon the principle that a state in the product of two IRVA is a pair of states, one for each operand. This is impractical, since it would require to store additional information inside the product IRVA. In the implementation, we avoided the modification of the data structure by keeping the pair inside the stack. Indeed, when a state of the product automaton is pushed on the stack in order to develop it in a future step of the algorithm, the pair of states used for its creation is pushed with it.

Later in the procedure, before creating an implicit state, we have to search the product IRVA in order to know whether the state already exists. We use a simple hash map with the pair of state as key and a state of the product as value in order to quickly determine its existence in the product.

9.5.3 Projection

Our implementation is only able to perform aligned projections (however, it could easily be generalized to unrestricted projections by adapting the procedure for projecting vector spaces, as explained in Section 8.2.

Let $\mathcal{A}$ be the input IRVA undergoing projection and $\mathcal{A}_{|\neq i}$ be the projected IRVA w.r.t. i.

The projection of the vector spaces associated to the implicit states of $\mathcal{A}$ is done by deleting the row corresponding to the coordinate component that is projected away and then by minimizing the set of column vectors in order to define a vector basis of a new vector space. First, all zero vectors are removed. Then, Gauss-Jordan elimination is used in order to keep only a matrix formed of linearly independent columns vectors.

Recall that an implicit state of $\mathcal{A}_{|\neq i}$ is a composite state formed of implicit states of $\mathcal{A}$. Similarly to the product, in order to avoid defining a new type of composite states, a set of states is associated to each implicit state of $\mathcal{A}_{|\neq i}$ by using an associative table.

When looking if an implicit state of $\mathcal{A}_{|\neq i}$ already exists, a hash map is used with the set as key and the corresponding state of $\mathcal{A}_{|\neq i}$ as value.

Inside the projection procedure, the set of implicit states we use as composites are always sets of minimal states as explained in Section 8.1.4. Those sets of minimals result in a union of several sets calculated by successive calls to the *minimals* algorithm.

In order to create and maintain this set of minimal states, we use the data structure presented in Section 9.3 for the calculation of sets of minimal states.

9.6 Results

In this section we show different case studies in order to observe the behavior of our prototype implementation.

The first two case studies are designed to emphasize a particular property of the data structure. With the first one, we see how the size of an IRVA can vary with rotations of the represented polyhedron, in order to verify that IRVA represent an improvement over RVA in that aspect.

With the second case study, we observe how the representation is affected when creating hypercubes with increasing number of dimensions.

The purpose of the other case studies are to test the behavior of the data structure in typical applications. In order to make observations as independent as possible from particular values of inputs, we repeat similar sequences of operations several times with a variation of those values.

For example, the third case study checks how the creation and projection of a pyramid in $\mathbb{R}^3$, with a triangular section, can vary with a variation of the orientations of its faces. Note that this is not similar to the first case study where the size of the pyramid is invariant and the change of orientation is only performed by rotation around a unique axis.

Giving the complete characterization of the polyhedra used in each case study would require an impractical amount of space in this manuscript. We will only describe the rules behind their construction. The complete data is available at `http://www.montefiore.ulg.ac.be/~degbomont/phdthesis`.

9.6.1 Methodology

In each of the following tests, we use a notion of *steps* to quantify the amount of work required to execute an operation. Those steps can be understood as the number of times a state of an IRVA – explicit or implicit – is considered, even if no action is performed. The number of steps of an execution is affected by the number of states considered in algorithms, in path region constructions, during the search for minimum covered element or construction of sets of minimal states.

Each of the following sections describe a particular case study. In each section, a table provides experimental results for each test.

The different case studies have been executed on a Pentium Dual-Core E5400 running at 2.70GHz with 4GB of RAM. We use the GNU Compiler Collection 4.4.3 (GCC) on Linux. We linked with the the Parma Polyhedra Library (PPL) in its version 0.10.2 and the GNU Multi-Precision (GMP) 3.5.2.

9.6.2 Rotation of a Pyramid in $\mathbb{R}^3$ with Square Section

In this case study, we emphasize the stability of the size of the data structure with respect to rotations. Indeed, IRVA have been developed in order to tackle the problem of the blowup in number of states of the RVA data structure with respect to large coefficients of linear constraints.

We stimulate the growing of coefficients of linear constraints defining an IRVA by constructing the representing pyramid in $\mathbb{R}^3$ of a square of size 4, centered on $(-5, 12)$, rotated by a varying angle.

The following table summaries the execution of the creation of the pyramid. Each row of the table contains information about the creation of the IRVA representing the pyramid. This IRVA is created by intersecting four IRVA representing half-spaces. Those half-spaces correspond to the rotated sides of the pyramid. The intersection is performed incrementally and each intermediate result is minimized.

In this table, the column *Angle* gives the angle used for the rotation ; *Steps* gives the number of steps taken to execute the operation, *t(s)* is the time, expressed in seconds, taken to perform the operation ; S_I and S_E give respectively the number of implicit states and explicit states of the resulting IRVA ; $S_{I\,max}$ and $S_{E\,max}$ show the maximum number of, respectively, implicit and explicit states of intermediate IRVA. Finally the column *vmem* corresponds to the maximum amount of virtual memory allocated by the process running the creation. It is expressed in kilobytes.

Angle	Steps	t(s)	S_I	S_E	$S_{I\,max}$	$S_{E\,max}$	vmem (kB)
0	5830	0.859	11	40	19	341	152
1	5842	0.873	11	36	19	343	152
7	4116	0.596	11	40	19	229	152
11	4872	0.660	11	31	19	282	152
13	4817	0.613	11	28	19	279	152
19	3052	0.409	11	24	19	160	152
21	2985	0.411	11	29	19	155	152
22	3354	0.482	11	46	19	180	152
72	4974	0.778	11	30	19	285	152
324	3120	0.479	11	37	19	163	152

We can see that the angle has an impact on the results, but this impact is minor, as expected. The observed variations are a consequence of the fact that decision structures are adapted to match the different orientations.

9.6.3 Pyramids of Increasing Dimensions

In this section, we show the impact of the dimensionality on the size of an IRVA corresponding to the representing pyramid of a unit hypercube centered on $\vec{0}$.

In the following table, we show the summary of the creation of each representing pyramid in $\mathbb{R}^2$, $\mathbb{R}^3$, $\mathbb{R}^4$, $\mathbb{R}^5$, $\mathbb{R}^6$ and $\mathbb{R}^7$. The column *Dim.* gives the dimensionality.

Dim.	Steps	t(s)	S_I	S_E	$S_{I\,max}$	$S_{E\,max}$	vmem (kB)
2	93	0.006	5	1	9	2	152
3	658	0.151	11	3	17	4	536
4	3563	1.573	29	7	41	10	2504
5	22134	19.010	83	15	113	22	9732
6	207888	289.585	245	31	329	46	36608
7	3259705	5518.931	731	63	977	94	43556

Unsurprisingly, the number of implicit states grows exponentially[1] with the dimensionality. Notice that, since we represent pyramids, the number of implicit states for a pyramid of $\mathbb{R}^n$ corresponds to the number of polyhedral component of the $(n-1)$-dimensional hypercube.

The other measures, such as steps, time and memory follow similar progression. This emphasize a drawback of using IRVA for representing high dimensional polyhedra.

Some ideas to tackle this particular problem will be discussed later in Section 10.2.

9.6.4 Incremental Building of a Pyramid in $\mathbb{R}^4$

From now on, each case study is a repetition of similar operations with randomized input values. A global analysis is given in the Section 9.6.10.

In this case study, we investigate the behavior of the minimization operation when performed on IRVA of various sizes. In order to obtain such a varied, but realistic, set of executions, we analyze the creation of a $4-$dimensional pyramid with a cuboid section that corresponds to the representing pyramid of :

$$[6, 12] \times [-7, -1] \times \left[-\frac{15}{2}, -\frac{1}{2}\right]$$

[1] A hypercube of $\mathbb{R}^n$ already has 2^n vertexes.

The pyramid is constructed incrementally as the intersection of six half-spaces H_1, H_2, H_3, H_4, H_5 and H_6, one for each face of the cuboid. This intersection is computed incrementally, each intermediate result being minimized before proceeding further.

The following table summaries the execution of the creation of the pyramid. The column *Op.* specifies the nature of the operation,

The operation $P := A \cap B$ denotes the creation of an IRVA P as the result of the intersection of two IRVA A and B. The operation $mini(P)$ denotes the operation of minimizing the intermediate IRVA P.

Op.	Steps	t(s)	S_I	S_E
$P_1 := H_1 \cap H_2$	60	0.01	9	0
$mini(P_1)$	36	0.01	5	0
$P_2 := P_1 \cap H_3$	182	0.04	15	0
$mini(P_2)$	116	0.01	9	0
$P_3 := P_2 \cap H_4$	272531	97.86	27	25351
$mini(P_3)$	129968	0.04	17	136
$P_4 := P_3 \cap H_5$	245822	243.55	31	2838
$mini(P_4)$	16967	0.03	21	186
$P_5 := P_4 \cap H_6$	185491	191.46	41	1628
$mini(P_5)$	9507	0.05	29	198

In these results, we can remark that the intersection of H_4 with P_2 takes significantly more steps and time than the intersection of H_1, H_2 and H_3. This is due to the fact that P_3 has significantly more states than P_2 and H_4.

We see that the subsequent minimization $mini(P_4)$ does, however, reduce drastically the size of P_3, since it removes 25215 explicit states and 10 implicit states.

This shows that performing Boolean combinations as a product and a coloring function to be, in some situations, needlessly complex.

9.6.5 Union of Pyramids in $\mathbb{R}^2$

Here, we create ten IRVA U_1, U_2, ..., U_{10}. To create each IRVA, we first build twenty IRVA representing random $2-$dimensional pyramids, and then compute the incremental union of those twenty pyramids, minimizing each intermediate result.

Each random pyramid is created as follows. Two random half-spaces are created. The first one is chosen randomly but the second one is chosen in a way such that the two boundaries of the half-spaces form an angle varying between five and twenty degrees. Those two half-spaces are intersected and the result is minimized.

The following table summarizes the execution of the constructions.

Twenty operations are showed in the rows of this table. Each row identified by *Pyramids i* corresponds to the summary of the creation of the twenty random pyramids that are used to create U_i, with $i \in \{1, \ldots, 10\}$. The row identified by *Create* U_i is the summary of the incremental union of the twenty pyramids used to create the IRVA U_i, with $i \in \{1, \ldots, 10\}$.

Note that, since the operation *Pyramids i* builds twenty different IRVA, we do not specify values for columns S_I and S_E.

Op.	Steps	t(s)	S_I	S_E	$S_{I\,max}$	$S_{E\,max}$	vmem (kB)
Pyramids 1	3273	0.206	–	–	9	8	544
Create U_1	14667	2.023	25	23	26	24	1448
Pyramids 2	3545	0.212	–	–	9	10	544
Create U_2	15357	2.298	23	30	29	34	1320
Pyramids 3	2980	0.159	–	–	9	8	544
Create U_3	12314	1.605	21	18	25	23	1324
Pyramids 4	3279	0.166	–	–	9	8	544
Create U_4	12352	1.681	23	24	25	27	1328
Pyramids 5	3142	0.175	–	–	9	10	544
Create U_5	11124	1.452	21	16	23	22	1320
Pyramids 6	2979	0.153	–	–	9	10	544
Create U_6	10524	1.456	18	19	21	23	1184
Pyramids 7	2979	0.167	–	–	9	8	544
Create U_7	11085	1.479	17	13	23	23	1188
Pyramids 8	3060	0.165	–	–	9	8	544
Create U_8	14327	1.919	25	25	29	30	1460
Pyramids 9	3264	0.184	–	–	9	10	544
Create U_9	12106	1.609	21	17	25	22	1324
Pyramids 10	2874	0.176	–	–	9	6	536
Create U_{10}	9427	1.194	19	17	21	17	1184

9.6.6 Pyramids in $\mathbb{R}^3$ with Triangular Section

In this case study, we create ten IRVA A_1, A_2, …, A_{10}. Each of these ten IRVA is created by the incremental intersection of three randomized half-spaces in $\mathbb{R}^3$ such that their intersection corresponds to a pyramid with a triangular section.

The operation noted *Create* A_i, with $i \in \{1, 2, \ldots, 10\}$ is the incremental construction of the pyramid A_i with each intermediate result minimized.

The second operation, noted $A_{i|\neq 2}$ is the projection of P_i w.r.t. 2, followed by a minimization.

Op.	Steps	t(s)	S_I	S_E	$S_{I\,max}$	$S_{E\,max}$	vmem (kB)
Create A_1	1549	0.216	9	9	15	61	148
$A_{1\|\neq 2}$	140	0.061	5	0	7	3	148
Create A_2	2170	0.264	9	10	15	106	148
$A_{2\|\neq 2}$	279	0.132	5	2	7	5	148
Create A_3	3355	0.616	9	41	15	148	148
$A_{3\|\neq 2}$	612	0.342	5	0	7	4	272
Create A_4	761	0.087	9	2	15	18	148
$A_{4\|\neq 2}$	107	0.047	5	0	7	3	148
Create A_5	4770	0.395	9	61	15	288	148
$A_{5\|\neq 2}$	653	0.374	5	3	5	3	272
Create A_6	2860	0.394	9	20	15	134	148
$A_{6\|\neq 2}$	209	0.102	5	2	5	2	148
Create A_7	546	0.054	9	2	15	10	148
$A_{7\|\neq 2}$	119	0.047	5	0	7	3	148
Create A_8	4063	0.500	9	46	15	212	148
$A_{8\|\neq 2}$	339	0.179	5	0	7	4	148
Create A_9	49109	9.732	9	65	15	2485	408
$A_{9\|\neq 2}$	510	0.312	5	2	7	3	412
Create A_{10}	501	0.064	9	1	15	20	148
$A_{10\|\neq 2}$	43	0.022	5	0	5	0	148

9.6.7 Union of Half-Lines in $\mathbb{R}^3$

We create ten IRVA. Each of them is constructed as the union of a random number of half-lines in $\mathbb{R}^3$. Each ray leaves $\vec{0}$ with a random direction characterized by a point chosen inside $[-100, 100]^3$.

The operation noted $R(x)$ in the following table is the construction of an IRVA corresponding to the incremental union of x half-lines, with intermediate results minimized.

Op.	Steps	t(s)	S_I	S_E	$S_{I\,max}$	$S_{E\,max}$	vmem (kB)
$R(86)$	808102	373.845	88	242	88	242	15660
$R(59)$	364632	167.114	61	178	61	178	7900
$R(16)$	16729	6.290	18	34	18	34	920
$R(58)$	362715	164.477	60	182	60	182	7568
$R(17)$	24414	9.786	19	52	19	52	972
$R(32)$	86377	36.204	34	90	34	90	2652
$R(15)$	19794	7.982	17	46	17	46	752
$R(21)$	35004	14.232	23	66	23	66	1364
$R(60)$	399816	183.980	62	184	62	184	8048
$R(6)$	2335	0.745	8	12	8	12	148

9.6.8 Pyramids in $\mathbb{R}^4$ with Cuboid Section

Here is a summary of the results of the creation of ten IRVA representing each a pyramid $\mathbb{R}^4$ with a cuboid section.

Each pyramid is constructed in a similar way than the creation of the pyramid described in Section 9.6.4, with intervals each chosen randomly inside $[-20, 20]$.

In the following table, the operation *Create* C_i, with $i \in \{1, 2, \ldots, 10\}$, corresponds to the incremental construction of the representing pyramid of an axis-aligned cuboid centered on a point randomly chosen inside $[-20, 20]^3$ having sides of size randomly chosen inside $[1, 10]$. Again, each intermediate IRVA is minimized.

Op.	Steps	t(s)	S_I	S_E	$S_{I\,max}$	$S_{E\,max}$	vmem (kB)
Create C_1	681532	341.955	29	408	41	15001	2288
Create C_2	542246	404.304	29	283	41	7005	1188
Create C_3	860680	552.672	29	198	41	25351	3424
Create C_4	207204	88.851	29	114	41	5747	948
Create C_5	1221103	784.026	29	410	41	23193	3284
Create C_6	2983598	1809.077	29	426	41	92941	11704
Create C_7	3116966	2248.521	29	293	41	35835	4676
Create C_8	330090	145.487	29	101	41	6766	1224
Create C_9	147345	60.589	29	168	41	5570	1176
Create C_{10}	88314	12.867	29	251	41	3571	864

We can observe a large variation in the cost of each creation of pyramids, since the creation of C_7 is a little more than 174 times slower than the creation of C_{10}.

9.6.9 Projection of Pyramids in $\mathbb{R}^4$

We now analyze the behavior of the projection procedure with 4-dimensional inputs : we project w.r.t. 1 the pyramids created in Section 9.6.8 and minimize the results.

In the following table, columns *input* S_I and *input* S_E give the number of, respectively, implicit and explicit states of the input IRVA of each operation. The row identified by $C_{i|\neq 1}$ of the following table corresponds to the projection w.r.t. 1 of the pyramid C_i. The row identified by $mini(C_{i|\neq 1})$ corresponds to the minimization of the projected IRVA $C_{i|\neq 1}$.

Op.	Steps	t(s)	input S_I	input S_E	S_I	S_E	vmem (kB)
$C_{1\|\neq 1}$	50050	53.9	29	408	11	88	–
$mini(C_{1\|\neq 1})$	546	0.01	11	88	11	58	740
$C_{2\|\neq 1}$	35125	38.211	29	283	11	60	–
$mini(C_{2\|\neq 1})$	407	0.01	11	60	11	42	704
$C_{3\|\neq 1}$	6989	8.635	29	198	11	11	–
$mini(C_{3\|\neq 1})$	175	0.01	11	11	11	11	584
$C_{4\|\neq 1}$	1439	1.692	29	114	11	1	–
$mini(C_{4\|\neq 1})$	116	0.01	11	1	11	1	484
$C_{5\|\neq 1}$	61064	63.91	29	410	11	54	–
$mini(C_{5\|\neq 1})$	395	0.01	11	54	11	40	704
$C_{6\|\neq 1}$	54763	66.871	29	426	11	34	–
$mini(C_{6\|\neq 1})$	282	0.01	11	34	11	30	760
$C_{7\|\neq 1}$	46026	54.363	29	293	11	47	–
$mini(C_{7\|\neq 1})$	321	0.01	11	47	11	31	668
$C_{8\|\neq 1}$	7027	8.191	29	101	11	37	–
$mini(C_{8\|\neq 1})$	290	0.01	11	37	11	27	700
$C_{9\|\neq 1}$	21264	22.31	29	168	11	43	–
$mini(C_{9\|\neq 1})$	315	0.01	11	43	11	32	696
$C_{10\|\neq 1}$	38091	41.26	29	251	11	35	–
$mini(C_{10\|\neq 1})$	284	0.01	11	35	11	30	660

This case study also shows a variation in the cost of each projection, since the computation of $C_{6|\neq 1}$ is a little more than 39 times slower than the computation of $C_{4|\neq 1}$.

While the number of states of the output seems to play an important role in these results, ($C_{4|\neq 1}$ contains one explicit state and $C_{6|\neq 1}$ has twenty-four), it does not totally depend on it, since the computation of $C_{9|\neq 1}$ is faster than the one of $C_{10|\neq 1}$ and they produce, respectively, IRVA of 43 and 35 explicit states.

9.6.10 Analysis

As pointed out in case studies of Sections 9.6.4, 9.6.8 and 9.6.9 we have observed, in those case studies, an important variations – in terms of number of steps, and time taken – between the executions of two similar operations with a slight perturbation of the input values.

A possible explanation for this variation lies in the fact that when linear constraints are close, deciding between the two on one face of the hypercube of normalization can require long encodings. This does not, however, totally explain the variations on the observations, since in most cases, the number of states of the corresponding minimized IRVA does not grow in proportion.

It remains an open problem that should be addressed in future work.

Nevertheless, from the observation made in Section 7.4.3 about the heavy use of the **minimals** function by the product and projection algorithms, we can propose the following ideas to reduce the cost of this function.

An implementation of an incremental version of the **minimals** algorithm, as discussed in Section 7.4.3, would certainly decrease the overall cost of the computations of product and projection operations, since they mainly rely on this function.

Furthermore, trying to reduce the cost of the operations carried out by the **minimals** function would certainly be beneficial, too. In particular, in the **minimals** procedure, one often needs to test whether two path regions – or a path region and a vector space – have a non empty intersection. As mentioned, this is done by the manipulation of convex polyhedra, via the *Non Necessarily Closed Polyhedron* [BRZH02] data structure available in the *PPL library* [BHZ08].

These manipulations, in the current implementation, consist in the construction of the convex polygon corresponding to the intersection of the two convex polyhedra followed by a test for emptiness.

Since path regions and vector spaces are easily generated by conjunctions of linear constraints, simplex-based approach and satisfiability test [DM06] can probably accelerate the procedure substantially. Indeed, in the **minimals** algorithm, in order to cut the search tree as soon as possible, a test for empty intersection is performed between the input region R (in the calling contexts, this region is always created as a path region) and a cover region corresponding to the current path. In the same procedure, when an implicit state is reached, a similar test is carried out between its vector space and the intersection of the input region and the path region defined by the path that led to this implicit state. The intersection of two path regions is still representable by a conjunction of linear constraints.

Chapter 10

Conclusions

The symbolic representation of polyhedra is essential to many applications such as computer-aided verification, computer-aided design, calculation of trajectories or physical simulations. Depending on the context, some features of polyhedra can be more critical than others. For example, for verification of systems, represented polyhedra can have an arbitrary number of dimensions.

The class of Nef Polyhedra [Nef78, BN88] corresponds to the polyhedra considered in this work. This class contains convex and non-convex polyhedra, including features such as open and closed boundaries, unconnected parts and non-manifold components.

We have introduced an original data structure, the Implicit Real Vector Automaton (IRVA), that is expressive enough for representing symbolically polyhedra. IRVA are not limited to regularized polyhedra [Req80, Män88, Hof89, Bru90] and are closed under Boolean operators and projection.

IRVA imitate the principles of Real Vector Automata (RVA) [BBR97], another automata-based data structure suited for polyhedra, but their advantage is to be considerably more concise.

An important characteristic of IRVA is that they admit an easily computable canonical form that turns out to also correspond to their minimal form. Thanks to the existence of such a canonical form, testing equality between two polyhedra represented by IRVA reduces to a simple and efficient syntactic test. Moreover, having a minimal form enables to keep the size of intermediate results under control when performing large series of manipulations on the data structure.

A very efficient algorithm make it possible to test whether or not a given point belongs to a polyhedron represented by an IRVA . A similar algorithm can also be used to detect to which polyhedral component this point belongs. This particular property makes the IRVA data structure a good data structure for performing collision detection or operations such as ray tracing.

Representing elementary IRVA corresponding to linear constraints or vector spaces is straightforward. An algorithm has been developed to compute the product of two IRVA. When a coloring function is applied to the product, it

can represent the result of arbitrary Boolean combinations.

Furthermore, we have also developed an algorithm for computing the projection of a represented polyhedron. In its actual form, it can compute projections that correspond to the removal of variables, but the algorithm is straightforwardly generalizable in order to perform arbitrary projections. Such operations are useful in the context of symbolic state space exploration techniques, in particular for Hybrid Systems [ACH+95].

An implementation of the algorithms presented in this work has been developed in order to experiment further with IRVA. This implementation, although mainly intended to provide a proof of concept, already make it possible to carry out operations that were impossible to perform with RVA.

10.1 Relation with Other Work

We have mentioned that IRVA are not limited to regularized polyhedra and are closed under Boolean operators. This is a clear advantage when compared with data structures such as Boundary Representations (B-REP). Indeed, B-REP representation fail at representing polyhedra that do not correspond to physical entities (with a non empty volume and in three dimensions). On the other hand, unlike the B-REP approach, IRVA are not suited for direct characterization of polyhedral components, nor for displaying polyhedra using a drawing method such as *rasterisation* [Pav79].

IRVA share some similarities with Selective Nef Complexes [GHH+03], which also decompose polyhedra into pyramids, and represent the incidence relation between them. IRVA have two original features : first, the general and deterministic decision structures that link implicit states. Second, IRVA do not rely on ad hoc mechanisms that depend on dimensionality to represent and link components. This provides the ability to represent any n-dimensional polyhedra, instead of being limited to two or three dimensions and provides a deterministic and efficient procedure for solving the point decision problem.

IRVA inherit from RVA the efficient algorithms for solving deterministically the point decision and classification problems. Their conciseness, compared to RVA, is due to the introduction of implicit states that replace some internal structures of RVA with algebraic information representing them. IRVA are, on the other hand less expressive than RVA whose expressiveness correspond to the additive theory of mixed integer and real variables.

Finally IRVA admit an easily computable canonical form. Data structures like *Cell Decomposition* or *Binary Space Partition* also admit a canonical form, but its definition requires the ability to canonize a set of cutting planes and the application of a lexical ordering of generators, which is a difficult problem [Tam07].

10.2 Perspectives

We now discuss some questions that could be addressed in future work.

The Curse of Dimensionality

The size of the structure grows exponentially with the dimensionality of the represented set. Although not limited in dimension by definition, IRVA are, in practice, often not suited to represent sets with a large number of dimensions, depending on the application. For example, a n-dimensional hypercube has 2^n vertexes, which translates to at least 2^n implicit states inside an IRVA.

In order to tackle this problem, a promising direction could be to adapt to IRVA the reduction techniques proposed in [BD97]. Since the transition relation of IRVA is acyclic, this reduction is presumably feasible. With such a reduction technique, the data structure should become more concise for polyhedra with large number of dimensions.

However, this reduction relies on detecting independences among sets of variables. This is incompatible with the fact that the current definition of IRVA is limited to representing only pyramids. Indeed, with the exception of some trivial sets, pyramids do not show such independences among their variables.

While this problem certainly requires a deeper investigation, we can already give a pointer to two options in order to address it :

A first idea is to develop a new version of IRVA able to represent polyhedra – and polyhedral partitions – and not just pyramids. This seems achievable, although a lot of practical problem quickly arise, in particular in the context of the product operation.

A second solution is to adapt the notion of independences of variables to account for pyramidal structures.

Import and Export Operation

Another future work could also address the problem of conversions to and from other representations. Currently, IRVA have to be defined either directly, or by performing a succession of Boolean combinations and projections. Having ways to straightly interact with other representations is probably mandatory in order to be able to use IRVA in frameworks like computer-aided design.

Other Operations

We have seen in Chapter 8 that the generalized projection associated with advanced coloring functions pave the way for a visualization tool for IRVA. Further more, ray tracing techniques could be adapted to IRVA, thanks to the deterministic decision structures of IRVA. Once again, being able to dynamically visualize IRVA would certainly make them more appealing for computer-aided design tools.

Algorithm Improvements

The product and projection algorithms rely heavily on a sub routine that computes the minimum covered element of an IRVA by a region of space. The actual algorithm already have mechanisms for avoiding some useless computations, but could probably be operationally improved.

In general, a more optimized software package than the prototype implementation mentioned in this work would certainly be useful.

Tools for Verification

In computer-aided verification of Hybrid Systems [ACH+95], linear hybrid systems have a transition relation that can be described by combination of linear constraints, called *Linear Hybrid Relations (LHR)*.

In this context, an essential problem consists in computing the acceleration of a LHR, in other words its transitive closure.

Since polyhedra can be used to represent symbolically a LHR, and since IRVA provide a simple mechanism for classifying of topological elements (such as vertexes, edges, faces, ...), an open problem is to exploit this classification by IRVA in order to compute those accelerations.

Planning of Trajectories

In the field of robotics, when a robot has to move inside an environment without following precomputed trajectories, one has to prevent collisions between the robot and obstacles.

A solution consists in building a first polyhedron to represent the environment and a second one to represent the possible configurations of the robot, and then to check whether those polyhedra have a non empty intersection or not.

Since IRVA can represent exactly arbitrary polyhedra and can be intersected efficiently, they could be used in this context.

Sets of Mixed Integer and Real Variables

The expressiveness of RVA corresponds to the additive theory of mixed integer and real variables. This theory includes polyhedra, but also makes it possible to define sets including discrete features such as periodicities.

A challenging problem is to generalize IRVA in order to make them as expressive as RVA.

Combination of Polynomial Constraints

Another interesting question is to see if the principles of IRVA can be adapted in order to represent sets corresponding to Boolean combinations of polynomial constraints, by for instance, the technique introduced in [MOS04].

Of course, such a generalization is not straightforward and requires the encoding scheme used in the data structure to be completely redefined.

Bibliography

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[Ake78] S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, 27(6):509–516, 1978.

[AKNS95] P. J. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors. *Hybrid systems II*, London, UK, 1995. Springer-Verlag.

[BBB10] B. Boigelot, J. Brusten, and V. Bruyère. On the sets of real numbers recognized by finite automata in multiple bases. *Logical Methods in Computer Science*, 6(1), 2010.

[BBD10] B. Boigelot, J. Brusten, and J.-F. Degbomont. Implicit Real Vector Automata. In *Proc. International Workshop on Verification of Infinite-State Systems*, volume 39 of *Electronic Proceedings in Theoretical Computer Science*, pages 63–76, Singapore, 2010.

[BBD12] B. Boigelot, J. Brusten, and J.-F. Degbomont. Automata-based symbolic representations of polyhedra. In *Proc. Language and Automata Theory and Applications*, volume 7183 of *Lecture Notes in Computer Science*, pages 3–20. Springer, 2012.

[BBL09] B. Boigelot, J. Brusten, and J. Leroux. A generalization of Semenov's theorem to automata over real numbers. In *Proc. Conference on Automated Deduction*, volume 5663 of *Lecture Notes in Computer Science*, pages 469–484. Springer, 2009.

[BBR97] B. Boigelot, L. Bronne, and S. Rassart. An improved reachability analysis method for strongly linear hybrid systems. In *Proc. Computer-Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 167–177. Springer-Verlag, 1997.

[BD97] V. Bertacco and M. Damiani. The disjunctive decomposition of logic functions. In *Proc. International Conference on Computer-Aided Design*, pages 78–82. IEEE Computer Society, 1997.

[BHZ08] R. Bagnara, P. M. Hill, and E. Zaffanella. The parma polyhedra library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1-2):3–21, 2008.

[BJW05] B. Boigelot, S. Jodogne, and P. Wolper. An effective decision procedure for linear arithmetic over the integers and reals. *ACM Transactions on Computational Logic*, 6(3):614–633, 2005.

[BMP99] O. Bournez, O. Maler, and A. Pnueli. Orthogonal polyhedra: Representation and computation. In *Proc. Hybrid Systems: Computation and Control*, volume 1569 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 1999.

[BN88] H. Bieri and W. Nef. Elementary set operations with d-dimensional polyhedra. In *Workshop on Computational Geometry*, volume 333 of *Lecture Notes in Computer Science*, pages 97–112. Springer, 1988.

[Bru90] B. Bruderlin. Robust regularized set operations on polyhedra. Technical Report UUCS-90-004, University of Utah, Salt Lake City, UT 84112, 1990.

[Bru11] J. Brusten. *On the sets of real vectors recognized by finite automata in multiple bases*. PhD thesis, University of Liège, 2011.

[BRW98] B. Boigelot, S. Rassart, and P. Wolper. On the expressiveness of real and integer arithmetic automata. In *Proc. International Colloquium on Automata, Languages and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 152–163, Aalborg, July 1998. Springer.

[Bry92] R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[BRZH02] R. Bagnara, E. Ricci, E. Zaffanella, and P. M. Hill. Possibly not closed convex polyhedra and the Parma Polyhedra Library. In *Proc. International Static Analysis Symposium*, volume 2477 of *Lecture Notes in Computer Science*, pages 213–229, Madrid, Spain, 2002. Springer-Verlag, Berlin.

[Büc62] J. R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. International Congress on Logic, Methodoloy and Philosophy of Science*, pages 1–12, Stanford, 1962. Stanford University Press.

[CGP00] E.M. Clark, O. Grumberg, and D.A. Peled. *Model checking*. The MIT Press, Cambridge, MA, 2000.

[DM06] B. Dutertre and L. De Moura. A fast linear-arithmetic solver for dpll(t). In *Proc. Computer Aided Verification*, pages 81–94, Berlin, Heidelberg, 2006. Springer-Verlag.

[DMY93] K. Dobrindt, K. Mehlhorn, and M. Yvinec. A complete and efficient algorithm for the intersection of a general and a convex polyhedron. In *Proc. Workshop on Algorithms and Data Structures*, volume 709 of *Lecture Notes in Computer Science*, pages 314–324. Springer, 1993.

[Edm60] J. R. Edmonds. *A combinatorial representation for oriented polyhedral surfaces.* University of Maryland, 1960.

[EK08] J. Eisinger and F. Klaedtke. Don't care words with an application to the automata-based approach for real addition. *Formal Methods in System Design*, 33(1–3):85–115, 2008.

[GHH+03] M. Granados, P. Hachenberger, S. Hert, L. Kettner, K. Mehlhorn, and M. Seel. Boolean operations on 3d selective nef complexes: data structure, algorithms, and implementation. In *Proc. European Symposium on Algorithms*, volume 2832 of *Lecture Notes in Computer Science*, pages 654–666. Springer, 2003.

[GN00] E. R. Gansner and S. C. North. An open graph visualization system and its applications to software engineering. *Software - Practice and Experience*, 30(11):1203–1233, 2000.

[Gra12] T. Granlund. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 2012. `http://gmplib.org/`.

[Hac07] P. Hachenberger. *Boolean operations on 3D selective Nef complexes: data structure, algorithms, optimized implementation, experiments and applications.* PhD thesis, Saarland University, 2007.

[HHK89] C. M. Hoffmann, J. E. Hopcroft, and M. E. Karasick. Robust set operations on polyhedral solids. *IEEE Computer Graphics and Applications*, 9(6):50–59, 1989.

[Hof89] C. M. Hoffmann. *Geometric and solid modeling : an introduction.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989.

[HRP94] N. Halbwachs, P. Raymond, and Y.E. Proy. Verification of linear hybrid systems by means of convex approximations. In *Proc. International Static Analysis Symposium*, volume 864 of *Lecture Notes in Computer Science*, pages 223–237. Springer-Verlag, 1994.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages and computation.* Addison-Wesley, 1979.

[KLSV10] D. K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager. The theory of timed i/o automata, second edition. *Synthesis Lectures on Distributed Computing Theory*, 1(1):1–137, 2010.

[LL01] S. H. Lee and K. Lee. Partial entity structure: a compact non-manifold boundary representation based on partial topological entities. In *Proc. Solid Modeling and Applications*, pages 159–170, New York, NY, USA, 2001. ACM.

[Löd01] C. Löding. Efficient minimization of deterministic weak ω-automata. *Information Processing Letters*, 79(3):105–109, 2001.

[Män88] M. Mäntylä. *Introduction to solid modeling*. W. H. Freeman & Co., New York, NY, USA, 1988.

[Mau91] A. Maulik. An efficient intersection algorithm for polyhedral cellular decompositions. In *Proc. of the first ACM symposium on Solid modeling foundations and CAD/CAM applications*, SMA '91, pages 109–118, New York, NY, USA, 1991. ACM.

[McC98] B. J. McCartin. Seven deadly sins of numerical computation. *The American Mathematical Monthly*, 105(10):929–941, 1998.

[McM93] K.L. McMillan. *Symbolic model checking*. Kluwer Academic Publishers, 1993.

[Mey00] C. D. Meyer, editor. *Matrix analysis and applied linear algebra*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[MOS04] M. Müller-Olm and H. Seidl. Computing polynomial program invariants. *Information Processing Letters*, 91(5):233–244, 2004.

[NAT90] B. Naylor, J. Amanatides, and W. Thibault. Merging bsp trees yields polyhedral set operations. *SIGGRAPH Computer Graphics*, 24(4), 1990.

[Nay90] B. F. Naylor. Binary space partitioning trees as an alternative representation of polytopes. *Computer-Aided Design*, 22(4):250–252, 1990.

[NB94] X. Ni and M. S. Bloor. Performance evaluation of boundary data structures. *IEEE Computer Graphics and Applications*, 14(6):66–77, 1994.

[Nef78] W. Nef. *Beiträge zur theorie der polyeder mit anwendungen in der computergraphik (Contributions to the theory of polyhedra, with applications in computer graphics)*. Beiträge zur Mathematik, Informatik und Nachrichtentechnik. H. Verlag and H. Lang and Cie. AG, Bern, 1978.

[Pav79] T. Pavlidis. Filling algorithms for raster graphics. *Computer Graphics and Image Processing*, 10(2):126–141, 1979.

[Per01] L. Perko. *Differential equations and dynamical systems.* Texts in Applied Mathematics. Springer, 2001.

[Pie91] L. Piegl. On nurbs: a survey. *IEEE Computer Graphics and Applications*, 11(1):55–71, 1991.

[Req80] A. G. Requicha. Representations for rigid solids: Theory, methods, and systems. *ACM Computing Surveys*, 12(4):437–464, 1980.

[Sha97] V. Shapiro. Maintenance of geometric representations through space decompositions. *International Journal of Computational Geometry and Applications*, 7(1/2):21–56, 1997.

[Sha02] V. Shapiro. *Handbook of Computer Aided Geometric Design*, chapter Solid Modeling, pages 473–518. Elsevier, Amsterdam, 2002.

[Tam07] J. Tammik. Autocad nef polyhedron implementation. Technical report, University of Bern, Bern, Switzerland, 2007.

[Var07] M. Vardi. The Büchi complementation saga. In *Proc. Symposium on Theoretical Aspects of Computer Science*, pages 12–22, Berlin, Heidelberg, 2007. Springer-Verlag.

[WB00] P. Wolper and B. Boigelot. On the construction of automata from linear arithmetic constraints. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1785 of *Lecture Notes in Computer Science*, pages 1–19. Springer Berlin Heidelberg, 2000.

[Wei85] K. Weiler. Edge-based data structures for solid modeling in curved-surface environments. *IEEE Computer Graphics and Applications*, 5(1):21–40, 1985.

[Wes78] C.H. West. Generalized technique for communication protocol validation. *IBM Journal of Research and Development*, 22(4):394–404, 1978.