Unbounded polyhedra conversion for reachability analysis

Owen Rouillé
Supervised by Stefan Schupp and Erika Ábráham

RWTH Aachen

Abstract. When a digital controller interacts with physical quantities, discrete behaviour is mixed with continuous behaviour. Such systems are called hybrid and they are omnipresent, e.g. planes, nuclear reactors, cars, just to name a few. Their analysis is hard and crucial, as they are often safety-critical. One way to ensure the safety of these systems is the reachability analysis via flowpipe construction: the construction of an over-approximation of all reachable states of the system within a possibly bounded time horizon. This method strongly relies on geometry for the representation of the states. This paper is a contribution to the HyPro project, a library dedicated to the state representation of hybrid systems by a variety of geometric shapes. Among other operations, the library provides conversions between bounded $\mathcal{H}$ and $\mathcal{V}$ polyhedra. The purpose of this paper is to extend the conversion to unbounded $\mathcal{H}$ and $\mathcal{V}$ polyhedra through the adaption of a pivoting algorithm: the Avis-Fukuda algorithm.

1 Introduction

Computer scientists are used to deal with discrete systems, i.e. systems whose evolution can be described by a sequence of discrete state changes. A widespread example is the program analysis: during the execution of a program, the system is abstracted into a machine which states changes every time an instruction is executed. Today, computer science is expected to be able to ensure the safety of very complex systems, such as planes or nuclear reactors. The states of these systems are described not only by a discrete one (flying, launching, exploding...), but also by physical variables which evolve continuously over the time. Such systems, with continuous and discrete components are called hybrid systems. These systems are omnipresent, whenever a digital controller interacts with continuous quantities there is a hybrid system, even for things as simple as a thermostat with the temperature.

To study these systems, they are abstracted into hybrid automata [**colas**]. These automata are then model checked. Example **??** describes a thermostat and the associated automaton.

Example 1. A thermostat controls a heater in a room. Let T be the temperature in the room, the thermostat is programmed to keep the temperature between 17°C and 23°C. The heater can be in two states:

– ON: the temperature in the room increases according a linear differential equation, if the temperature is above 22°C and before it reaches 23°C, the heater is turned off;
– OFF: the temperature in the room decreases according a linear differential equation, if the temperature is below 18°C and before it reaches 17°C, the heater is turned on.

The thermostat can be described with the state of the heater, discrete, and the temperature of the room, continuous, it is a hybrid system.

The associated hybrid automata has the same discrete states as the system: ON and OFF. There will be only one continuous variable: the temperature. The evolution of the temperature is controlled by the current discrete state, being able to change or to stay in a state is controlled by the temperature. Figure **??** shows the evolution of the temperature for a system initialized with a heater on and 20°C and the resulting hybrid automata.

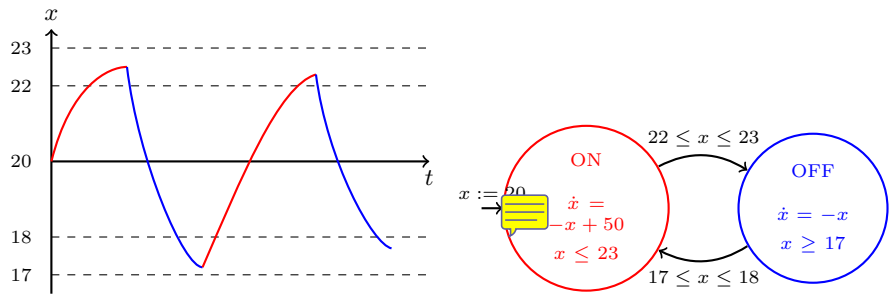

Fig. 1. On the left: the evolution of the temperature for the given thermostat initialized on the state (ON,20°C). On the right: the corresponding hybrid automata. Illustrations taken from Erika Ábráham's presentation in Genoa, Octorber 2015.

A hybrid automata is described by a set of possible discrete states and a set of continuous variables. The evolution of the variables is determined by the current discrete state. The current state of the system is defined by the discrete state it is in and by the current value of the continuous variables. A transition between two discrete states can be done under certain conditions (called a guard) over the variables, a well as staying in a given state (called the invariant of the state). A transition can reassign the variables. All these parts can be seen in Figure **??**.

Analysis of such systems is very difficult. Table 1 gives the complexity of model checking for different classes of automata. Several methods exist to deal with this problem, theorem proving (model it for a theorem prover), interval based methods (model it for an SMT solver) and flowpipe computation. This paper contributes to the flowpipe-construction-based reachability analysis methods.

As the model checking becomes rapidly non-decidable, the idea of the flowpipe computation is to determine an over approximation of a pipe containing

Sub-classes	derivative	conditions	bounded reachability	unbounded reachability
TA	$\dot{x} = 1$	$x \stackrel{?}{=} c$	✓	✓
IRA	$\dot{x} \in [c_1; c_2]$	$x \stackrel{?}{\in} [c_1; c_2]$ jump must reset x when $\dot{x}$ changes	✓	✓
RA	$\dot{x} \in [c_1; c_2]$	$x \stackrel{?}{\in} [c_1; c_2]$	✓	X
LHA I	$\dot{x} = c$	$x \stackrel{?}{=} g_{linear}$	✓	X
LHA II	$\dot{x} = f_{linear}$	$x \stackrel{?}{=} g_{linear}$	X	X
HA	$\dot{x} = f$	$\dot{x} = g$	X	X

Table 1. Decidability results for subclasses of automata. c, c_1, c_2 are constants, f and g are function of other variables. TA = timed automata, IRA = initialised rectangular automata, RA = rectangular automata, LHA I = hybrid automata with constant derivatives, LHA II = hybrid automata with linear ODE's, HA = general hybrid automata.

the continuous variables at any instant. To do so, ~~the~~ time is discretized and, at each time step, all the possible values of the continuous variables are determined (analytically if the system is linear, with approximation otherwise). The successive sets containing the potential values of the variables are then linked together with bigger sets that includes all the possible values between the two instants. The result is a connected set containing all the possible values for the variables at any time. Figure **??** illustrates the construction of the "linking" phase of the flowpipe construction and an example of flowpipe fully constructed. The construction of the flowpipe stops at a given time limit or when a fix point is reached.

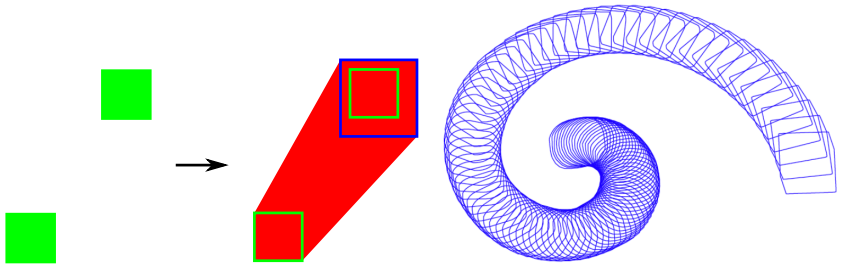

Fig. 2. On the left: an example of the construction of a containing set for all the possible values between two sets obtained by analytic solving (in green). First bloat one of the sets(in blue), then take the convex hull of the resulting sets (in red). On the right: an example of flowpipe.

There exists different sub-methods for the flowpipe construction, each corresponding to different methods to describe the sets of the possible values of the variables. Zonotopes (left part of Figure **??**), support functions and boxes are examples of representations. Several operations have to be performed on these sets along the way, for instance intersections to detect a collision with a forbidden state. How these set are described in the machine has a heavy influence on the cost to do one or another operation. The internship is about a method to switch between two representations of polyhedra: the $\mathcal{V}$ representation and the $\mathcal{H}$ representation according to Theorem 1. First some notations and definitions for the rest of the paper:

Definition 1 (Some geometry). *The problem is studied in* $\mathbb{R}^d$.

- $conv(V)$ *defines the convex hull of the set of vertices* V*:* $\{x \in \mathbb{R}^d | x = \sum_{v \in V} \lambda_v v, \sum_{v \in V} \lambda_v = 1, \forall v \in V,\ 0 \leq \lambda_v \in \mathbb{R}\}$.
- $cone(C)$ *defines the conic hull of the vectors in* C*:* $\{x \in \mathbb{R}^d | x = \sum_{c \in C} \lambda_c c, \forall c \in C,\ 0 \leq \lambda_c \in \mathbb{R}\}$.
- $lineal(L)$ *is the linear space generated by* L*:* $\{x \in \mathbb{R}^d | x = \sum_{l \in L} \lambda_l l, \forall l \in L,\ \lambda_l \in \mathbb{R}\}$.
- *In the paper, a sum between two sets is the Minkowsky sum:* $S_1 + S_2 = \{s_1 + s_2 | s_1 \in S_1,\ s_2 \in S_2\}$.
- *A half-space* h *is an area of* $\mathbb{R}^d$ *defined by a normal vector* a *and a constant* b, $h = \{x \in \mathbb{R}^d | x.a \leq b\}$, *its border is the hyperplane* $\{x \in \mathbb{R}^d | x.a = b\}$. *A family of half-spaces are said independent if their normal vectors are independent. Independent hyperplanes are defined respectively.* d *independent hyperplanes intersect in a vertex.*
- *Let* A *be a matrix which rows are normal vectors of a finite set of half-spaces and* B *the column vector composed by the corresponding constants.* $P(A, B) = \{x \in \mathbb{R}^d | Ax \leq B\}$ *is the set of the points contained in all the half-spaces.*
- *An* $\mathcal{H}$*-polyhedron* P *is the intersection of a given finite set of half-spaces. There exist* A *and* B *such that* $P = P(A, B)$.
- *A* $\mathcal{V}$*-polyhedron* P *is the Minkowsky sum (from now on refereed as sum) of the convex hull of a finite set of points and a conic hull of a finite set of points,* $P = conv(V) + cone(C)$ *for some finite* V *and* C.
- *A polyhedron is an* $\mathcal{H}$*-polyhedron or a* $\mathcal{V}$*-polyhedron.*
- *A polytope is a bounded polyhedron (and respectively with* $\mathcal{H}$ *and* $\mathcal{V}$*-polytopes).*

Theorem 1 (Polyhedra representation). *A Subset* $P \subseteq \mathbb{R}^d$ *is an* $\mathcal{H}$*-polyhedron if and only if it is a* $\mathcal{V}$*-polyhedron [***ziegler˙polytopes***].*

These two representations have their own advantages and disadvantages, which are summed up in Table **??**, being able to switch between these two representations is thus crucial.

The hosting research group develops, in the context of the HyPro project, an open-source stand-alone C++ library (the tool used to draw the flowpipe in Figure **??**) for the most relevant geometric state set representations like covering

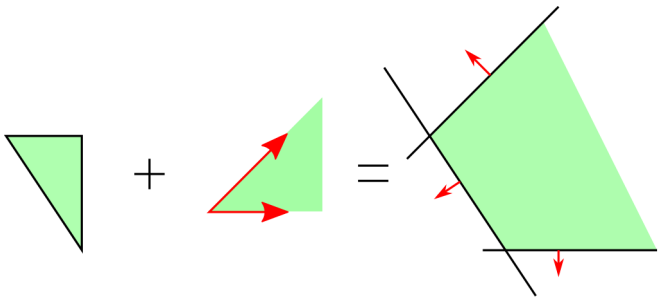

Fig. 3. Illustration of Theorem 1. From left to right: the sum of the convex hull of a set of vertices and a cone equals an intersection of half-spaces.

	$. \cap .$	$. \cup .$	$. + .$
$\mathcal{V}$-Polyhedra	difficult	easy	easy
$\mathcal{H}$-Polyhedra	easy	difficult	difficult

Table 2. Comparison of the cost of different operations between the two representation of the polyhedra.

boxes, polytopes, zonotopes, support functions. The library allows the combination of different representations and over-approximative conversion between them. So far it includes conversion between bounded $\mathcal{V}$ and $\mathcal{H}$-Polyhedra, the objective of the internship is to expand the library with the conversion between unbounded polyhedra.

The method developed in this paper and the library is an extension of a method for to convert $\mathcal{H}$-polytopes into $\mathcal{V}$-polytopes. This method was not ~~the one~~ implemented in HyPro, its implementation is a part of the contribution.

Section **??** is a state of the art, presenting a method containing the main idea used in this paper, the simplex algorithm, and an algorithm used for the conversion of bounded $\mathcal{H}$-polyhedra into $\mathcal{V}$-polyhedra: Fukuda's algorithm. Section **??** is the contribution: how to use Fukuda's algorithm on unbounded $\mathcal{H}$-polyhedra and how to reduce the conversion of unbounded $\mathcal{V}$-polyhedra into $\mathcal{H}$-polyhedra into the previously resolved problem. This reports ends by a summing up conclusion in Section **??**.

2 State of the art

Dantzig's simplex algorithm [**simplex**] (1947) is one of the oldest algorithm to solve linear optimization problems. Its geometric interpretation allowed the creation of a family of algorithms for the vertex enumeration problem called ~~the~~ pivoting algorithms. This section presents the simplex and one of the pivoting algorithms: the Avis-Fukuda algorithm [**fukuda**] (refereed as Fukuda's algorithm).

2.1 The simplex algorithm

Linear optimization. The simplex algorithm is an algorithm that solves *linear optimization problems*. A linear optimization problem is finding the positive optima a linear function (the objective) $(x_i)_{i=1}^n \mapsto \sum_{i=1}^n c_i x_i$ under a set of inequality constraints $\sum_{i=1}^n a_i x_i \leq b_i$. Geometrically, such a problem is to find the point which coordinates maximizes the objective, which can be seen as a direction, in an area defined by a set of half-spaces (each constraint is the equation of a half-spaces).

To simplify the problem, the origin is assumed to be contained in the intersection of all the half-spaces (equivalently: all the b_i are positive). When a point is on a constraint (its coordinates turn the equality into an equality), the constraint is said *saturated*. The set of the points respecting all the constraints is called the feasible area.

The first step of the algorithm is to turn all the inequalities into equalities by the addition of positive *slack* variables. These variables corresponds to a distance from the associated constraint. Note that every variable (slack or not) corresponds to a constraint and thus to a half-space. Assigning a variable to a negative value means violating the corresponding constraint. In the following, the non-slack variables will be refereed as *original*.

The dictionary. By adding to these equalities an other equality corresponding to the cost function, a tableau of coefficients is obtained. This tableau is called a dictionary. So far, every row corresponds to a slack variable plus one for the objective function and every column corresponds to an initial variable, plus one for the constants. In the rest of this report, the coefficient in the row i and column j is named a_{ij}.

The dictionary allows to find the value of every variable knowing the value of those associated to the columns. The set of the variables associated to the columns is called the *cobasis*, the set of the variables associated to the row is called the *basis*. Setting all the variables of the cobasis to zero provides a valuation for all the original variables. Such valuation describes a point against d hyperplanes: a vertex of the hyperplane arrangement.In the following, the basis is called B, the cobasis N, the row corresponding to the objective function is f ($\in B$) and the column containing the constants is g ($\in N$).

The pivot. The idea of the simplex is to exchange the variables between the basis and the cobasis (and updating in consequence the coefficients of the dictionary) such that when the cobasis is set to zero, the objective function increases and no basic variable is set to a negative value. This operation is called a pivot. From a geometrical point of view this operations consist in keeping $d-1$ constraints saturated and exchanging the remaining one for an other.The vertices of the feasible area are explored until a maximum is reached. For a pivot occuring between the variables r and s, a_{rs} is refereed as the coefficient corresponding to the pivot. If it equals zero, the pivot is impossible and it means that the variable of the basis is independent from the variable from the cobasis. The most

spread rule to select the pivot of a dictionary it Bland's rule, it ensures that the next dictionary describes a point in the feasible area and the termination of the algorithm.

Some properties. A basic variable is primal feasible if the associated constant is non negative, a cobasic variable is said dual feasible is the associated coefficient in the objective function is non positive. A dictionary is primal (resp. dual) feasible if all the variables is the basis (resp. cobasis) are primal (resp. dual) feasible. A dictionary is optimal if it is both primal and dual feasible. A dictionary is primal feasible if and only if it describes a point inside the feasible area. A dictionary is dual feasible if and only if there is no pivot able to increase the objective.

Proposition 1. *The normal vectors of the hyperplanes associated to the variables of the cobasis describe an independent family: the cobasis is, in fact, a basis.*

This proposition is obtained by induction. At the beginning of the algorithm the cobasis is the canonic basis. The only operation applied to the dictionaries (the pivoting) maintain this property. If one tries to break this invariant, the new potential member of the cobasis is a linear combination of the others. This means the value of the corresponding variable is determined by the other member of cobasis which implies the pivoting provoked a division by zero.

Alternative forms. So far, the origin is assumed to be in the feasible area and the variables were positive. There exist alternative forms of the simplex to handle these t cases.

If the origin is not in the feasible area, the simplex functions in two phases: joining the feasible area, then optimizing. Joining the feasible area is done by an other simplex instance:

- the constraints are the complementary half-space of each violated constraints alongside with the non violated one;
- the function to maximize is the opposite of the sum of the slack variables corresponding to the violated constraints.

The simplex then continues with the initial constraints and objective function, but the current basis and cobasis.

If the variables are not all positive, the idea of the simplex remains: pushing the cobasis to its bound to obtain a vertex. The sole difference is that the bounds are not 0 anymore. Since the original variables might not be bounded, or that there might be both lower and upper bounds, a valuation for all the variables is kept (initialized at 0 for all). The valuation is modified when a pivot occurs: the the new cobasic variable is push to its bound that maximizes the objective.

An example of the simplex algorithm. Here is given an example of use of the simplex algorithm on the linear optimization problem: maximize $x+y$ under $y \leq 2$ and $x - y \leq 2$.

Creation of the dictionary: the slack variables are defined $0 \leq s_1 = 2 - y$ and $0 \leq s_2 = 2 - x + y$. These equalities are then put together in the dictionary:

x ↓	y ↓	constants ↓	
0	-1	2	$= s_1$
-1	1	2	$= s_2$
1	1	0	← objective function

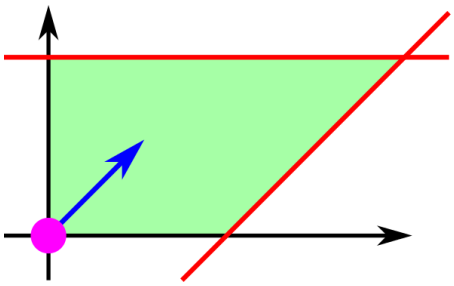

Fig. 4. Geometrical interpretation of the maximization problem. The constraints are in red, the blue arrow corresponds to the objective function, the feasible area is shaded, the purple point is the origin (the point described by the first dictionary).

Bland's rule tells to pick x and s_2 for the first pivot, which leads to

s_2	y	const	
0	-1	2	$= s_1$
-1	1	2	$= x$
-1	2	2	← obj

.

Then y and s_1:

x	s_1	const	
0	-1	2	$= s_1$
-1	-1	4	$= s_2$
-1	-2	6	← obj

.

There is no other pivot selected by Bland's rule, note that the final dictionary is optimal.

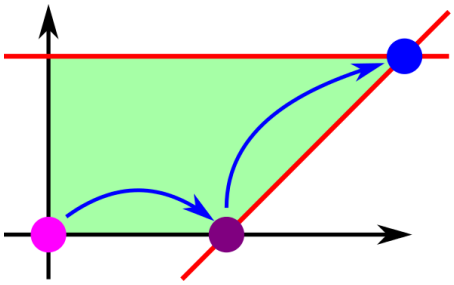

Fig. 5. The current point after the first pivot is $(2, 0)$, and after the second $(4, 2)$.

2.2 Fukuda's algorihm

Fukuda's algorithm allows to enumerate all the vertices of an $\mathcal{H}$-polytope included in the positive orthant ($(\mathbb{R}_+)^d$, all its points have non-negative coordinates) for which the origin is a vertex. This algorithm uses the pivoting scheme of the simplex to explore the polytope.

It is based on a simple idea: starting from any vertex, the simplex algorithm allows to reach a vertex maximizing a linear function by a unique (thanks to Bland's rule) path among the vertices. The algorithm picks a function maximized only by the origin on the positive orthant and walks backward on all the paths leading to it from all the vertices. Every time a vertex is encountered, it is output.

The first thing to do is to find the end of all these paths: the dictionary corresponding to the origin. It ~~roug~~ consists in taking the canonical base as cobasis and the set constraints as the basis and setting $-\sum_{i=0}^{d} x_i$ as the cost function. All the possible pivots are tried from this dictionary. If a pivot leads to a primal feasible dictionary (defining a vertex inside the polytope) for which the Bland's method leads back to the previous point it is said to be a valid reverse

Bland's pivot. It means that a step backward has been made on one of the paths, the new point is output, and the method has to be applied from this new point.

However, if a vertex is defined by more hyperplanes than required, several dictionaries will describe the same point, which is said to be degenerated. Fukuda overcomes this problem by defining a partial order on the dictionaries. This order corresponds to a lexicographic order on the basis for the dictionaries defining the same point. A point is only output if its dictionary is lexicographical minimal. Note that the algorithm continues on the dictionary, even if it is not lexicographical minimal.

If the origin is degenerated, then all the optimal dictionaries defining it have to be found. This is done by using a procedure very similar from the one above: by looking only at the hyperplanes redefining the origin, every pivot is tried. The difference is instead of using the Bland's rule, a variation of the dual Bland's rule is used. If the new dictionary is dual feasible (still an optimal for the cost function) and the dual Bland's rule brings it back to the previous one, then the new dictionary is valid with respect to the dual Bland's rule etc. This research presents the same uniqueness properties than the previous one. The previous research has to be launched from every dictionary obtained this way.

Figure **??** shows the paths to reach the origin for the vertices of the unit cube (constraints: $0 \leq x, y, z \leq 1$) and the tree of the exploration made by the algorithm.

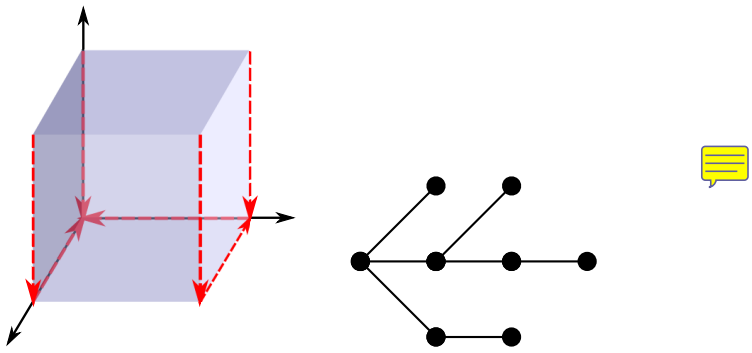

Fig. 6. On the left: the paths to reach the origin for the vertices of the unit cube (constraints: $0 \leq x, y, z \leq 1$). On the right: the tree of the vertices given by Fukuda's algorithm.

Complexity: for n half-space in a space of dimension d, Fukuda gives a complexity of $O(nd(n+d)g)$, where g is the number of vertices of the polyhedron, counted with their multiplicity (a degenerated vertex can be define in several ways, it is counted several times).

3 Contribution

The purpose of this internship was to add two methods in HyPro: one to find the vertices, the cone and the linealty space of an $\mathcal{H}$-polyhedron and one to find the convex hull of a $\mathcal{V}$-polyhedron.
Fukuda's algorithm allows to enumerate the vertices of an $\mathcal{H}$-polytope. The first thing to do was extending it into being able to handle non positive polytopes and then polyhedra. This method implemented, an other one had to be created to convert the convex hull problem such that he former method could handle the other problem.

3.1 Adaptation of Fukuda's algorithm for polyhedra

Is this section, Fukuda's algorithm is adapted for the vertex enumeration of unbounded polyhedra. For a given $\mathcal{H}$-polyhedron, described by half-spaces, the objective is to find a set of vertices $\mathcal{V}$ and a set of vectors C such that $P = conv(V) + cone(C)$. Note that if a line directed by a vector l is included in P, $\{l, -l\} \subset C$.

Converting to positive coordinates and detecting the linealy space Knowing a vertex of a polyhedron allows to find an affine transformation which sends it in the positive orthant. The process is to map the vertex to the origin and the normals of the hyperplanes defining it to the canonical base by respectively a translation and a change of basis.

Finding a vertex of the polyhedron is done using a dictionary as presented in Section 2.1. In fact, obtaining a cobasis composed only by slack variables pushed to their bounds with all the other variables in their bounds gives a dictionary that describes a vertex of the polyhedron. Here the row corresponding to the cost function and the column corresponding to the constants are not used.

To do so, the first thing is to create a dictionary and to reach the feasible area, to check for emptiness. This is exactly the first phase of the simplex algorithm. Then, since the cobasis has to be composed only by slack variables, all the original variables in the cobasis are pivoted with a suitable slack variable (the corresponding coefficient has to be non null) but the assignments are not changed: it could bring the dictionary out of the polyhedron. $\mathcal{H}$

Detection of the linealty space: The linealty space is the biggest affine subspace included in the polyhedron. If an original variable has no suitable slack variable to be exchanged with, it means this variable has no influence on the distance to each constraints, there is a line included in the polyhedron: an affine subspace has been found. To overcome this problem, two constraints are added with opposed normal vectors, their directions being the affine subspace's one, and the constant is zero for both. This provides a suitable slack variable to pivot around and the linealty space is saved, Example 8 illustrates this procedure. A cobasis full of slack variables means there exists enough independent constraints to define a vertex,

and ~~there is~~ no linealty space exists. Since this phase adds new constraints, the feasible area might have to be reached again.

Example 2. Let $-1 \leq x+y \leq 1$ be two constraints, the corresponding dictionary is:

x	y	
-1	-1	$= s_1$
-1	-1	$= s_2$

. x can be pivoted with s_1:

s_1	y	
-1	-1	$= x$
1	0	$= s_2$

. In this dictionary, y and s_2 are independents, a pivot can't occur. It means that for all values of y there is a suitable x in the feasible area: there is and affine space directed by $(-1, 1)$ included in the polyhedron. the constraints $-x + y = 0$ are added.

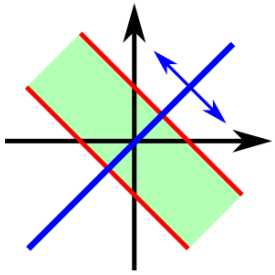

Fig. 7. Geometrical interpretation of the constraints, with the feasible area shaded. The constraints represented with their normal vectors are added after detection of the linealty space

The last thing is pushing one by one the variables of the cobasis to their bounds. If doing so would make other variables violate their bounds, the variable is exchanged with the one which bound would be violated first, the latter is then set to its bound (note that the original variables have no bounds). A vertex is found with the hyperplanes that define it.

Complexity: required to apply Fukuda's algorithm, this phase costs the finding of the feasible area, which is equivalent to a linear optimization, then pushing every variable to their bounds (which might imply pivoting for every variable). The final cost is $O(max(n, d)d^2 + L(d, n))$ for a problem with n half-spaces, d dimensions and $L(d, n)$ the complexity of a linear optimization in d dimensions and n constraints. This can be neglected in front of the cost of Fukuda's Algorithm.

Cone detection in Fukuda's algorithm Here the problem is reduced to the vertex enumeration of a potentially unbounded polyhedron in the positive orthant. Fukuda's algorithm allows to explore all the vertices. The unboundedness in general has no influence on the behaviour of the algorithm: it corresponds to a lack of constraints, while the algorithm just switches between the constrains.

The research of the cone relies on the following proposition:

Proposition 2. *Let P be a polyhedron. If d hyperplanes intersect in a vertex v, and $d-1$ of them intersect in a line directed by a vector l, if $\forall k \in \mathbb{R}_+$, $v + kl \in P$, then l belongs to the cone of P. The cone of P is generated by the vectors obtained this way.*

The belonging to the cone comes from it definition: the cone is the set of all the unbounded directions of the polyhedron. Let V be the set P's vertices and L

the set of vectors given by the proposition, the rest of the property is obtained by showing $P = conv(V) + cone(L)$. $conv(V) + cone(L) \subset P$ is already done, the other inclusion is obtained by, for any point of P, projecting with the vectors of L until a bounded face is reached (all the point of a bounded face belongs to the convex hull of the vertices of the face).

Proposition 2 states that to find all the vectors of the cone, it is sufficient to check every vertex for the existence of a conic direction, i.e. directions such that $d-1$ independent constraints are saturated. Fukuda's algorithm explores the polyhedron and finds all the possible arrangement of constraints that define the vertices, thus for every definition of a vertex, the dictionary is checked for the existence of a cobasic variable that can be increased while not decreasing any other (not getting closer from other constraint) and which decreases the objective function (not getting closer from an axis). Conversely, a vector found this way is a ray included in the polyhedron: it belong to the cone. The rays can be output several times, a suppression of the duplicates is done. Example 9 illustrates the detection of a ray for a given dictionary.

Example 3. Cone detection with the dictionary:

x ↓	y ↓	const ↓	
-1	1	2	$= s_1$
-1	-1	0	← obj

. This dictionary corresponds to the constraints $x - y \leq 2$. As s_1 would be decreasing, x does not provides a cone direction, yet y does: the objective is decreased and s_1 increases: $(0, 1)$ belongs to the cone.

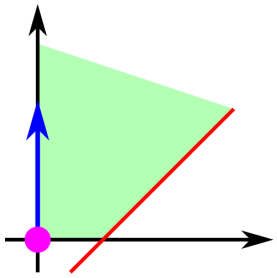

Fig. 8. Geometrical interpretation, the arrow represents the conic direction found.

Complexity: for each vertex, the existence of a conic direction is to be tested. This adds $O(gnd)$ to the cost of Fukuda's algorithm, which cost remains the same.

The algorithm enumerates the linealty space when looking for a first vertex, and the cone and the vertices during Fukuda's algorithm: it allows to switch from an $\mathcal{H}$-polyhedron to a $\mathcal{V}$-polyhedron.

3.2 From vertex enumeration to convex hull

In geometry, vertices and half-spaces are close concepts: a vertex $(a_i)_{i=0}^d$ can be associated to a half-space $\sum_{i=0}^d x_i a_i \leq 1$ et vice-versa. This association is the *geometric duality* and it allows to use the vertex enumeration algorithm to find the convex hull of a polyhedron. The emptiness and the equality to a single point of a $\mathcal{V}$-polyhedron is trivial to check, in the following, the polyhedron and

polytopes are assumed to be at least two-dimensional. Figure **??** shows a pair vertex/half-space and two triangles that are dual from each other.

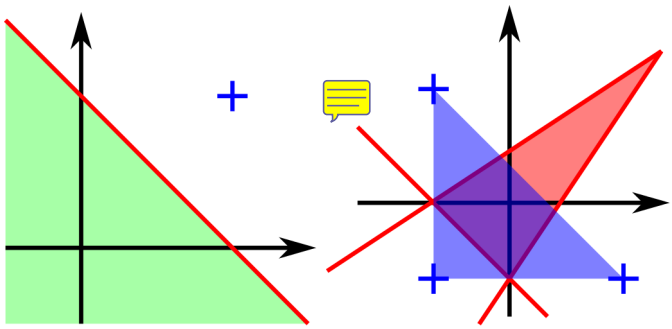

Fig. 9. Left: pair vertex/half-space that are dual from each other. Right: two triangles dual from each other.

The convex hull of a polytope The polytope studied is refereed as P, convex hull of the set V. Even it means to translate P, it is assumed that $\sum_{x \in V} x = 0$. Since the polytope is at least two-dimensional, the origin is not a vertex.

Definition 2 (The dual of a subset of $\mathbb{R}^d$). *The dual P^Δ of a subset P of $\mathbb{R}^d$ is defined by $\{c | \forall x \in P, cx \leq 1\}$*

Theorem 2 holds the idea of the transformation:

Theorem 2. *If P is a polytope and $0 \in P$, $P = (P^\Delta)^\Delta$.*

Starting with the set of the vertices of P, the algorithm begins by switching to P^Δ, which is described by a set of half-spaces. The vertices of P^Δ enumerated and the dual is taken, which leads to a set of half-spaces describing P. Indicing by the description mean, the algorithm does $P_V \to P_H^\Delta \to P_V^\Delta \to P_H^{\Delta\Delta} = P_H$.

The first step ($P_V \to P_H^\Delta$) does not hold any problem and is done with the theorem 3:

Theorem 3. *For P the convex hull of a given set V of vertices, $P^\Delta = \{c | \forall x \in V, cx \leq 1\}$.*

The second step ($P_H^\Delta \to P_V^\Delta$) is done by the vertex enumeration algorithm.

The third ($P_V^\Delta \to P_H^{\Delta\Delta}$) does not cause any problem if the origin is not on a facet of P (or a face of other dimension, seen as an intersection of facets): then P^Δ is bounded and the theorem 3 holds.

Otherwise, let $\{x | w.x \leq 0\}$ be a facet of P (since 0 is the iso-barycenter of P, $\{x | -w.x \leq 0\}$ is also a facet). Let $x \in P$, $y \in P^\Delta$ and $\alpha \in \mathbb{R}$, $(y + \alpha w).x = y.x + \alpha(w.x) = y.x$, this means that $vect(w)$ is a linealty space of P^Δ. $P^\Delta = conv(V') + lineal(L')$ thus $P^{\Delta\Delta} = conv(V')^\Delta \cap L'^\perp$ by the proposition 3. Which end the convex hull problem for a polytope.

Proposition 3. *For V a set of vertices and L a set of non null vectors $(conv(V)+lineal(L))^\Delta = conv(V')^\Delta \cap L'^\perp$.*

Proof. Let $y \in (conv(V) + lineal(L))^\Delta$, for all $v \in V$, $l \in L$ and $\alpha \in \mathbb{R}$, $y.(v + \alpha l) \leq 1$. The inequality holding for any α, $y.l = 0$ and $y.v \leq 1$ and $y \in conv(V')^\Delta \cap L'^\perp$. Conversely, let $y \in conv(V')^\Delta \cap L'^\perp$ which means $y.\alpha l = 0$ and $y.v \leq 1$ thus $y.(v + \alpha l) \leq 1$ for all $v \in V$, $l \in L$ and $\alpha \in \mathbb{R}$: $y \in (conv(V) + lineal(L))^\Delta$.

Complexity: taking the dual of a polytope costs $O(n)$ if the object is not totally rewritten, even if rewritten it costs $O(nd)$. Either way, the cost of the convex hull problem is the cost of Fukuda's algorithm.

The convex hull of a cone To find the convex hull of a cone C, the convex hull algorithm for a set of points is used. The origin is added to the set of vectors describing the cone, as for all c in C, the segment between the origin and c must belong to the cone. Then the half-spaces for which the constant is not zero (i.e. the origin does not saturate the constrain) are deleted. This deletion is required to ensure the unboundedness of the cone. Note that this method allows to handle a linealty direction as the sum of two vectors in the cone.

The convex hull of a polyhedron Here the linealty space is injected in the cone (as the sum of two opposite vectors), as the treatment does not differ. Let $P = conv(V) + cone(C)$ be a polyhedron, the method is given by the following result:

Proposition 4. *Let $P = conv(V)+cone(C)$ be a polyhedron, C' the set obtained from C by adding a 0 as a 0^{th} coordinate to every vectors $(c \mapsto (0, c))$ and V' the set obtained by, by adding a 1 as a 0^{th} coordinate to every vectors $(v \mapsto (1, v))$, then $P = cone(V', C') \cap \{x | x = (1, y), y \in \mathbb{R}^d\}$*

Starting from P, C' and V' are easily obtained and $cone(V', C')$ is given in 3.2. The intersection is done as follows:

$$\{x \in \mathbb{R}^{d+1} | a_0 x_0 + \sum_{i=1}^{d} a_i x_i \leq b\} \cap \{x | x_0 = 1\} = \{x \in \mathbb{R}^d | \sum_{i=1}^{d} a_i x_i \leq b - a_0\}$$

This operation can generate half-spaces with an equation such as $0 \leq b$, such half-spaces are those which does not intersect with $\{x | x_0 = 1\}$ and they are eliminated.

Complexity: the cost of the convex hull in the unbounded case is the cost of Fukuda's algorithm with one more point and a dimension bigger by one.

4 Conclusion

The conversion between $\mathcal{V}$ and $\mathcal{H}$ polyhedra is crucial for the flowpipe based analysis using polyhedra, each representation having its own advantages/disadvantages.

The library developed by the hosting group included two different algorithms for the conversion in the bounded case. The internship the report follows contributed to the library by implementing Fukuda's algorithm for the conversion $\mathcal{H}$ to $\mathcal{V}$ in the bounded case, and by implementing the following method for the unbounded problem.

Unbounded $\mathcal{H}$ to $\mathcal{V}$: the first thing is to find a vertex of the polyhedron to be able to express it with positive coordinates (for Fukuda's algorithm). Doing so, any affine subspace included in the polyhedron is found. Then Fukuda's algorithm is applied and every vertex is searched for vectors of the cone. The whole cone and the vertices of the polyhedron are found.

Unbounded $\mathcal{V}$ to $\mathcal{H}$: the polyhedron is transformed into a cone of greater dimension. Thanks to properties of the geometrical dual, the vertices of the dual of this cone are found with the previous conversion ($\mathcal{H}$ to $\mathcal{V}$). Taking the dual gives the convex hull of the cone, the algorithm ends by extracting the polyhedron from the cone.

The theoretical complexity of both conversion equals the complexity of Fukuda's algorithm.