Abstract acceleration in Linear relation analysis

Laure Gonnord, Nicolas Halbwachs

Unité Mixte de Recherche 5104 CNRS - INPG - UJF
Centre Equation
2, avenue de VIGNATE
F-38610 GIERES
tel : +33 456 52 03 40
fax : +33 456 52 03 50
http://www-verimag.imag.fr

Abstract Acceleration in Linear Relation Analysis

Laure Gonnord, Nicolas Halbwachs

Verimag Research Report n^o TR-2010-10

March 2010

Abstract Acceleration in Linear Relation Analysis

Laure Gonnord, Nicolas Halbwachs

March 2010

Abstract

Linear Relation Analysis [28, 39] is now a classical abstract interpretation based on an approximation of reachable numerical states of a program by convex polyhedra. Since it works with a lattice of infinite depth, it makes use of a widening operator to enforce the convergence of fixpoint computations. This paper takes place in the many attempts to improve the precision of the results reached using such a widening. It will first present an extended survey of the existing approaches in that direction. Then it will investigate the cases where the exact (abstract) effect of a loop can be computed. This technique is fully compatible with the use of widening, and whenever it applies, it generally improves both the precision and the performances of the analysis.

Keywords: Program analysis, linear relation analysis, polyhedra, widening, acceleration.

Reviewers: reviewer1, reviewer2

Notes: This work has been partially supported by the APRON project of the "ACI Sécurité Informatique" of the French Ministry of Research. Laure Gonnord is Assistant Professor at Lille University. Nicolas Halbwachs is CNRS Researcher at Verimag.

How to cite this report:

```
@techreport { ,
title = { Abstract Acceleration in Linear Relation Analysis},
author = { Laure Gonnord, Nicolas Halbwachs},
institution = {  Verimag Research Report },
number = {TR-2010-10},
year = { 2010},
note = { }
}
```

1 Introduction

Linear Relation Analysis [28, 39] (LRA) is one of the very first applications of abstract interpretation [27]. It aims at computing an upper approximation of the reachable states of a numerical program, as a convex polyhedron (or a set of such polyhedra). It was applied in various domains like compile-time error detection [29], program parallelization [43], automatic verification [41, 42] and formal proof [10, 11].

Like any approximate verification method, LRA is faced with the compromise between precision and cost. Since its relatively high cost restricts its applicability, any situation where the precision can be improved at low cost must be exploited. One source of approximation in LRA is widening, an operation that ensures the termination of iterative computations, by extrapolating an upper approximation of their limits.

Improving the precision of the result of widened iterations has motivated so many works that we will devote a whole section to their survey. Some authors propose better widening operators, while others consider the way the widening is applied. The first track raises the question of "what is a better widening?". The fact that one single application of a widening operator gives smaller results does not necessarily mean that its repeated application will involve a convergence towards a more precise limit. Moreover, such "more precise" widenings, and the use of many proposed application policies — like delaying the widening — are likely to slow down the convergence, by increasing the number of necessary iterations.

These remarks led us to look at situations where the widening can obviously be improved — in the sense that it involves a faster convergence towards a better limit — at low cost with respect to the cost of usual polyhedra operators. A source of inspiration are the so-called "acceleration techniques" proposed by several authors [14, 64, 23, 32, 7]. These works consist in identifying categories of loops whose effect can be computed exactly. Roughly speaking, the effect of a simple loop, guarded by a linear condition on integer variables, and consisting of incrementations/decrementations of these variables can be computed exactly as a Presburger formula. These methods have the advantage of giving exact results. Now, because they are exact, they are restricted to some classes of programs (e.g., "flat counter automata", *i.e.,* without nested loops). Moreover, the exact computation with integer variables has a very high complexity (2^{EXP}). So the applicability of these methods is somewhat limited.

In this paper, we investigate on the use of acceleration methods in LRA, *in complement to widening*. Of course, when the effect of a loop can be computed exactly (and at low cost) there is no need to approximate it. Now, since we want to integrate these results in LRA, only the exact *abstract* effect of the loop is necessary, that is the *convex hull* of the reachable states during or after the loop. This means that we won't use expensive computations in Presburger arithmetic. Moreover, we only look for an improvement of standard LRA: wherever an acceleration is possible, its application will improve the results, but the resulting method is not restricted to those programs where acceleration applies everywhere.

The paper is organized as follows: After a reminder of the principles of LRA (Section 2), Section 3 and Section 4 surveys the existing works about widening and acceleration techniques. Our proposal is introduced by a motivating example in Section 5. Then, Section 6 addresses the trivial case of a single loop where variables are just incremented with constants, and defines the notion of *abstract acceleration*. In Section 7, we consider the case of several translation and reset loops. We are then able to describe the mechanisms that we implemented in our tool Aspic (Section 8). In Section 9, some experimental results are described, before giving some conclusions.

2 Linear Relation Analysis

In this section, we briefly recall the principles of Linear Relation Analysis (LRA), and we introduce some notations used throughout the paper.

The goal of Linear Relation Analysis is to attach to each control point of a program a system of linear inequalities satisfied by the numerical variables whenever the control is at that point. This is done by propagating systems of linear inequalities along the control paths of the program.

2.1 Convex polyhedra

Let n denote the number of numerical variables. A state of the numerical variables will be a point $\vec{x}$ in the affine space $\mathbb{Q}^n$. The set of solutions of a system of linear inequalities — say, $A\vec{x} \leqslant B$, where A is an $m \times n$ matrix, for some $m \geqslant 0$, and B is an m-vector — is a (closed convex) polyhedron in $\mathbb{Q}^n$. A point (resp., a ray) of the polyhedron is a point $\vec{x}$ (resp., a vector $\vec{r}$) such that $A\vec{x} \leqslant B$ (resp., $A\vec{r} \leqslant 0$). A polyhedron P can also be characterized by a *system of generators*, which is made of a finite set V of points (called *vertices*) and a finite set R of rays, such that each point $\vec{x}$ of P can be expressed as the sum of a convex combination of elements of V and a positive combination of elements of R:

$$\left\{ \sum_{\vec{v}_i \in V} \lambda_i \vec{v}_i + \sum_{\vec{r}_j \in R} \mu_j \vec{r}_j \mid \lambda_i, \mu_j \in \mathbb{Q}^+, \sum \lambda_i = 1 \right\}$$

If $A\vec{x} \leqslant B$ is a system of constraints, we will often note simply $\{A\vec{x} \leqslant B\}$ the polyhedron of its solutions. Similarly, [V,R] will denote the polyhedron generated by the system of generators (V, R). If P is a polyhedron and $R \subset \mathbb{Q}^n$ is a finite set of vectors, we will note $P \nearrow R$ the polyhedron $\{\vec{x} + \sum_{\vec{r}_j \in R} \mu_j \vec{r}_j \mid \vec{x} \in P, \mu_j \in \mathbb{Q}^+\}$ obtained by adding to P all the vectors of R as new rays.

This double representation of polyhedra, by means of systems of constraints and systems of generators, is important, since some operations are easier on one or the other representation. Moreover, the knowledge of both representations allows them to be minimized (*i.e.,* removing redundant constraints, and non extremal vertices and rays). Algorithms have been proposed [22, 47, 63] for translating each representation into the other.

The lattice of polyhedra: The set of polyhedra in $\mathbb{Q}^n$ is a lattice, with least element $\bot$ (the empty polyhedron), greatest element $\top$ (the universe polyhedron $\mathbb{Q}^n$); the greatest lower bound operator $\sqcap$ is the intersection, but the least upper bound operator $\sqcup$ is not the union (which is generally not convex), but the *convex hull*, *i.e.,* the least polyhedron containing the operands.

Saturations, redundancies: Let us say that a vertex $\vec{v}$ (resp. a ray $\vec{r}$) *saturates* a constraint $\kappa = a\vec{x} \leqslant b$ if $a\vec{v} = b$ (resp. $a\vec{r} = 0$). Let $Sat(\kappa)$ denote the set of vertices and rays saturating the constraint κ. A constraint κ of a polyhedron P is an *equation* if it is saturated by all the generators of P. A constraint κ is *redundant* in a system of constraints, if there is another constraint κ' in the system such that $Sat(\kappa) \subset Sat(\kappa')$ and κ' is not an equation. κ and κ' are *mutually redundant* if $Sat(\kappa) = Sat(\kappa')$.

2.2 Operations on polyhedra

We define the essential operations used in LRA:

Intersection: A system of constraints of the intersection of two polyhedra is simply the conjunction of their systems of constraints:

$$\{A\vec{x} \leqslant B\} \sqcap \{A'\vec{x} \leqslant B'\} \;=\; \left\{ \left[\begin{array}{c} A \\ A' \end{array} \right] \vec{x} \leqslant \left[\begin{array}{c} B \\ B' \end{array} \right] \right\}$$

Convex hull: A system of generators of the convex hull $P \sqcup Q$ is the union of those of P and Q:

$$[V, R] \sqcup [V', R'] \;=\; [V \cup V', R \cup R']$$

Affine transformation: The image of a polyhedron P by an affine transformation $\vec{x} \mapsto C\vec{x} + D$, where C is an $n \times n$ matrix and $D \in \mathbb{Q}^n$, is the polyhedron $[C\vec{x} + D](P) = \{\vec{y} \in \mathbb{Q}^n \mid \exists \vec{x} \in P, \vec{y} = C\vec{x} + D\}$. Then:

$$[C\vec{x} + D][V, R] \;=\; [\{C\vec{v} + D \mid \vec{v} \in V\}, \{C\vec{r} \mid \vec{r} \in R\}]$$

Variable elimination: The projection (or existential quantification) of a polyhedron according to some variable x_i can be obtained

- either by eliminating x_i from the system of constraints of P by the Fourier-Motzkin procedure;
- or by adding to P the rays $\vec{u_i}$ and $-\vec{u_i}$, where $\vec{u_i}$ is the unit vector of the i-th dimension.

The result is noted $\exists x_i.P$. It can be generalized to a set of variables.

Test for inclusion: P is included in Q if and only if all vertices of P belong to Q, and all rays of P are rays of Q:

$$[V, R] \sqsubseteq \{A\vec{x} \leqslant B\} \;\Leftrightarrow\; (\forall \vec{v} \in V, A\vec{v} \leqslant B) \;\wedge\; (\forall \vec{r} \in R, A\vec{r} \leqslant 0)$$

Test for emptiness: A polyhedron is empty if and only if it has no vertex:

$$[V, R] = \emptyset \;\Leftrightarrow\; V = \emptyset$$

2.3 Models of programs

Throughout the paper, a program will be represented by an interpreted automaton — or a control-flow graph —, $(K, k_{init}, \mathcal{T})$, where

- K is a finite set of *control points*
- $k_{init} \in K$ is the initial control point
- $\mathcal{T}$ is a finite set of transitions, each transition being a 4-tuple (k, g, a, k'), where k (resp., k') is the source (resp., target) control point, $g \in (\mathbb{Q}^n \mapsto \mathbb{B})$ is a guard (function from variable valuations to Booleans), and $a \in (\mathbb{Q}^n \mapsto \mathbb{Q}^n)$ is an action.

Concretely, guards and actions will generally be expressed by means of conditions and assignments on a set $\mathcal{V} = x_1, x_2, \ldots x_n$ of variables.

Operational semantics: The set of states of the program is $K \times \mathbb{Q}^n$. A state $(k, \vec{x})$ is reachable if and only if there exists a sequence of states $(k_0, \vec{x}_0), (k_1, \vec{x}_1), \ldots, (k_p, \vec{x}_p)$ such that:

- $k_0 = k_{init}$, $k_p = k$ and $\vec{x}_p = \vec{x}$
- for each $i = 0 \ldots (p-1)$, there exists in $\mathcal{T}$ a transition (k_i, g, a, k_{i+1}) such that $g(\vec{x_i}) = \mathit{true}$ and $\vec{x}_{i+1} = a(\vec{x_i})$.

Let $\mathcal{R}$ denote the set of reachable states.

Collecting semantics: For each control point $k \in K$, let us define $\mathcal{R}_k$ to be the set of possible variable valuations when the control is in k:

$$\mathcal{R}_k = \{\vec{x} \in \mathbb{Q}^n \mid (k, \vec{x}) \in \mathcal{R}\}$$

Then, we have, for each $k \in K$:

$$\mathcal{R}_k = \text{ if } k = k_{init} \text{ then } \mathbb{Q}^n \text{ else } \bigcup_{(k', g, a, k) \in \mathcal{T}} a(\mathcal{R}_{k'} \cap g)$$

where $\mathcal{R}_{k'} \cap g$ stands for $\{\vec{x} \in \mathcal{R}_{k'} \mid g(\vec{x}) = \mathit{true}\}$ and $a(\mathcal{R}_{k'} \cap g)$ stands for $\{a(\vec{x}) \mid \vec{x} \in \mathcal{R}_{k'} \cap g\}$.

2.4 The analysis

Abstract semantics: As usual in abstract interpretation, the abstract semantics define approximation of the sets $\mathcal{R}_k$. Let $g^\sharp$ and $a^\sharp$ denote respectively the polyhedral approximations of a guard g and an action a:

- if g is a system of linear inequalities, $g^\sharp = g$, otherwise it can be any system of linear inequalities implied by g, including $\top$, the system with 0 inequalities.
- if a is an affine assignment, say $\vec{x} := C\vec{x} + D$, then $a^\sharp = a$. Otherwise, it can be made of an affine assignment of some variables (say $\vec{y} := C\vec{x} + D$) and a loss of information about other variables (say $\vec{z} :=?$), in which case $a^\sharp$ is the composition of an affine assignment — changing $\vec{y}$ into $C\vec{x} + D$ and leaving $\vec{z}$ unchanged — and a projection according to $\vec{z}$:

$$a^\sharp : P \mapsto \exists z.[C\vec{x} + D, z](P)$$

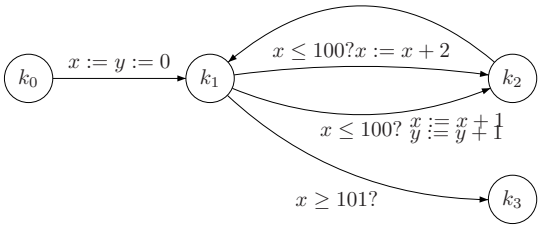


Figure 1: A simple "program"

Then, we can attach with each control point k a polyhedron P_k, which is an upper approximation of $\mathcal{R}_k$, as follows:

$$P_k = \text{ if } k = k_{init} \text{ then } \top \text{ else } \bigsqcup_{(k',g,a,k)\in\mathcal{T}} a^\sharp(P_{k'} \sqcap g^\sharp)$$

Iterative computations: We are left with a system of fixpoint equations

$$(P_k = F_k(P_1, ...P_k, ...))_{k\in K}$$

whose least solution can be computed by iteration from $P_k = \bot, \forall k$. Of course, this computation will raise the problem of termination.

Example 1: Let us consider the "program" of Fig. 1. The corresponding system of abstract equations is

$$\begin{array}{ll} P_0 & = \top \\ P_1 & = [(x,y) := (0,0)](P_0) \sqcup P_2 \\ P_2 & = [(x,y) := (x+2,y)](P_1 \sqcap \{x \leqslant 100\}) \sqcup [(x,y) := (x+1,y+1)](P_1 \sqcap \{x \leqslant 100\}) \\ P_3 & = P_1 \sqcap \{x \geqslant 101\} \end{array}$$

Starting the iterative resolution, we get:

Step	0	1	2	3	...
P_0	$\bot$	$\top$	$\top$	$\top$	...
P_1	$\bot$	$\{x = y = 0\}$	$\{0 \leqslant y \leqslant x \leqslant 2 - y\}$	$\{0 \leqslant y \leqslant x \leqslant 4 - y\}$	...
P_2	$\bot$	$\{x + y = 2, 0 \leqslant y \leqslant x\}$	$\{2 \leqslant x + y \leqslant 4, 0 \leqslant y \leqslant x\}$	$\{2 \leqslant x + y \leqslant 6, 0 \leqslant y \leqslant x\}$	...
P_3	$\bot$	$\bot$	$\bot$	$\bot$	...

Obviously, at step i we will get $P_1 = \{0 \leqslant y \leqslant x \leqslant 2i - 2 - y\}$, $P_2 = \{2 \leqslant x + y \leqslant 2i, 0 \leqslant y \leqslant x\}$, and the iterations will continue until P_1 intersects $\{x \geqslant 101\}$. Performing this extrapolation at once is the role of the widening operator.

3 Widening in LRA: state of the art

3.1 Principles

According to [27], a widening operator on polyhedra is a binary operator ∇ such that

- $\forall P, Q,\ P \sqcup Q \subseteq P \nabla Q$
- (chain condition) for any sequence $P_0, P_1, \ldots$ of polyhedra, the sequence $Q_0 = P_0$, $Q_i = Q_{i-1} \nabla P_i$ is not strictly increasing.

Let $(P_k = F_k(P_1, ...P_k, ...))_{k\in K}$ be a system of fixpoint equations. A widening operator is used to ensure the convergence of iterative computations of fixpoints as follows: Let K_∇ be a subset of K, such that each loop in the program graph contains at least one control point in K_∇. Then, for each $k \in K$, define F_k^∇ by

$$F_k^\nabla(P_1, ...P_k, ...) = \begin{cases} P_k \nabla F_k(P_1, ...P_k, ...) & \text{if } k \in K_\nabla \\ F_k(P_1, ...P_k, ...) & \text{otherwise} \end{cases}$$

Then, the iterative computation of the least solution of the system $\left(P_k = F_k^{\nabla}(P_1, ...P_k, ...)\right)_{k\in K}$ converges after a finite number of steps towards a solution $(P_k^{\nabla})_{k\in K}$ which is an upper approximation of the least solution of the initial system.

3.2 Standard widening

The very first widening operator over polyhedra was proposed in [28]: $P_1 \nabla P_2$ is defined to be the polyhedron whose system of constraints is made of the constraints of P_1 satisfied by P_2. Moreover, $\bot \nabla P = P$ for all P. Since the set of constraints of $P_1 \nabla P_2$ is included in the set of constraints of P_1, the widening cannot be applied indefinitely without convergence (chain condition).

This initial operator has the drawback that its result depends on the form of the systems of constraints. For instance, in our Example 1, if k_1 is the widening point, at step 2 we have to compute $\{x = 0, y = 0\}\nabla\{0 \leqslant y \leqslant x \leqslant 2 - y\}$, *i.e.,* $\{0 \leqslant x \leqslant 0, 0 \leqslant y \leqslant 0\}\nabla\{0 \leqslant y \leqslant x \leqslant 2 - y\}$ which gives simply $\{0 \leqslant x, 0 \leqslant y\}$. Now, if we rewrite the first system of constraints into the equivalent system $\{0 \leqslant y \leqslant x \leqslant 0\}$, the widening evaluates to $\{0 \leqslant y \leqslant x\}$, which is much more precise.

This is why [39] proposed a better operator, often referred to as *standard widening*: it consists of keeping for $P_1 \nabla P_2$ not only the constraints of P_1 satisfied by P_2, but also the constraints of P_2 mutually redundant with some constraint of P_1 in the system of constraints of P_1.

In the example before, P_1 is defined by the system of constraints $\{0 \leqslant x \leqslant 0, 0 \leqslant y \leqslant 0\}$ and the system of generators $V = \{(0, 0)\}, R = \emptyset$. In the system of constraints of P_2, $\{0 \leqslant y \leqslant x \leqslant 2 - y\}$, $y \leqslant x$ is saturated by the unique vertex of P_1, $(0, 0)$, so it is mutually redundant with any constraint of P_1, and is kept in the widening.

Example 1 (continued): Using the standard widening in our example, we get the following iterations which converge at step 3:

Step	0	1	2	3
P_0	$\bot$	$\top$	$\top$	$\top$
P_1	$\bot$	$\{x = y = 0\}$	$\{0 \leqslant y \leqslant x\}$	$\{0 \leqslant y \leqslant x\}$
P_2	$\bot$	$\{x + y = 2, 0 \leqslant y \leqslant x\}$	$\{2 \leqslant x + y \leqslant 202, 0 \leqslant y \leqslant x \leqslant 102\}$	$\{2 \leqslant x + y \leqslant 202, 0 \leqslant y \leqslant x \leqslant 102\}$
P_3	$\bot$	$\bot$	$\{0 \leqslant y \leqslant x, x \geqslant 101\}$	$\{0 \leqslant y \leqslant x, x \geqslant 101\}$

3.3 Descending sequence

The first way of improving the results of the widened sequence is the general descending/narrowing method, proposed as early as in [27]. It is guaranteed that the solution $(P_k^{\nabla})_{k\in K}$ is a post-fixpoint of the exact function $(F_k)_{k\in K}$, *i.e.,*

$$\forall k \in K, F_k(P_1^{\nabla}, \ldots, P_k^{\nabla}, \ldots) \subseteq P_k^{\nabla}$$

If, for some k, this inclusion is strict, one can improve the solution by iterating the exact function from (P_k^{∇}) (*i.e.,* continuing the iterations without widening). These additional iterations are not guaranteed to terminate (unless a *narrowing* operator is used [27]), but each iterate is a correct approximation of the least fixpoint, meaning that the iterations can safely be stopped at any step.

Example 1 (end): It is the case in our simple example, since

$$F_1(P_0^{\nabla}, P_2^{\nabla}) = \{0 \leqslant y \leqslant x \leqslant 102, x + y \leqslant 202\} \subset P_1^{\nabla}$$

Continuing the iterations without widening, we get one more iteration which provides a fixpoint and cannot be improved further:

Step	0	1	2	3
P_0	$\bot$	$\top$	$\top$	$\top$
P_1	$\bot$	$\{x = y = 0\}$	$\{0 \leqslant y \leqslant x\}$	$\{0 \leqslant y \leqslant x \leqslant 102, x + y \leqslant 202\}$
P_2	$\bot$	$\{x + y = 2, 0 \leqslant y \leqslant x\}$	$\{2 \leqslant x + y \leqslant 202, 0 \leqslant y \leqslant x \leqslant 102\}$	$\{2 \leqslant x + y \leqslant 202, 0 \leqslant y \leqslant x \leqslant 102\}$
P_3	$\bot$	$\bot$	$\{0 \leqslant y \leqslant x, x \geqslant 101\}$	$\{0 \leqslant y \leqslant 202 - x, 101 \leqslant x \leqslant 102\}$

There were two main tracks for reducing the imprecision due to the widening: The first one investigates better widening operators, while the second one consists in proposing widening application strategies.

3.4 Improving the widening operator

3.4.1 Parma widening

[6] proposes a general framework to design new widening operators from an existing one. They use the notion of "limited growth" to guarantee the termination of the computations:

$$P_1 \curvearrowright P_2 \stackrel{def}{=} \begin{cases} \text{if} & \dim P_2 > \dim P_1 \\ \text{or} & \text{codim}(P_2) > \text{codim}(P_1) \\ \text{or} & \sharp cons(P_1) > \sharp cons(P_2) \\ \text{or} & \sharp cons(P_1) = \sharp cons(P_2) \text{ and } \sharp V_1 > \sharp V_2 \\ \text{or} & \sharp cons(P_1) = \sharp cons(P_2) \text{ and } \sharp V_1 = \sharp V_2 \text{ and } \chi(R_1) \gg \chi(R_2) \end{cases}$$

In other words, $P_1 \curvearrowright P_2$ in one of the following cases:

- The dimension of P_2 (*i.e.,* the dimension of the least subspace containing P_2) is strictly greater than P_1's dimension;
- The codimension of P_2 (the dimension of the largest subspace strictly included in P_2) is strictly greater than P_1's codimension;
- The number of constraints of P_1 is strictly greater than the one of P_2;
- The number of constraints of P_1 is the same as the one of P_2, and the number of vertices of P_1 is strictly greater than the one of P_2;
- P_1 and P_2 have the same number of constraints and vertices, and the multisets $\chi(R_i)$ of the non null coordinates of P_i's rays verify $\chi(P_1) >> \chi(P_2)$ (multiset classic ordering).

This definition guarantees that a sequence $(P_i)_{i\in\mathbb{N}}$ where for all i, $P_i \curvearrowright P_{i+1}$ is finite ($\curvearrowright$ is a well-founded strict ordering). So, the operator ∇' defined by

$$P_1 \nabla' P_2 = \begin{cases} P_1 \sqcup P_2 & \text{if } P_1 \curvearrowright P_2 \\ P_1 \nabla P_2 & \text{otherwise} \end{cases}$$

where ∇ is the standard widening, is also a widening (the operator proposed in [6] is slightly more complicated), and for any P_1, P_2, $P_1 \nabla' P_2 \subseteq P_1 \nabla P_2$.

So, one application of the new widening gives a more precise result. However, there is no guarantee that the limit of the widened sequence will be more precise: most experimental results show an improvement, but there are also counter-examples. Moreover, the cost of the analysis can increase significantly, because the convergence is generally slower.

3.4.2 Limited widening

This technique, called "*widening up to*" in [40, 41] and "*widening with thresholds*" in [12], consists in precomputing a set $\mathcal{U}$ of constraints that are likely to be invariants in a widening location (e.g., the negation of the exit condition of a "for" loop), and in keeping in $P_1 \nabla P_2$ all the constraints in $\mathcal{U}$ which are satisfied by both P_1 and P_2. In [40], the set $\mathcal{U}$ is computed, at a given widening point, as the set of all conditions that permit to come back to this control point, by propagating the exiting condition on the global loops. The main drawback of this method is that propagating the exit condition of all loops has an exponential cost. In general, only the preconditions of the exiting transitions are computed.

Example 2: To illustrate the use of this heuristic, we show in Figure 2 an example taken from [40]: it models the speedometer of a car, which counts the number t of received "seconds" (time elapsed), the number d of received "meters" (distance covered), and the number s of meters received each second (instantaneous speed). It is assumed that (1) the car stops within 4 seconds, (2) the car crashes into a wall after 10 meters, and (3) the maximum speed is 2 meters per second. The goal is to show that the car stops before crashing, *i.e.,* that the control point k_4 is not reachable.

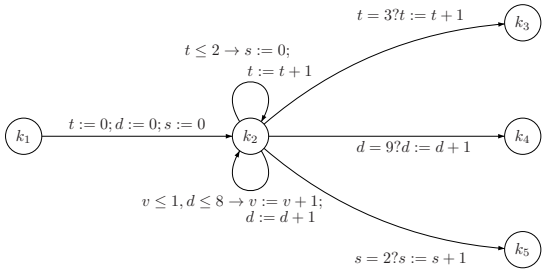


Figure 2: Car example

Let's choose $\mathcal{U}_{k_2} = \{s \leqslant 2, d \leqslant 9, t \leqslant 3\}$ as limiting constraints at the widening point k_2 (these are the postconditions of the guards of the transitions looping on k_2). Then, after the first iteration, we have $P_2^{(1)} = \{d = 0; t = 0; v = 0\}$, and at the second iteration we have to compute

$$P_2^{(2)} = P_2^{(1)} \nabla \{0 \leqslant d = s \leqslant 1; 0 \leqslant t \leqslant 1; t + d \leqslant 1\}$$

The standard widening would give $P_2^{(2)} = \{0 \leqslant s = d; 0 \leqslant t;\}$, but all the constraints in $\mathcal{U}_{k_2}$ are satisfied by both operands of the widening, so the limited widening gives $P_2^{(2)} = \{0 \leqslant s = d \leqslant 2; 0 \leqslant t \leqslant 3\}$. At third iteration, we get

$$\begin{aligned} P_2^{(3)} &= P_2^{(2)} \nabla \{0 \leqslant s \leqslant d \leqslant 2t + s; t \leqslant 3; d \leqslant 2\} \\ &= \{0 \leqslant s \leqslant \boldsymbol{d \leqslant 2t + s}; t \leqslant 3; d \leqslant 2\} \end{aligned}$$

which is a fixpoint, and from which we get $P_4 = \bot$ as expected. Notice that the constraint $d \leqslant 2t + s$ in $P_2^{(3)}$ is the "speed constraint", which is the key property for this example and is missed by the standard widening.

3.4.3 Widening with landmarks

A similar idea is developed in [54], which proposes to compute a set of inequalities that are *not* satisfied while staying in the loop. Unlike the limited widening, this set of constraints is computed dynamically: the computation of the invariant associated to one control point depends on the incoming transitions, and each of these transitions can have a empty contribution. The proposed widening takes into account the guards of these incoming *empty* transitions, in order to guess when they will be enabled.

In practice, with each widening point is associated a set of *landmarks* representing a set of non satisfiable inequalities when the control is inside the associated strongly connected subcomponent: these landmarks are of the form $< c, d_1, d_2 >$ where c is a (non satisfied) constraint, d_1 is the Euclidean distance from the current polyhedron to this constraint, d_2 is the previous distance which is strictly smaller. The design of the widening operator is changed in order to take these landmarks into account: the assumption is that the distance follows an arithmetical law, so we can estimate the number of steps necessary to reach the constraint c; then the algorithm performs an extrapolation of the result of m steps, where m is the minimum of these numbers of steps associated with all considered constraints c .

The main advantage of this method is that it can take some non convex guards like $i \neq 0$ into account. The drawback is that the dynamic computation of the landmarks is costly, because the computation of the distances makes use of linear programming.

3.4.4 Guiding the widening by a care set

In [62], the authors use counterexamples of the property to prove, in order to improve the precision. In fact, they consider the LRA classic process which consists of successive applications of forward and backward iterations. If a forward iterator does not manage to prove the goal formula ψ, then a backward iteration is performed, starting from the polyhedra obtained so far, intersected with $\neg\psi$. The authors propose to use

some information during the backward analysis to get some new constraints that *must* be avoided in the next forward iteration. Then they propose to use this information in one of the following two ways:

- the first one consists in using the negation of each constraint as an "up-to" constraint, and use the classic limited widening operator.
- the second one is to use an extrapolation operator — which does not guarantee the termination — instead of widening. This extrapolation of P with respect to Q under C is defined by the following algorithm (P,Q,C are polyhedra, $P \subseteq Q$, C is the "care set", so $P \cap C = \emptyset$ and $Q \cap C = \emptyset$):
  - Build a new polyhedron P' by dropping from P each constraint c not satisfied by Q, and whose removal does not make P' intersect the care set (*i.e.,* such that $(P \setminus \{c\}) \cap C \neq \emptyset$).
  - From Q, drop any constraint c' not satisfied by P', and return the obtained polyhedron.

  The authors claim that the result doesn't intersect the care set C, and is generally smaller than $P \nabla Q$.

The first solution has the advantage to increase the precision while always guaranteeing the termination. However, this method is guided by a proof goal, and then is only useful in verification. Our goal is to discover better invariants in the general case. This method can also be combined with ours.

3.5 Widening strategies

3.5.1 Delaying the application of the widening operator

An obvious way for improving the precision of the limit of the widened sequence is to delay the application of the widening: during the first m steps, the exact function is applied, then the widening is applied to enforce the convergence. The greater the parameter m, the more precise is the limit. For instance, in our Example 1, with the first very rough widening proposed in [28] (cf. Section 3.2), if the widening is applied at once, we get only

$$P_1 = \{x = y = 0\} \nabla \{0 \leqslant y \leqslant x \leqslant 2 - y\} = \{x \geqslant 0, y \geqslant 0\}$$

but if it is applied only one step later, we get

$$P_1 = \{0 \leqslant y \leqslant x \leqslant 2 - y\} \nabla \{0 \leqslant y \leqslant x \leqslant 4 - y\} = \{0 \leqslant y \leqslant x\}$$

i.e., the result provided by the standard widening without delay. This delaying strategy was proposed in [40, 12]. A variant is the *loop unrolling* technique used in [37, 52]. Of course, this technique may increase significantly the cost of the analysis, since on one hand it involves additional iterations, but also because the first exact steps of computation generally produce complex polyhedra, which would be simplified by the widening.

3.5.2 New control path

The widening technique assumes some regularity in the behavior of the program: the first iterations in a loop are assumed to allow the widening operator to predict the behavior of further iterations. However, this regularity hypothesis is clearly violated when a path in the loop becomes feasible only after some iterations: the effect of taking this path cannot be predicted before it is taken at least once. As a consequence, when a new path of a strongly connected component becomes feasible at some iteration of the analysis, specific strategies should be applied:

- [12] simply applies least upper bound instead of widening;
- [40] proposes to extrapolate the result from the first non empty polyhedron: basically, instead of computing $P_k^{(n+1)} = P_k^{(n)} \nabla F_k(P^{(n)})$, one computes $P_k^{(n+1)} = P_k^{(1)} \nabla F_k(P^{(n)})$.

None of these strategies endangers the termination, since the allowance of a new path can only happen a finite number of times.

3.5.3 Lookahead Widening

The approach proposed in [35, 36] tends to subsume both the limited widening and the new path strategy. The main idea is to make a complete analysis (increasing and decreasing) by *loop phase*. A phase is a period where no new path is being activated.

Intuitively, after a first step of the analysis of a strongly connected component of the control graph, the part of this SCC that has been found feasible is considered alone, and a full — ascending *and descending* — sequence is computed on this subgraph. Then, from the obtained results, new paths are added if they are found feasible, and a full analysis is performed on this extended graph, and so on, until no new path is found.

The lookahead widening is not guaranteed to give more precise results, but experiments show that it often does, at the price of some additional complexity. It is a valuable method since it can easily be combined with other approaches.

4 Acceleration techniques for exact computations

Another track of research concerns the *exact* computation of the reachable states of programs with *integer* variables, the sets of states being characterized by Presburger formulas. Although some authors in this track also use widening techniques [19, 20], most of these works aim at computing reachable states exactly, when possible. For that, the notion of "*acceleration*" has been introduced, to compute the effect of a loop. For instance, the reachable states at the entry of the loop

```
x:= x0; y:=y0; while x<=100 do {x:=x+2; y:=y+3}
```

can be characterized by the Presburger formula

$$\exists k \geqslant 0 \text{ s.t. } x = x0 + 2k \wedge y = y0 + 3k \wedge \forall k', (0 \leqslant k' < k) \Rightarrow (x0 + 2k' \leqslant 100).$$

Let us recall some decidability results for the iteration of some classes of transitions, and then describe how they are used in reachability analysis.

4.1 Theoretical results

Several results have been obtained in the past 15 years:

- In [14], the authors use periodic sets in order to represent the reachability sets of integer programs: the tuple (Δ, δ, P, q) where Δ, P are (integer) matrices and δ, q are vectors, represents the set $\{\vec{x} \in \mathbb{Z}^n; \exists \vec{k} \in \mathbb{Z}^n$, such that $\vec{x} = \Delta\vec{k} + \delta$ and $P\vec{k} \leqslant q\}$. Then the authors give an algorithm to compute the exact effect of affine guarded transitions $(A\vec{x} \leqslant B \rightarrow \vec{x} := C\vec{x} + D)$ with integer constants on a given input periodic set, but only for the transitions where $C^2 = C$.
- [13] and [31] use an automaton encoding of sets to compute the iteration of a Presburger set by an affine guarded function verifying $\exists p, C^{2p} = C^p$. In §6.6, we will come back to this algebraic characterization.
- [23, 24] only consider guards of the form $\bigwedge y_j \# y_i + c$, where $\# \in \{=, <, \leqslant\}$, and y_i are primed or non primed variables (output and input variables). The actions are of the form $x' = Cx + d$. They give a constructive procedure to compute a Presburger formula on x, x' which is true if and only if $x' \in \tau^*(x)$. But this procedure has a very high complexity and has not been implemented.

4.2 Application to reachability analysis

These results are then used for reachability analysis in the following ways:

- In the Lash tool [45], the user designates which loops should be accelerated. Then the analysis is performed forward. Termination is not guaranteed but if the analysis succeeds, the result is the exact reachability set.
- In [13], the author's prototype detects if some loop is accelerable, and if so, adds the corresponding *meta-transition* which subsumes the effect of the whole loop. In the case of two nested loops, the termination is not guaranteed, except for the special case where the internal loop is deterministic, that is, the number of its iterations is completely determined by the initial values of the variables.
- The Fast [30] tool uses the UBA (automata to encode integer sets, [46]) representation to accelerate Presburger sets. In practice, if all the transitions in the graph loops have a finite associated monoid

(see §6.6), then the authors use an enumeration of restricted linear regular expressions (*i.e.,* words of the form $u_1^* u_2^* \dots u_p^*$ with u_i words on the transition alphabet) to enumerate loop paths [9, 8]. If the procedure terminates, then it returns exactly the set of reachable states. It has been shown [9] that the programs for which this procedure terminates are exactly the counter automata that are *L*-flatable, *i.e.,* such that there exists an equivalent automaton (with respect to the reachability sets) that is flat (*i.e.,* without nested loops). This condition is undecidable. This algorithm has been improved, concerning the enumeration of class paths, and the management of some kinds of nested loops by the *union reduction* ([8]). In the Fast tool, the user can provide some help by using graph strategies.

All these methods are based on the fact that accelerating single loops is sometimes enough to compute the reachability set. In the general case, there are loops that cannot be accelerated, and above all, general programs contain nested loops that cannot be flattened. In conclusion, these methods have the advantage of being exact. But they involve computations of high complexity in Presburger arithmetic. Moreover, they are restricted to some classes of programs, and cannot be straightforwardly integrated with general methods like LRA.

5 Motivating example: the gas burner

To motivate our contribution, let us start with a classical example of hybrid system, the "leaking gas burner" [21]: whenever the gas burner leaks, it is always fixed within 10 seconds, and the minimum interval between two leakages is 50 seconds. The standard modelling of this system is by a linear hybrid automaton [1, 42] (see Fig. 3).

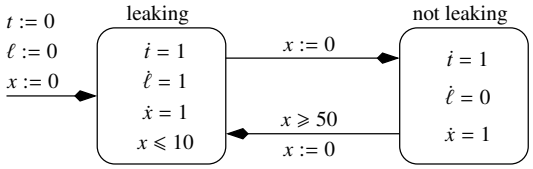


Figure 3: Hybrid automaton of the gas burner

Let us recall the behavior of this model: 3 continuous variables — t, counting the absolute time, ℓ counting the global leaking time, and x counting the time elapsed since the last event — are initialized to 0, and initially the gas burner is leaking. In the leaking location, all the variables evolve continuously with unit derivatives: they all count time, as long as the *invariant* $x \leqslant 10$ is satisfied; at any time during this delay, the gas burner can be fixed, in which case the automaton enters the location "not leaking", and x is reset to 0; in the "not leaking" location, the variable ℓ doesn't change (its derivative is 0), while the other two count time; the automaton may go back to the "leaking" location only when $x \geqslant 50$.

LRA was extended in [1, 41] to deal with such linear hybrid automata. The abstract semantics of discrete transitions is standard. To deal with the continuous evolution of variables in a given location, the "time elapse" operator on polyhedra was introduced: let D be a polyhedron representing the domain of variable derivatives in a given location; let I be the polyhedron representing the invariant attached to the location. Assume the location is entered while the variable values belong to some polyhedron P. Then, the set of variable values which can be reached during the stay in the location is

$$\{x + td \mid x \in P, d \in D, t \geqslant 0\} \cap I$$

which can be computed as $(P \nearrow \{V, R\}) \cap I$, where (V, R) is a system of generators of D: one adds as rays to P the vertices and rays of D, and intersects the result with the invariant.

In Fig. 4, we represent the projection onto the variables t and ℓ of the successive polyhedra computed when analyzing the hybrid automaton of the gas burner:

- At step 1, the "leaking" control point is reached with the unique point $\{t = \ell = 0\}$, and the time elapse operator gives the segment $\{0 \leqslant t = \ell \leqslant 10\}$. Thus, the "not leaking" control point is reached with the polyhedron $\{0 \leqslant t = \ell \leqslant 10\}$, and the time elapse operator gives the polyhedron $\{0 \leqslant \ell \leqslant 10, t \geqslant \ell\}$.

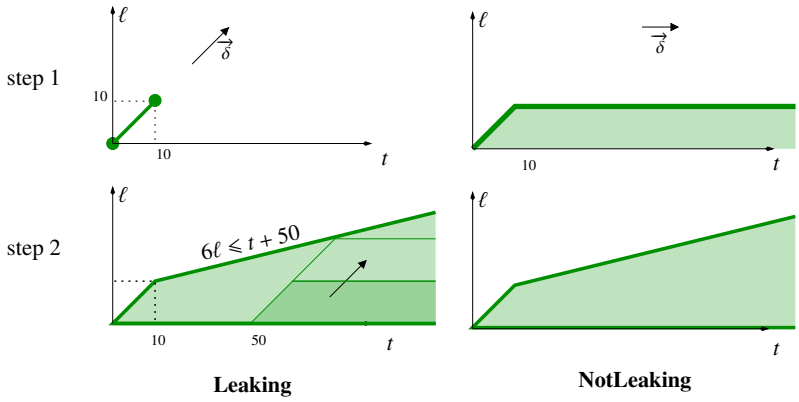


Figure 4: Analysis of the hybrid gas burner

- At step 2, the contribution of the return transition from "not leaking" to "leaking" is the polyhedron $\{0 \leqslant \ell \leqslant 10, t \geqslant \ell + 50\}$ (in dark in the bottom-left figure), then the time elapse operator is applied, then the convex hull with the preceding polyhedron is computed, and finally the widening operator provides $\{0 \leqslant \ell \leqslant t, t \geqslant 6\ell - 50\}$. If we propagate we find the same polyhedron for the "not leaking" location, and we stop.

Notice that the results are the exact convex hulls of the reachable points.

Now let us consider a discrete version of the gas burner, where continuous variables are replaced by counters (Fig. 5): in the *L*(eaking) location, the 3 variables are incremented as long as $x \leqslant 9$. In the *N*(on leaking) location, only t and x are incremented.

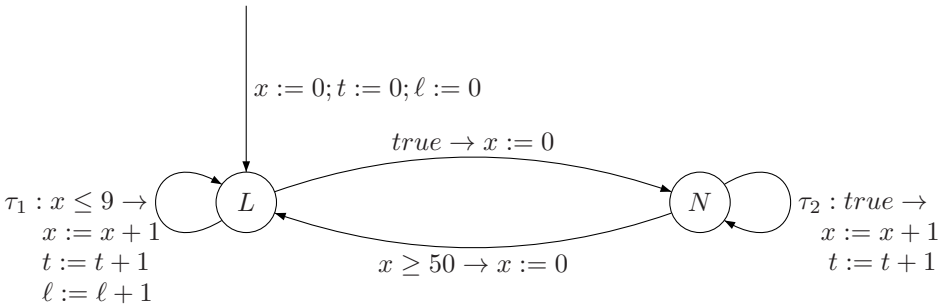


Figure 5: Discrete automaton of the gas burner

Let us apply the classical LRA to this automaton: In L, we get first $t = \ell = 0$, then $t = \ell = 1$ (with no contribution back from N), so the convex hull is $\{0 \leqslant t = \ell \leqslant 1\}$, and the widening provides $\{0 \leqslant t = \ell\}$. The complete analysis terminates with

$$P_L = \{0 \leqslant x \leqslant \ell, \ell \leqslant t\} \quad P_N = \{0 \leqslant x, 0 \leqslant \ell; x + \ell \leqslant t\}.$$

This result is much less precise than in the continuous case, and is not improved by the descending sequence. To obtain better results, we should delay the widening for at least 60 iterations (to get the same invariant as for the hybrid version). Of course, delaying the widening in such a way is expensive; moreover it is rather *ad-hoc*, and it would not work if the constants of the problem were replaced by some symbolic parameters.

This example shows that the analysis can give much better results on a hybrid automata than on the corresponding discrete counter automata. The obvious reason is the availability of the "time elapse" operator,

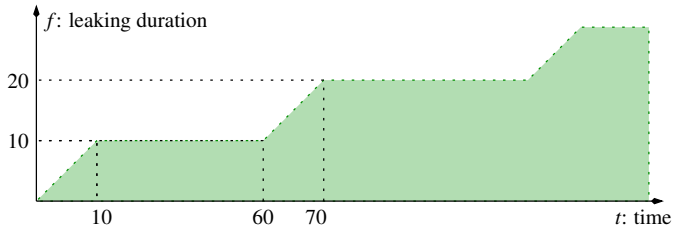


Figure 6: Behaviour of the discrete gas burner

which plays the role of a specialized exact widening operation. A result of the following sections will be to detect that the effect of the single loops in the counter automaton of Fig. 5 can be computed exactly, so that these loops can be treated as single transitions, exactly as it is done by using the time elapse operator on hybrid automata. In other words, instead of analyzing the automaton of Fig. 5, we will apply the standard analysis to the automaton of Fig. 7, where $\tau_1^{\otimes}$, $\tau_2^{\otimes}$ denote the operations subsuming the effect of the two single loops in the initial automaton. In this standard analysis, the two single loops will be accelerated, but the widening is still applied, e.g., at L, because of the remaining global loop.

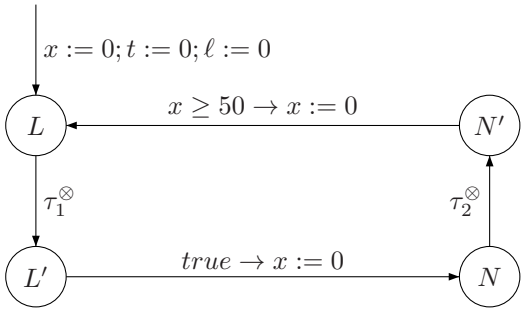


Figure 7: The "accelerated" automaton of the gas burner

The rest of the paper is devoted to identifying the loops and sets of loops that can be "accelerated" in that way. We will consider first single loops, *i.e.,* self-looping transitions around a control state.

6 Single loops

6.1 Some definitions and first remarks

Let $(K, k_{init}, \mathcal{T})$ be a program.

Definition 1 *A* loop *of size p around a control point $k \in K$ is a sequence of transitions $(k, \tau_1, k_1) \rightarrow (k_1, \tau_2, k_2) \rightarrow \ldots \rightarrow (k_{p-1}, \tau_p, k)$. A* single loop *is a loop of size* 1*, (k, τ, k).*

In this section, we consider the acceleration of single loops. We restrict ourselves to transitions with affine guards and affine assignments: $\tau : A\vec{x} \leqslant B \rightarrow \vec{x} := C\vec{x} + D$. Such a transition can be expressed as a function from $\mathbb{Q}^n$ to $\mathbb{Q}^n$:

$$\tau : \vec{x} \mapsto \text{ if } A\vec{x} \leqslant B \text{ then } C\vec{x} + D \text{ else } \vec{x}$$

Let P_0 be a polyhedron, we want to characterize the effect of any number of applications of τ on P_0, *i.e.,* to compute

$$\tau^*(P_0) = \{\vec{x} \mid \exists i \in \mathbb{N},\ \exists \vec{x_0} \in P_0,\ \vec{x} = \tau^i(\vec{x_0})\}$$

or, if, for each x_0, we define the sequence $(x_\ell)_{\ell\geqslant 0}$ by $x_\ell = C^\ell x_0 + \sum_{j=0}^{\ell-1} C^j D$:

$$\vec{x} \in \tau^*(P_0) \;\Leftrightarrow\; \exists \vec{x_0} \in P_0, \exists i \in \mathbb{N}, \text{ such that } \vec{x} = \vec{x_i} \text{ and } \forall j \in [0, i-1], A\vec{x_j} \leqslant B.$$

In general, obtaining a general expression for C^ℓ is too expensive, and the quantification over i and j cannot be computed. So, let us look at some cases where the computation is possible; in such cases, the loop will be said to be *accelerable*:

- [59] considers the same kind of loops, and shows that their termination is decidable. The method uses algebraic characterization of the C matrix, but does not provide any loop invariant.
- In [31], the class of linear functions $\lambda\vec{x}.C\vec{x}+D$ such that the cardinal of $\{C^\ell, \ell \in \mathbb{N}\}$ is finite is pointed out to be accelerable. But the upper-bound that is given is too large, and as far as we know, the complexity of finding whether a monoid generated by a matrix is finite or not is an open problem (it is known to be decidable [38]).
- The case where $C^2 = C$ is interesting, since it covers the loops which increment or decrement variables by constants, and/or set variables to constants.
- The simplest case is when $C = I$ (the identity matrix), *i.e.,* when all variables are incremented or decremented by constants. We call such loops *translation loops* and we first consider this simple case.

6.2 Abstract acceleration of a translation function

Let us consider a translation $\tau : A\vec{x} \leqslant B \rightarrow \vec{x} := \vec{x} + D$. First we distinguish the case where τ is not applied at all from the other applications: $\tau^*(P_0) = P_0 \cup \tau^+(P_0)$. To apply τ an arbitrary positive number of times, it is enough that the guard — since it is convex — be satisfied at the first and last applications of the function:

$$\vec{x} \in \tau^+(P_0) \;\Leftrightarrow\; \exists i \in \mathbb{N}^*, \exists \vec{x_0} \in P_0, \vec{x} = \vec{x_0} + iD \wedge A\vec{x_0} \leqslant B \wedge A(\vec{x_0} + (i-1))D) \leqslant B.$$

We are faced with an arithmetic problem: the exact effect of the loop is defined with $i \in \mathbb{N}^*$, but there is no way to compute simply it by elementary operations on polyhedra. To avoid this, we decide to introduce the notion of *dense abstract acceleration*, which is defined as follows:

Definition 2 *Let $\tau : A\vec{x} \leqslant B \rightarrow \vec{x} := \vec{x} + D$ be a translation. We call* dense abstract acceleration *of τ the function $\tau^\otimes : P_0 \mapsto P_0 \sqcup \tau^\oplus(P_0)$ with*

$$\vec{x} \in \tau^\oplus(P_0) \;\Leftrightarrow\; \exists i \in \mathbb{Q}^+, \exists \vec{x_0} \in P_0, \vec{x} = \vec{x_0} + iD \wedge A\vec{x_0} \leqslant B \wedge A(\vec{x} - D) \leqslant B.$$

$\tau^\oplus(P_0)$ is a polyhedron which can be easily computed by means of usual operations on polyhedra:

Proposition 1 *Let $\tau : A\vec{x} \leqslant B \rightarrow \vec{x} := \vec{x} + D$. Then:*

$$\tau^\oplus(P_0) = \Big((P_0 \cap \{A\vec{x} \leqslant B\}) \nearrow \{D\}\Big) \cap \{A(\vec{x} - D) \leqslant B\}.$$

This proposition is illustrated in Figure 8, where g stands for the guard $A\vec{x} \leqslant B$.

Proof The fact that, in Definition 2, $\exists x_0 \in P_0 \cap g, \exists i \in \mathbb{Q}^+$, such that $x = x_0 + iD$, exactly expresses that $x \in (P_0 \cap g) \nearrow \{D\}$, by definition of a ray. The last intersection with $\{A(\vec{x}-D) \leqslant B\}$ results straightforwardly from the guard. ■

The abstract dense acceleration $\tau^\otimes(P_0)$ is guaranteed to contain $\tau^*(P_0)$, but generally it is only an over-approximation of the convex hull of $\tau^*(P_0)$, as shown by the following example:

Example 1 *Let $\tau : \{x \leqslant 11\} \rightarrow x := x + 2$ with $P_0 = \{x = 0\}$. The exact convex hull of $\tau^*(P_0)$ is $\{0 \leqslant x \leqslant 12\}$, while the above definition gives $\tau^\otimes(P_0) = P_0 \sqcup \Big(\{x = 0\} \nearrow (1) \cap \{x \leqslant 13\}\Big) = \{0 \leqslant x \leqslant 13\}$.*

However, the next example shows that, because of arithmetics, the exact behavior of variables can be complex, so computing the exact convex hull is not realistic.

Example 2 *Let $P_0 = \{0 \leqslant x = 2y \leqslant 4\}$ and $\tau : y \leqslant 3 \rightarrow y := y + 2, x := x + 1$. Figure 9 shows the effect of the successive applications of τ to points in P_0. On this figure, we can see that $\tau^*(P_0)$ (in dark) is quite irregular, due to arithmetics. The light part shows the difference between $\tau^\otimes(P_0)$ and $\tau^*(P_0)$.*

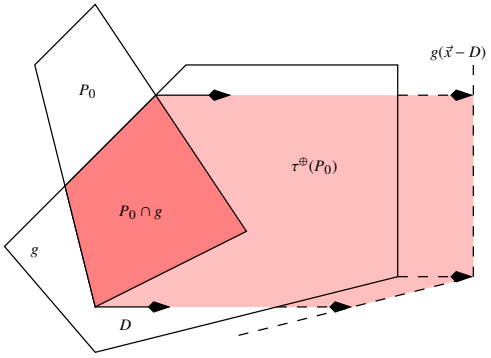


Figure 8: Adding ray algorithm

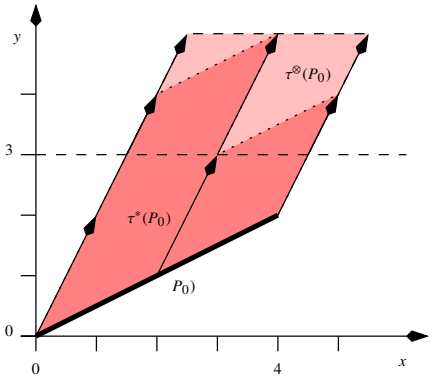


Figure 9: Approximation of a complex arithmetic behavior

In the particular case of Example 1, the Omega algorithm ([51]), which uses integer linear programming, would give the exact result, and thus its convex hull. In the general case, the linear programming algorithms give better results *in the case where the program deals with integer variables*. Indeed, the projection algorithm in the Omega test uses the projection of one integer variable on polyhedra *intersected with a lattice* (here the word lattice is used for $\mathbb{Z}^n$), and thus cannot be used in our general context.

6.3 The gas burner example with abstract acceleration

Applying the results of the section 6.2 to the example of Fig. 5 and **??**, we obtain the following expressions for the acceleration of τ_1 and τ_2:

$$\tau_1^{\otimes}(P) = P \nearrow D_L \sqcap \{x \leqslant 10\} \text{ and } \tau_2^{\otimes}(P) = P \nearrow D_N$$

where $D_L = (1,1,1)$ and $D_N = (1,1,0)$ are the translation vectors of the translation loops around control points L and N, respectively.

- Step 1. $P_L^1 = \{x = t = \ell = 0\} \nearrow D_L \sqcap \{x \leqslant 10\} = \{0 \leqslant x = t = \ell \leqslant 10\}$. Then, $P_N^1 = \tau_2^{\otimes}[x := 0](P_L^1) = \{\ell + x = t, \ell \leqslant 10, 0 \leqslant \ell \leqslant t\}$.
- Step 2.
  - **Location L:** We compute the contribution $[x := 0](P_N^1 \sqcap \{x \geqslant 50\})$, then the convex hull with P_L^1, to obtain $P_L^{\sqcup} = \{x = 0, 0 \leqslant 6\ell \leqslant t, \ell \leqslant 10\}$. Then, we apply the widening operator: $P_L^1 \nabla \tau_1^{\otimes}(P_L^{\sqcup})$, and we obtain $P_L^2 = \{\mathbf{6\ell \leqslant t + 5x}, 0 \leqslant x \leqslant 10, x \leqslant \ell\}$.
  - **Location N:** A similar computation without widening provides: $P_N^2 = \{\mathbf{6\ell + x \leqslant t + 50}, \ell + x \leqslant t, 0 \leqslant l, 0 \leqslant x\}$.

- Step 3 shows the convergence.

We have obtained in 3 steps the same invariants as in the hybrid version, without delaying the widening. Let us notice that the expressions of $\tau_1^{\otimes}$ and $\tau_2^{\otimes}$ compute the exact effect of the loops because variables are incremented by one at each time.

6.4 Abstract acceleration of a translation/reset function

If τ is a translation combined with some reset, that is, τ can be rewritten in the following way:

$$\tau : A\vec{x} \leqslant B \rightarrow \begin{cases} \vec{y} := \vec{y} + D_y & \text{(translated variables)} \\ \vec{z} := D_z & \text{(reset variables)} \end{cases}, \text{ where } \vec{x} = \begin{bmatrix} \vec{y} \\ \vec{z} \end{bmatrix}$$

we can obtain a similar definition and algorithm for $\tau^{\otimes}(P_0)$. Indeed, a translation/reset behaves like a translation of vector $\begin{pmatrix} D_y \\ 0 \end{pmatrix}$ if the number i of loop iterations is strictly greater than 1. So we compute the acceleration of $P_1 = \tau(P_0)$, thanks to the fact all points in $\tau(P_0)$ satisfy $Z = D_z$.

Definition 3 $\tau^{\otimes} : P_0 \mapsto P_0 \sqcup \tau^{\oplus}(\tau(P_0))$ *with:*

$$\tau^{\oplus} : P_1 \mapsto \left\{ \vec{x} \mid \exists i \in \mathbb{Q}^+, \exists \vec{x}_1 \in P_1,\ g(\vec{x}_1) \wedge g\Big(\vec{x} - \begin{bmatrix} D_y \\ 0 \end{bmatrix}\Big),\ \vec{x} = \vec{x}_1 + i \begin{bmatrix} D_y \\ 0 \end{bmatrix} \right\}.$$

Proposition 2 *Let $\tau : A\vec{x} \leqslant B \rightarrow \vec{x} := C\vec{x} + D$ with C a diagonal matrix with only 1 and 0 on the diagonal. Then:*

$$\tau^{\oplus}(P_1) = \Big((P_1 \cap \{A\vec{x} \leqslant B\}) \nearrow \left\{ \begin{bmatrix} D_y \\ 0 \end{bmatrix} \right\}\Big) \cap g\Big(X - \begin{bmatrix} D_y \\ 0 \end{bmatrix}\Big).$$

6.5 The finite monoid class

In this section, we consider a transition $\tau : A\vec{x} \leqslant B \rightarrow \vec{x} := C\vec{x} + D$, where the matrix C verifies $\exists p, C^{2p} = C^p$. We have already seen in section 4 that [13] and [31] obtain acceleration results for this particular class of transitions. Let's show that, in this case, we are able to compute an over-approximation of $\tau^*(P_0)$ by using a reduction to translation/reset functions.

Lemma 1 *Let $\tau : A\vec{x} \leqslant B \rightarrow \vec{x} := C\vec{x} + D$ be a guarded affine transition with $\exists p \geqslant 1, C^{2p} = C^p$. Then the computation of $\tau^*(P_0)$ can be reduced, in polynomial time in p and n (the size of C), to the computation of the iterated image of p polyhedra by an affine guarded transition τ' whose matrix C' verifies $C'^2 = C'$, i.e., is a projection matrix.*

Proof Let $P' = \tau^*(P_0) = P_0 \sqcup P_1 \ldots \sqcup P_{p-1} \sqcup P_p \sqcup \ldots$ then we can write:

$$\begin{aligned} P' \quad = \quad & (P_0 \sqcup \tau^p(P_0) \sqcup \tau^{2p}(P_0) \sqcup \ldots) \sqcup (P_1 \sqcup \tau^p(P_1) \sqcup \tau^{2p}(P_1) \sqcup \ldots) \\ & \sqcup \ldots \sqcup (P_{p-1} \sqcup \tau^p(P_{p-1}) \sqcup \ldots). \end{aligned}$$

Then we remark that $\vec{x}$ has an image by τ^p if and only if the condition: $A\vec{x} \leqslant B \wedge A(C\vec{x} + D) \leqslant B \wedge \ldots \wedge A(C^{p-1}\vec{x} + (C^{p-2} + C^{p-3} + \ldots + I)D) \leqslant B$ is verified.

Finally, if we write

$$C' = C^p, B' = \begin{bmatrix} B \\ B - AD \\ B - A((C+I)D) \\ \ldots \\ B - A((C^{p-1} + \ldots + I)D) \end{bmatrix}, A' = \begin{bmatrix} A \\ AC \\ AC^2 \\ \ldots \\ AC^{p-1} \end{bmatrix}$$

and $D' = (C^{p-1} + C^{p-1} + \ldots I)D$, we have $\tau' : A'\vec{x} \leqslant B' \rightarrow C'\vec{x} + D'$, and finally:

$$P' = \bigsqcup_{0 \leqslant i \leqslant p-1} \widetilde{P_p} \text{ with } \widetilde{P_p} = \tau'^*(P_p).$$

So we are led to the computation of p polyhedra images by the affine guarded transition τ' which verifies $C'^2 = C'$. The computation of the new transition is independent of P_0 and costs $4p$ matrix multiplications. Let us notice that the size of the guard of τ' is p times the one of the guard of τ. ■

Lemma 2 *The case where $C^2 = C$ can be polynomially reduced to the case of a diagonal matrix with only* 0*s and* 1*s on the diagonal.*

Proof If $C^2 = C$ then C is a projection matrix so we can move to a base adapted to this projection. We compute fixpoints of C (classically, the kernel of $C - I$), and the points that have a null image (kernel of C). These vectors constitute an (invertible) matrix Q such that $Q^{-1}CQ$ is diagonal with only 0s and 1s on the diagonal. Then, we are able to compute the modified guard, D and P_0. Each operation costs $O(n^3)$. ■

Finally, we obtain:

Proposition 3 *Let $\tau : A\vec{x} \leqslant B \rightarrow \vec{x} := C\vec{x} + D$ be an affine guarded transition verifying $\exists p, C^{2p} = C^p$ and P_0 an input polyhedron. Then we compute an overapproximation of $\tau^*(P_0)$ by reducing it to p computations of abstract accelerations of translation/reset.*

Proof Immediate from lemmas 1 and 2 and the results of the two previous subsections. ■

6.6 Conclusion of the section

The previous subsections have shown that we can compute overapproximations of a whole class of loops, containing swap loops, translation, translation/reset, and some kind of rotations. Let us point out the fact that our algebraic condition is equivalent to the one of [13] ("Is there a power C^p of C which is diagonalizable and which eigenvalues are in the $\{0, 1\}$ set ?") and also of [31] ("is the multiplication monoid $< C >= \{I, C, C^2, \ldots\}$ finite ?"). Deciding one of these questions can be done in $O(n^4)$ where n is the size of C. But p can be very large:

Proposition 4 *[48] A power p such that $C^{2p} = C^p$ can be very big. More precisely, the maximum of such p verifies $\ln(p_{max}) \sim \sqrt{n \ln(n)}$.*

This result influences the algorithm implementation. In [13], such a decision algorithm for finding p is implemented, and the author uses the result to compute the desired (precise) acceleration. In [31, 8], the authors propose semi-algorithms that terminate if and only if the associated monoid is finite, but do not decide if it is the case. In our case, we decided to search for a such p up to a constant (generally 3 or 4). This choice comes from the fact that for larger p, the computation becomes both too expensive and too unprecise (because of the $p - 1$ convex hulls performed at the end of the computation).

7 Multiple loops

In this section, we address the case of multiple single loops around the same control point (Fig. 10), and we focus on translation and translation/reset functions.

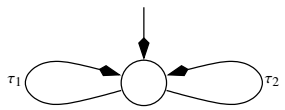


Figure 10: Multiple single loops

7.1 Combined Translation loops

We first consider the case of transition loops $\tau_i : g_i \rightarrow \vec{x} := \vec{x} + D_i$. We give some theoretical results concerning this case, and we give an algorithm to compute efficiently a precise over-approximation of the convex hull of reachable states.

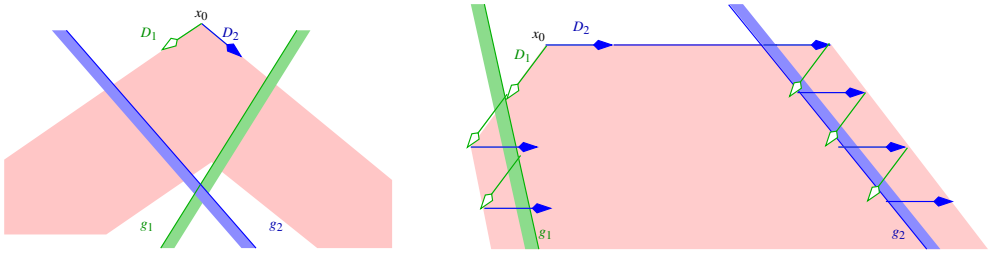


Figure 11: More complex behaviors

First remarks: When there are multiple loops around the same control point, the computed polyhedra become more complex. Even in the case of combined translations, the successive applications of the transitions may introduce non convexity and/or complex oscillations. Consider Fig 11, where we take $P = \{x_0\}$ and we apply a succession of τ_1 and τ_2 actions, when possible (the guards are the half-spaces that are delimited by the deep lines). The left-hand Figure shows a non convex behavior, the right-hand one shows oscillations. This last example shows that the acceleration results of section 6 cannot be applied in the general case because the graph is not flat (the successive applications of τ_1^*, then τ_2^*, then τ_1^* never reach a fixpoint).

The problem to compute $(\tau_1 + \tau_2 + \ldots)^*(P_0)$ is known to be hard, as shown by the previous works on acceleration and also the results obtained for Piecewise Constant Derivative (PCD) systems ([3]). Such a system (in dimension 2) is drawn in Figure 12. The space is divided into polyhedral regions, and a vector is associated to each region. The direction of the trajectory of one point in a region is given by the associated vector, and changes as soon as a frontier is crossed.

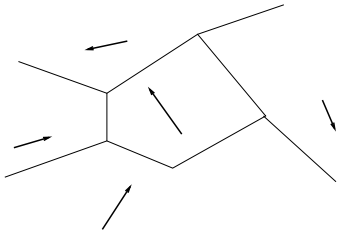


Figure 12: A PCD in dimension 2

Proposition 5 ([5, 4]) *The reachability problem for PCDs in dimension 3 and higher is undecidable.*

Consequently, as the computation of the successive iterations of the τ_i translation functions on the entry polyhedron P_0 can be reduced to reachability in a PCD, our problem is still undecidable in this case.

In this section, we consider the case of two translations. The successive applications of (an over approximation of) τ_1^*, then τ_2^*, then τ_1^* and so on, is not guaranteed to terminate.

7.1.1 A first proposition, partitioning the graph

In [34], we proposed to compute in one step all points that are reachable while continuously satisfying both g_1 and g_2:

Definition 4 *$\tau^\circ_{1,2}(P_0)$ is composed of all $\vec{x}$ that can be obtained from $P_0 \cap g_1 \cap g_2$ by "rationally" applying the two translations τ_1 and τ_2 while staying in $g_1 \cap g_2$:*

$$\begin{array}{ll} \vec{x} \in \tau^\circ_{1,2}(P_0) \text{ iff} & \exists \vec{x}_0 \in P_0 \cap g_1 \cap g_2, \\ & \exists \vec{x}_1, \vec{x}_2 \ldots, \vec{x}_\ell \in g_1 \cap g_2, \exists \vec{x}'_1, \vec{x}'_2 \ldots, \vec{x}'_\ell \in g_1 \cap g_2, \\ & \exists i_1, i_2, \ldots, i_\ell, i'_1, i'_2, \ldots, i'_\ell \in \mathbb{Q}^+, \\ & \textit{such that } \vec{x} = \vec{x}'_\ell, \textit{ and } \vec{x}_j = \tau_1^{i_j}(\vec{x}'_{j-1}), \vec{x}'_j = \tau_2^{i'_j}(\vec{x}_j), j = 1..\ell. \end{array}$$

Remark 1 *Since we compute in the lattice of closed convex polyhedra, we will in fact compute the closure of this preceding set, i.e., the set of all limits of sequences of points staying in $g_1 \cap g_2$ and resulting from a sequence of rational applications of τ_1 and τ_2. This closure will also be noted $\tau^\circ_{1,2}(P_0)$ henceforth.*

The following proposition gives an algorithm to compute (the closure of) $\tau^\circ_{1,2}(P_0)$:

Proposition 6 *Let $\tau_i : g_i \rightarrow \vec{x} := \vec{x} + D_i$, $(i = 1, 2)$, then:*

- *if there exist $\widetilde{x} \in P_0 \cap g_1 \cap g_2$, and $\varepsilon > 0$ such that $\widetilde{x} + \varepsilon D_1 \in g_1 \cap g_2$, or $\widetilde{x} + \varepsilon D_2 \in g_1 \cap g_2$ (i.e., there is at least one point of P_0 from which one of the two transitions can be rationally applied while staying in $g_1 \cap g_2$), then:*

$$\tau^\circ_{1,2}(P_0) = ((P_0 \cap g_1 \cap g_2) \nearrow \{D_1, D_2\}) \cap g_1 \cap g_2.$$

- *otherwise $\tau^\circ_{1,2}(P_0) = P_0 \cap g_1 \cap g_2$.*

Proof Let $P_l = ((\tau_1 + \tau_2)^\circ(P_0))$ et $P_r = ((P_0 \cap g_1 \cap g_2) \nearrow \{D_1, D_2\}) \cap g_1 \cap g_2$.

- $P_r \subseteq P_l$ is obvious.
- $P_l \supseteq P_r$:

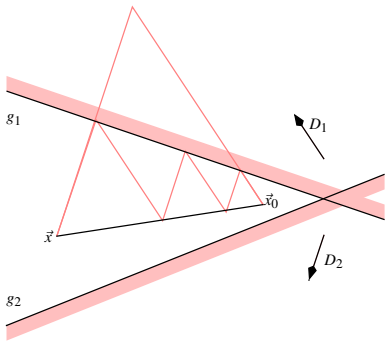


Figure 13: Proof draw

Let $\vec{x} \in P_r$. Then there exists $k_1, k_2 \in \mathbb{Q}^+$, and $\vec{x}_0 \in P_0 \cap g_1 \cap g_2$ such that $\vec{x} = \vec{x}_0 + k_1 D_1 + k_2 D_2$. The hypothesis of the proposition implies that $\vec{x}_0$ (or one point in its neighborhood) satisfies $\exists i_0 > 0, \vec{x}_0 + i_0 D_1 \in P_0 \cap g_1 \cap g_2$. Then let us define $\vec{x}_1 = \vec{x}_0 + i_0 D_1 + \frac{i_0}{k_1} k_2 D_2$. By construction $\vec{x}_1$ belongs to the segment $[\vec{x}_0, \vec{x}]$ and thus, since both $\vec{x}$ and $\vec{x}_0$ satisfy $g_1 \cap g_2$, so does $\vec{x}_1$ (by convexity). We also have $\left\|\vec{x} - \vec{x}_1\right\| < \left\|\vec{x} - \vec{x}_0\right\|$. By iterating the process, we obtain a sequence of $\vec{x}_i$s which converges to $\vec{x}$ (dark line on figure 13). $\vec{x}$ being the limit of points of $\tau^\circ_{1,2}(P_0)$, it belongs to the closure of $\tau^\circ_{1,2}(P_0)$. ■

Remark 2 *The first condition on $P_0 \cap g_1 \cap g_2$ comes from the fact that there must be one (rational) application of τ_1 or τ_2 to initialize the successive iterations. This condition is easy to check by taking N_1 a normal vector to g_1, then the condition is equivalent to $N_1.D_2 > 0$ (scalar product).*

CFG partitioning Now we can partition the control-flow graph in order to use the previous results. The idea is to separate the cases where both transitions are enabled from those where each transition is enabled alone. This operation on the CFG, illustrated in figure 14, consists in:

- creating new control points q_1, q_2, q_{12}. All valuations reaching q_1 (respectively q_2, q_{12}) will satisfy $g_1 \wedge \neg g_2$ (resp. $g_2 \wedge \neg g_1$, $g_1 \wedge g_2$).
- creating the transition for the initialization of these new control points, from q_0 (ε is the identity action).
- creating ending transitions from q_1, q_2, q_{12} to q', with empty transitions (ε on the figure).
- creating single loops around q_1 and q_2, taking into account the location invariant. For instance, around q_1, the transition $(q_1, (pre(g_1 \wedge \neg g_2, a_2), a_1), q_1)$ is created (where $pre(g, a)$ stands for the precondition of g according to a). For q_{12} we add the loop $\tau^\circ_{1,2}$.

- creating the new transitions between these new control points. For instance, on the figure 14, the transition $\mu_{1,2}$ denotes the transition $(q_1, (pre(g_2 \wedge \neg g_1, a_1), \varepsilon), q_2)$.

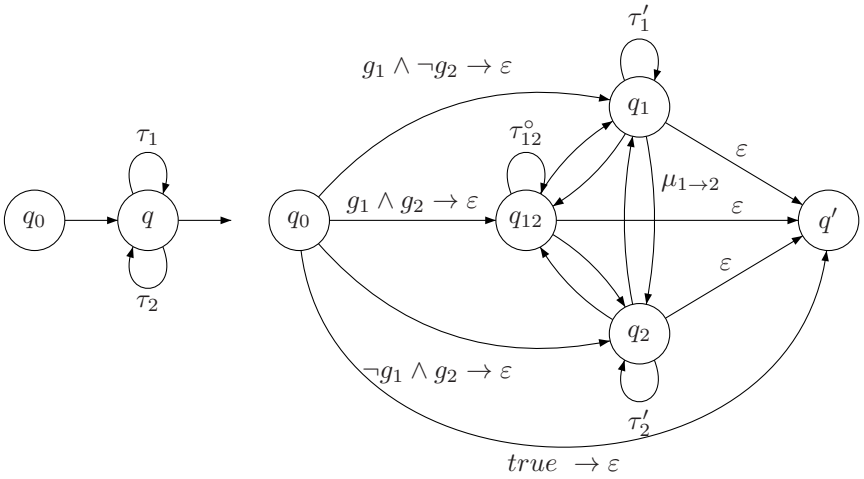


Figure 14: Partitioning the GFC among guards

Remark 3 *The complement $\neg g_2$ of a polyhedron g_2 is not necessarily a polyhedron, but a union of polyhedra. We can then create a new transition for each element of this union.*

After this partitioning, we use the classic LRA strategy, in fact the new partitioned graph has more loops, so new strongly connected sub-components. In particular, q_1, q_2 and q_{12} are new widening points. We can replace however τ_1 and τ_2 by their acceleration, which strongly reduces the number of iterations.

In practice, this partitioning is not realized, because of the induced combinatorial explosion in case of many loops. A first heuristic consists in computing an approximate solution of the system $P = P_0 \sqcup \tau_{1,2}^{\circ}(P) \sqcup \tau_1^{\otimes}(P) \sqcup \tau_2^{\otimes}(P)$, using the widening operator if necessary. Experimentally, it often happens that $P = P_0 \sqcup \tau_{1,2}^{\circ}(P_0) \sqcup \tau_1^{\otimes}(P_0) \sqcup \tau_2^{\otimes}(P_0)$ is already a post-fixpoint, in which case the widening is not applied. Of course, it is one strategy among others, we could compute for instance $P = P_0 \sqcup \tau_{1,2}^{\circ}(P_0) \sqcup \tau_2^{\otimes}(\tau_1^{\otimes}(P_0)) \sqcup \tau_1^{\otimes}(\tau_2^{\otimes}(P_0))$ or other combinations, as the Fast tool does ([7]).

Example 3 *Let us come back to the example of Fig. 1, which can be considered as the CFG of Fig. 15.a. We can compute* $\tau_{1,2}^{\circ}(\{x = y = 0\}) = \{(0,0)\} \nearrow \{(1,1),(2,0)\} \cap \{x \leqslant 100\} = \{0 \leqslant y \leqslant x \leqslant 100\}$ *(see Fig. 15.c) Then we compute the following polyhedra:*

- $\tau_1^{\otimes}(\tau_{1,2}^{\circ}(P_0)) = \{0 \leqslant y \leqslant 100, y \leqslant x, x \leqslant 102\}$
- $\tau_2^{\otimes}(\tau_{1,2}^{\circ}(P_0)) = \{0 \leqslant y \leqslant x \leqslant y + 100\}$

The convex hull of these polyhedra is $\{0 \leqslant y \leqslant x \leqslant 102, y + x \leqslant 202\}$, which is stable by any application of τ_1 or τ_2, so we have reached a post fixpoint. We are then able to propagate the information to control point k_3, for which we get the polyhedron $\{0 \leqslant y \leqslant 202 - x, 101 \leqslant x \leqslant 102\}$. So, we obtain in only one iteration the same results that we got with widening and descending sequence in §3.3.

7.1.2 A direct algorithm

A second possibility is to directly compute an over-approximation of $\bigsqcup(\tau_1 + \tau_2)^*$. We will see that we can compute directly some over-approximations without dealing with integer arithmetic. Then we propose an algorithm to deal with all cases.

First results:

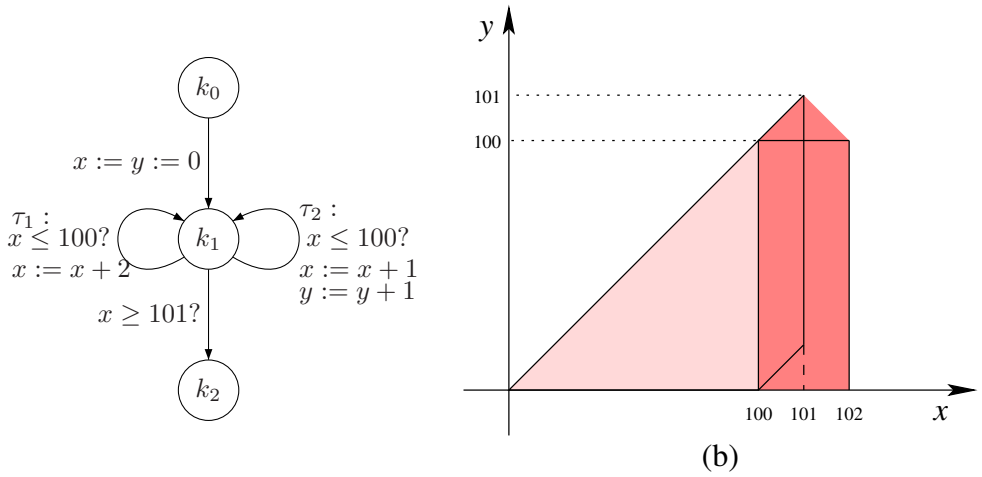


Figure 15: Example with two translations

Proposition 7 *If D_1 is a ray of g_1, D_2 a ray of g_2, then*

$$\bigsqcup(\tau_1 + \tau_2)^*(P_0) = \begin{cases} P_0 & \text{if } P_0 \cap g_1 = \emptyset \text{ and } P_0 \cap g_2 = \emptyset \\ P_0 \nearrow \{D_1\} & \text{if } P_0 \cap g_2 = \emptyset \text{ and } (P_0 \nearrow \{D_1\}) \cap g_2 = \emptyset \\ P_0 \nearrow \{D_2\} & \text{if } P_0 \cap g_1 = \emptyset \text{ and } (P_0 \nearrow \{D_2\}) \cap g_1 = \emptyset \\ P_0 \nearrow \{D_1, D_2\} & \text{otherwise} \end{cases}$$

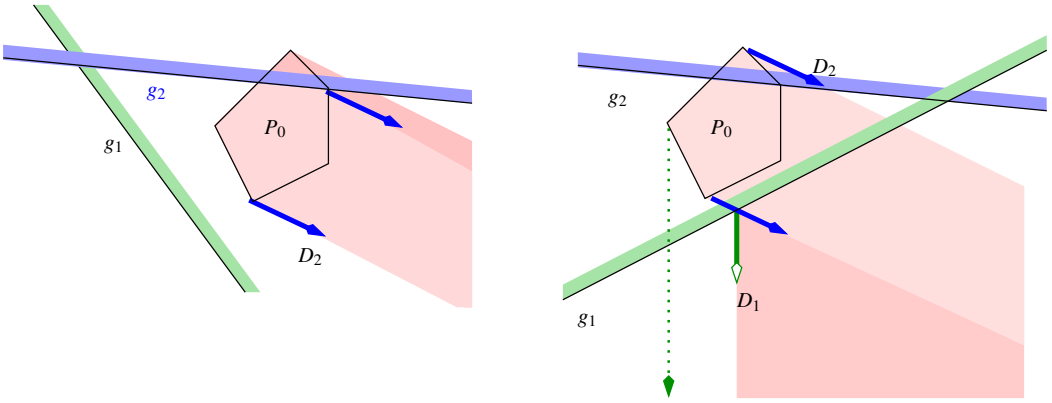


Figure 16: Illustration of Proposition 7

Proof Let P be $\bigsqcup(\tau_1 + \tau_2)^*(P_0)$. If $P_0 \cap g_1 = \emptyset$ and $P_0 \cap g_2 = \emptyset$, then $P = P_0$. Let us now suppose that $P_0 \cap g_1 = \emptyset$ and $P_0 \cap g_2 \neq \emptyset$ (Figure 16). Since D_2 is a ray of g_2, each point of $P_0 \cap g_2 \nearrow D_2$ belongs to P. Then, since P is convex, each point of $P_0 \nearrow D_2$ belongs to P (Figure 16.(a)). Now, there are two possibilities:

- if $P_0 \nearrow \{D_2\} \cap g_1 = \emptyset$, there are no more points. This is the case of Fig. 16.(a).
- if $P_0 \nearrow \{D_2\} \cap g_1 \neq \emptyset$ (Fig. 16.(b)), and the ray D_1 should be added, again because of the convexity of P.

■

Proposition 8 *If D_2 is a ray of both polyhedra g_1 and g_2, then $(\tau_1 + \tau_2)^*(P_0) = \tau_1^*(\tau_2^*(P_0))$.*

Proof Let us show that τ_1 and τ_2 can commute in the following way:

$$\forall X_0 \in \mathbb{R}^n, \forall p_i \in \mathbb{N}, \quad \tau_2^{p_1}(\tau_1^{p_2}(\tau_2^{p_3}(X_0)) = \tau_1^{p_2}(\tau_2^{p_1+p_3}(X_0))$$

Let us name L the left expression and R the right one. Then:

- If $p_1 = 0$, or $p_2 = 0$,or $p_3 = 0$ then $L = R$.
- Else, suppose $p_1, p_2, p_3 \neq 0$ and let $X = L$. As $p_2 \neq 0$, the left expression implies that $\tau_2^{p_3}(X_0) = X_0 + p_3 D_2$ satisfies g_1. As D_2 is a ray of g_1, we get:

  $$\forall i \in \{0, \ldots, p_1\}, g_1(X_0 + p_3 D_2 + iD_2),$$

  so $g_1(\tau_2^{p_1+p_3}(X_0))$. The left expression also implies that, for all $i \in \{0 \ldots, p_2 - 1\}$, $\tau_2^{p_3} + iD_1$ satisfy g_1, so as D_2 is a ray of g_1,

  $$\forall i \in \{0, \ldots, p_2 - 1\}, g_1\Big(X_0 + (p_1 + p_3)D_2 + iD_1\Big),$$

  and then all the τ_1 applications in R are valid and $X = R$.

Thus any point obtained as a combination of τ_1 and τ_2 applications can be also obtained by a combination where all the applications of τ_2 are made before τ_1 ones, which means $(\tau_1 + \tau_2)^*(P_0) \subseteq \tau_1^*(\tau_2^*(P_0))$. Let us notice that this equality is valid even if $P_0 \cap g_2 = \emptyset$, because in this case $\tau_2^*(P_0) = P_0$.

■

This expression gives a simple algorithm to compute an over approximation in this case by using the simple ones obtained in the previous section. We have a similar result for τ_1 by permuting g_1 and g_2 and D_1 and D_2 on the previous proposition.

Algorithm 1 *Let P_0 be the entry polyhedron. Then:*

1. *If D_1 is a ray of g_1 and D_2 a ray of g_2, then (cf. proposition 7):*
   - *If $P_0 \cap g_1 = \emptyset$ and $P_0 \cap g_2 = \emptyset$ then return P_0.*
   - *Else, if $P_0 \cap g_2 = \emptyset$, compute $P_1 = P_0 \nearrow \{D_1\}$, then*
     - *if $P_1 \cap g_2 = \emptyset$, return P_1,*
     - *else return $P_0 \nearrow \{D_1, D_2\}$.*
   - *Else if $P_0 \cap g_1 = \emptyset$, compute $P_2 = \nearrow \{D_2\}$, then*
     - *if $P_2 \cap g_1 = \emptyset$, return P_2,*
     - *else return $P_0 \nearrow \{D_1, D_2\}$.*
   - *Else return $P_0 \nearrow \{D_1, D_2\}$.*
2. *If D_2 is a ray of both g_1 and g_2, then: (cf. Figure 17.(a))*
   - *Compute $P_2 = (P_0 \cap g_2) \nearrow \{D_2\}$ (applications of τ_2 first).*
   - *Compute $P_{21} = (P_2 \cap g_1) \nearrow \{D_1\} \cap post(g_1, D_1)$.*
   - *Return $P_0 \sqcup P_2 \sqcup P_{21}$ (the dotted polyhedron).*
3. *If D_1 is a ray of both g_1 and g_2, then (symmetric case of item 2):*
   - *Compute $P_1 = (P_0 \cap g_1) \nearrow \{D_1\}$ (applications of τ_1 first).*
   - *Compute $P_{12} = (P_1 \cap g_2) \nearrow \{D_2\} \cap post(g_2, D_2)$.*
   - *Return $P_0 \sqcup P_2 \sqcup P_{12}$.*
4. *In all other cases (see Figure 17.(b) and 17.(c)):*
   - *Compute $P_1 = (P_0 \cap g_1) \nearrow \{D_1\} \cap post(g_1, D_1)$. (a segment starting from the point P_0 in Fig. 17.(c), a light dashed polyhedron in Fig. 17.(b))*
   - *Compute $P_2 = (P_0 \cap g_2) \nearrow \{D_2\} \cap post(g_2, D_2)$ (empty in Fig. 17.(c)).*
   - *If $P_1 \cap g_2 \subseteq P_2$ then let $P_{12} = \emptyset$ else compute $P_{12} = (P_1 \cap g_2) \nearrow \{D_2\} \cap post(g_2, D_2)$ (P_{12} is empty in Fig. 17.(b)).*

- *If $P_2 \cap g_1 \subseteq P_1$ then let $P_{21} = \emptyset$ else $P_{21} = (P_2 \cap g_1) \nearrow \{D_1\} \cap post(g_1, D_1)$ (empty set in Fig. 17.(c)).*
- *Compute $P' = P_0 \sqcup P_1 \sqcup P_2 \sqcup P_{21} \sqcup P_{12}$. If this polyhedron is stable by τ_1 and τ_2, then return P' (Fig. 17.(b)). Else, iterate like in classic LRA,* i.e.*, take as P_0 the polyhedron $P_0 \nabla_C (P' \sqcup \tau_1(P') \sqcup \tau_2(P'))$ with with $C = \{post(g_1, D_1), post(g_2, D_2)\}$), and restart the algorithm. (This example is depicted in Fig 17.(c)) where P_0 is reduced to one point and $g_1 = \neg g_2$, P' is the dashed polyhedron, which is not stable, and the result is finally obtained by applying widening).*

Remark 4 *The expressions for cases 2 and 3 are included in the expression of case 4. If the first test is false, we then use the fourth sub-algorithm to compute the desired polyhedron.*

Example 4 *Let us consider again our example 3 (Fig 15). None of the vectors $D_2 = \begin{pmatrix}2\\0\end{pmatrix}$ and $D_1 = \begin{pmatrix}1\\1\end{pmatrix}$ are rays of the guards $g_1 = g_2 = \{x \leqslant 100\}$. So, we compute:*

- *$P_1 = \{0 \leqslant x \leqslant y \leqslant 101\}$.*
- *$P_2 = \{y = 0, 0 \leqslant x \leqslant 102\}$.*
- *$P_{12} = \{0 \leqslant x \leqslant 102, x \leqslant y, 0 \leqslant y \leqslant 100\}$.*
- *$P_{21} = \{y = 0, 0 \leqslant x \leqslant 101\}$.*
- *The convex hull is $\{0 \leqslant y \leqslant x \leqslant 102, x + y \leqslant 202\}$. This set is stable under the applications of τ_1 or τ_2, we do not need any widening application.*

The p loops case We adapt the previous algorithm to the p loop case:

1. For all $i \in [1, p]$, we compute $P_i = (P_0 \cap g_i) \nearrow \{D_i\} \cap post(g_i, D_i)$.
2. For all i, j if $P_i \cap g_j \subseteq P_j$ let $P_{i,j} = \emptyset$ else $P_{i,j} = (P_i \cap g_j) \nearrow \{D_j\} \cap post(g_j, D_j)$
3. Then, let $P' = P_0 \sqcup \bigsqcup_i P_i \sqcup \bigsqcup_{i,j \neq i} P_{i,j}$.
4. If P' is stable by each τ_i, return P', else start again with $P' \nabla_C (\bigsqcup_{i \in [1,p]} \tau_i(P'))$ with $C = \{post(g_1, D_1), post(g_2, D_2), \ldots, post(g_p, D_p)\}$.

7.2 Combined Translation-Reset loops

In this section, we give some partial results concerning the combination of translation and translation/reset.

The Figure 18 subsumes our notations. We decompose our variable vector into two components $\vec{x} = (\vec{y}, \vec{z})$. In the first loop, all the variables are translated while in the second only the variables denoted by $\vec{y}$ are translated, and the variables $\vec{z}$ are reset to 0 (the case of reset to a constant c is very similar).

Remark 5 *Without loss of generality, we can assume that P_0 satisfies $\vec{z} = 0$, since it is the case after the first application of τ_r.*

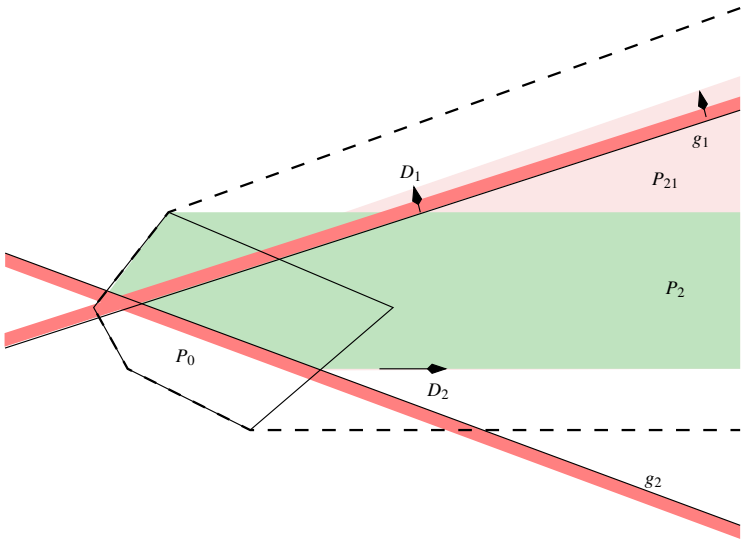
7.3 Simple reset

We first study the case of a translation τ_1 combined with a simple reset τ_2 (*i.e.,* no variables $\vec{y}$).

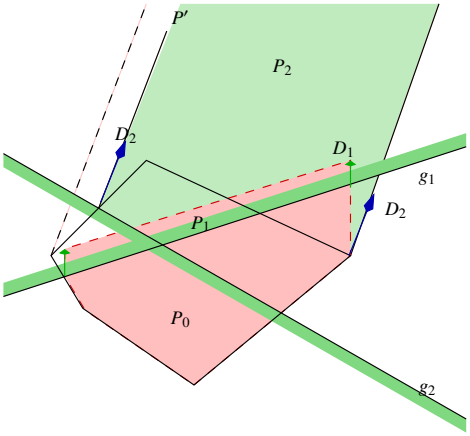
Proposition 9 (Simple reset) *Let $\tau_1 : g_1 \to \vec{x} := \vec{x} + D_1$ and $\tau_2 : true \to \vec{z} := 0$ and assume $P_0 \subseteq g_1 \cap \{\vec{z} = 0\}$. Then, if we note $d_1 = D_1 \downarrow \{\vec{z} = 0\}$ the projection of D_1 on $\vec{z} = 0$, $P_0 \nearrow \{D_1, d_1\} \cap post(g_1, D_1)$ is an over approximation of $(\tau_1 + \tau_r)^*(P_0)$.*

Proposition 10 (simple reset with guards) *Let $\tau_1 : g_1 \to \vec{x} := \vec{x} + D_1$ and $\tau_2 : g_r \to \vec{z} := 0$ and $d_1 = D_1 \downarrow \{\vec{z} = 0\}$. We also assume $P_0 \subseteq g_1 \cap \{\vec{z} = 0\}$, then, let $P' = P_0 \nearrow \{D_1\} \cap post(g_1, D_1)$:*

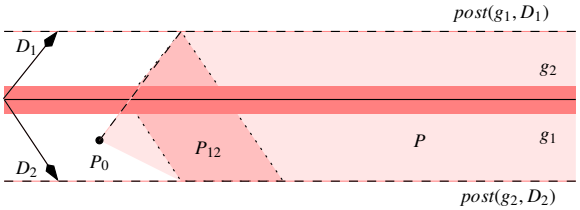
- *if $P' \cap g_r = \emptyset$ then P' is an overapproximation of $(\tau_1 + \tau_r)^*(P_0)$.*
- *else $P_0 \nearrow \{D_1, d_1\} \cap post(g_1, D_1)$ is an overapproximation of $(\tau_1 + \tau_r)^*(P_0)$.*



(a)



(b)



(c)

Figure 17: Illustrations for Algorithm 1

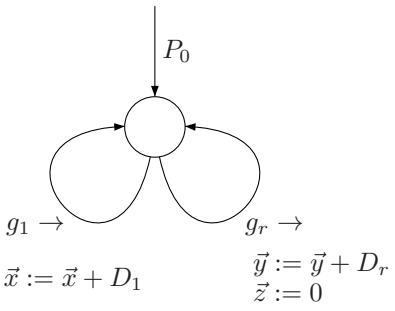


Figure 18: Combination of translation and translation/reset functions

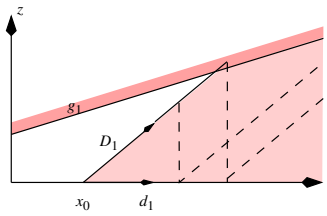


Figure 19: Simple reset

Example 5 *Consider the program of Figure 20.(a). With the previous notations (x is the first variable z the second one), we get:* $D_1 = \begin{pmatrix}1\\1\end{pmatrix}$ *and* $d_1 = \begin{pmatrix}1\\0\end{pmatrix}$. *We compute* $P_0 \nearrow \{D_1\} \cap \{x \leqslant 10\}$, *this polyhedron has a non empty intersection with the constraint* $g_r = \{x = 10\}$. *Finally, we obtain* $P_0 \nearrow \{D_1, d_1\} \cap \{x \leqslant 10\}$, *which is the desired polyhedron. Notice that we get a precise result also with the program of Fig. 20.(b), even if the exact reachable state set is not Presburger definable (the exact acceleration methods cannot converge).*

7.4 Reset with translation

A first result Now let us consider the case where $D_r \neq 0$ does not belong to the plane (D_1, d_1). The consequence is that some variables (called $\vec{y}$) are translated by the second loop out of the plane (D_1, d_1). In the sequel, we first suppose that $g_r = true$, and also that g_1 is of the form $z \leqslant K$ (parallel to the hyperplane $z = 0$, see remark 6). Moreover, and without loss of generality, we assume that $P_0 \subseteq \{z = 0\} \cap g_1$, that D_1 lies in the subspace $\vec{y} = 0$ and D_r lies in the subspace $\vec{z} = 0$ (see Fig. 21).

Then the variables behave as follows:

- (Fig. 21.a) From a point $\vec{x}_0$ satisfying $g_1 \wedge (z = 0)$, we can apply both τ_1 or τ_r. From $\vec{x}_0$, the translation τ_1 can be applied at most k_{max} times, where k_{max} is the least of all quantities $\lfloor K_{z_i}/D_{1_{z_i}} + 1 \rfloor$, for all z_i reset variables (on the figure, $k_{max} = 2$). At any time, τ_r can be applied, as its guard is always true; in this case, variables $\vec{z}$ are reset (projection onto the subspace $\vec{z} = 0$) and a translation according to the vector D_r is performed in the subspace $\{z = 0\}$. Thus, after k applications of τ_1 $(0 \leqslant k \leqslant k_{max})$ followed by one application of τ_r, the current point is defined by $x = x_0 + kd_1 + D_r$ with $d_1 = D_1 \downarrow [z = 0]$ as before.
- (Fig. 21.b) From the points reached so far, the same succession of some (at most k_{max}) applications of τ_1 followed by one application of τ_r can occur.
- Finally, we get the whole reachable domain shown in Fig. 21.c, which is the polyhedron with vertices $\vec{x}_0$, $\vec{x}_0 + k_{max}D_1$ and rays D_r, $k_{max}d_1 + d_r$.

We obtain the proposition:

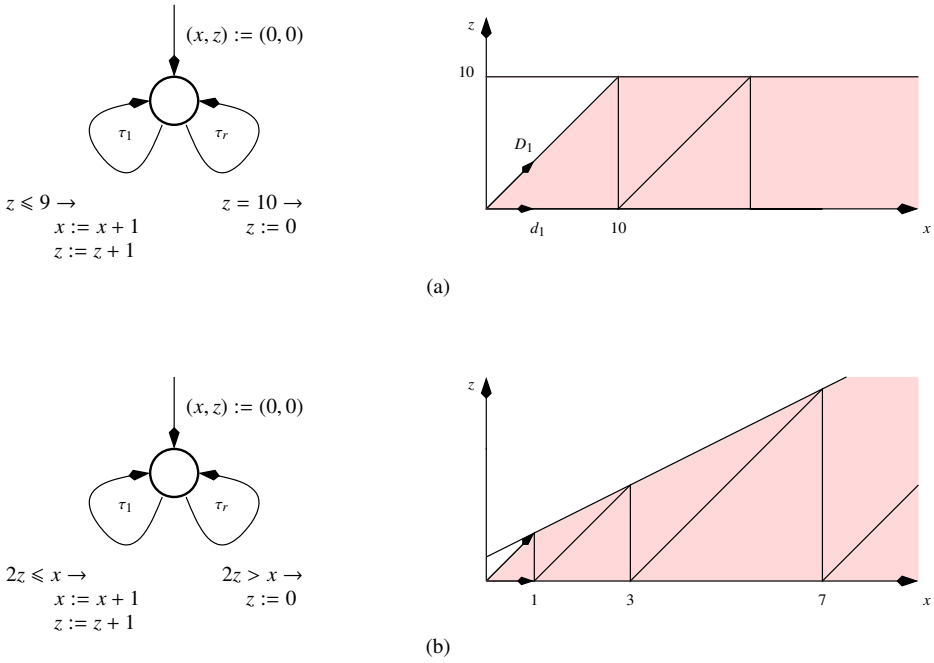


Figure 20: Two loops with simple reset, and the convex hull of reachable states

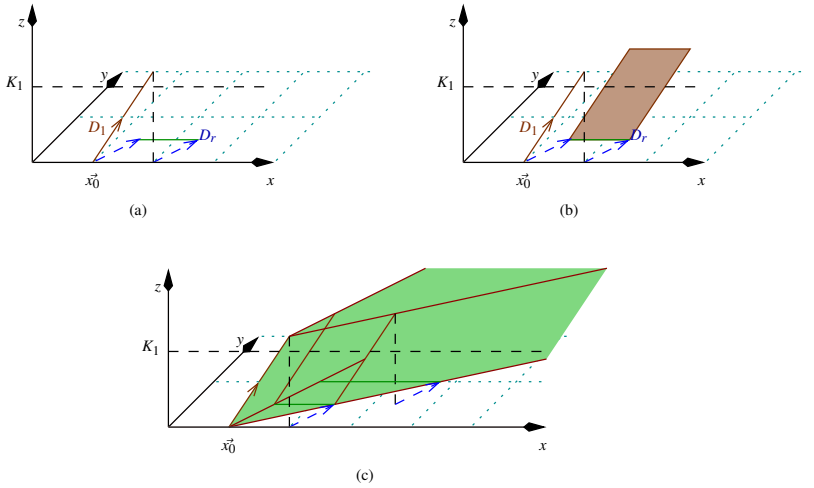


Figure 21: Reset with translation

Proposition 11 *Let τ_1 be of the form $(\vec{z} \leqslant K) \rightarrow \vec{x} := \vec{x} + D_1$ and τ_r : true $\rightarrow \vec{y} := \vec{y} + D_{ry}; \vec{z} := 0$. Suppose $P_0 \subset \{\vec{z} = 0\}$. Let also $d_1 = D_1 \downarrow [\vec{z} = 0]$ and $D_r = \begin{pmatrix} D_{ry} \\ 0 \end{pmatrix}$. Then*

- *If $P_0 \nearrow \{D_1\}$ does not intersect $g_1 : (z \leqslant K)$, then $P_0 \nearrow \{D_1, d_1, D_r\}$ is a convex overapproximation of $(\tau_1 + \tau_r)^*(P_0)$.*
- *Else let $k_{max} = \lfloor K/D_{1z} + 1 \rfloor$, then $P_0 \nearrow \{D_1, D_r, k_{max}d_1 + D_r\} \cap post(g_1, D_1)$ is a convex overapproximation of $(\tau_1 + \tau_r)^*(P_0)$.*

Remark 6 *In contrast with the example of Fig. 20.(b), if the guard g_1 is not of the form $\vec{z} \leqslant K$, the variable behaviour can be non linear, as shown in Fig. 6, one border of the reachable region is then a parabola.*

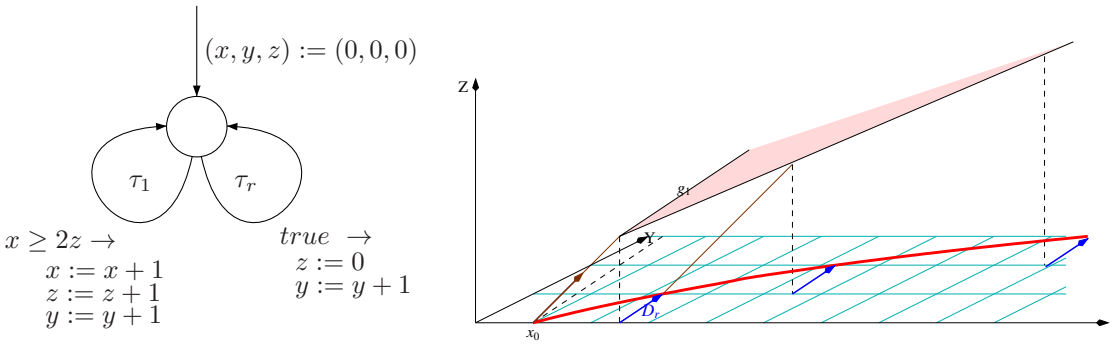


Figure 22: A parabolic behaviour

A more general algorithm The previous result can be extended to more general cases:

- If P_0 is not included in $\{\vec{z} = 0\}$, we first compute $P'_0 = \tau_r(\tau_1^{\otimes}(P_0))$, which is included in $\{\vec{z} = 0\}$.
- The next proposition will deal with more general conditions for g_r.

Proposition 12 *[34] Let τ_1, τ_r of the form:*

$$\tau_1 : (\vec{z} \leqslant K_1) \rightarrow \vec{x} := \vec{x} + D_1 \text{ and } \tau_r : (\vec{z} \bowtie K_r) \rightarrow \vec{y} := \vec{y} + D_r; z := 0$$

where $\bowtie \in \{\leqslant, =, \geqslant\}$. Assume $K_1 > 0$ and $D_1 \cdot u_z > 0$ (i.e., D_1 is not a ray of g_1). Let $k_{max1} = \lfloor K_1/D_{1z} + 1 \rfloor$, $d_1 = D_1 \downarrow [z = 0]$, and $k_{maxr} = \lfloor K_r/D_{1z} + 1 \rfloor$. We also assume then $P_0 \subseteq g_1 \cap \{\vec{z} = 0\}$. Then let P' be the polyhedron computed with the following algorithm:

- *if $\bowtie$ is "$\leqslant$" then*
  - *if $K_r < 0$ then (τ_r is never applied) $P' = P_0 \nearrow \{D_1\} \cap post(g_1, D_1)$*
  - *if $K_r > K_1$ then $P' = P_0 \nearrow \{D_1, D_r, D_r + k_{max1}d_1\} \cap post(g_1, D_1)$*
  - *if $K_1 \geqslant K_r > 0$ then $P' = P_0 \nearrow \{D_1, D_r, D_r + k_{maxr}d_1\} \cap post(g_1, D_1)$*
- *if $\bowtie$ is "$=$" then*
  - *if $K_1 \geqslant K_r > 0$ and $\exists k, K_r = kD_{1z}$ then $P' = P_0 \nearrow \{D_1, D_r + kd_1\} \cap post(g_1, D_1)$*
  - *else (τ_r never applies) $P' = P_0 \nearrow \{D_1\} \cap post(g_1, D_1)$*
- *if $\bowtie$ is "$\geqslant$" then*
  - *if $K_r > K_1$ and $K_r > 0$ then (τ_r never applies) $P' = P_0 \nearrow \{D_1\} \cap post(g_1, D_1)$*
  - *if $K_1 \geqslant K_r \geqslant 0$, then $P' = P_0 \nearrow \{D_1, D_r + k_{max1}d_1, D_r + k_{maxr}d_1\} \cap post(g_1, D_1)$*

– *if* $K_r < 0$ *then* $P' = P_0 \nearrow \{D_1, D_r, D_r + k_{max1} d_1\} \cap post(g_1, D_1)$

Then P' *is a precise overapproximation of* $(\tau_1 + \tau_r)^*(P_0)$.

Remark 7 *If* $D_1 \cdot u_z < 0$ *with the notations of Proposition 12,* g_1 *is always true, then the overapproximation becomes* $P_0 \nearrow \{D_1, d_1, D_r\}$

An algorithm for a combination of translation and translation/reset

Algorithm 2

- *If* $P_0 \subseteq g_1 \cap \{z = 0\}$ *:*
  - *If* $D_r = \overrightarrow{0}$*, or* $D_r \in Vect(D_1, d_1)$*, we first compute* $d_1 = D_1 \downarrow [\vec{z} = 0]$*. Then:*
    - *If* $P_0 \nearrow \{D_1\} \cap post(g_1, D_1) \cap g_r = \emptyset$*, then return* $P_0 \nearrow \{D_1\} \cap post(g_1, D_1)$
    - *Else, return* $P_0 \nearrow \{D_1, d_1\} \cap post(g_1, D_1)$.
  - *Else:*
    - *If* D_1 *is a ray of* g_1 *or the* z *in* g_1 *are not of the form* $z \leqslant K_1$*, then return* $P_0 \nearrow \{D_1, d_1, D_r\} \cap post(g_1, D_1)$.
    - *Else, apply proposition 12.*
- *If* $P_0 \subsetneq g_1 \cap \{z = 0\}$*, by computing* $P_1 = \tau_1^{\otimes}(P_0) = (P_0 \cap g_1) \nearrow \{D_1\} \cap post(g_1, D_1)$*, and* $P_{1r} = \tau_r(P_1 \cap g_r)$ *we reduce the case into the preceding one. We compute the result* P_2*, then* $P' = P_0 \sqcup P_1 \sqcup P_2 \sqcup P_{1r}$*. If this polyhedron is stable under* τ_1 *and* τ_2*, return* P' *else continue classical LRA iterations with* $P_0 \nabla_C P'$ *with* $C = \{post(g_1, D_1)\}$.

Proposition 13 *This algorithm computes an overapproximation of* $(\tau_1 + \tau_r)^*(P_0)$.

8 Aspic tool implementation

In this section, we briefly describe our prototype tool Aspic (Accelerated Symbolic Polyhedral Invariant Computation), which implements most of the techniques proposed in this paper.

Aspic is implemented over a fixpoint generic analyzer called Analyseur ([2]), developed by B. Jeannet at Inria. This tool performs a fixpoint analysis, provided an encoding of the control flow graph and an implementation of the abstract lattice of properties. We chose the polyhedral library NewPolka ([49]).

Aspic takes as input the textual automata input format of the tool Fast ([7]). A Fast file is composed of two parts:

- A "model", which contains a textual description of a *unique* counter automaton: numerical variables, control points and transition functions consisting of a source, a destination, a numerical guard (possibly non convex) and an affine action over the numerical variables.
- A "strategy", which defines "regions", and computation objectives. In contrast with the tool Fast itself, our tool only needs an initial region; an "error region" is only optional, and no additional information is required.

The Aspic input language grammar can be found in Appendix.

Classical Linear Relation Analysis. The Aspic tool makes a forward accessibility analysis, which aims at discovering some polyhedral invariants at each program control point. If an error region is defined (a formula over numerical variables and control points), the goal is transformed into a non accessibility problem by creating new bad states and new transitions; if, after convergence, all the bad states are associated an empty polyhedron, the goal is proved, otherwise the result is inconclusive.

The analysis is performed through the strategy of strongly connected subcomponents given in [16]. The decomposition is precomputed at the beginning of analysis by a variant of Tarjan algorithm [58].

Dealing with undeterministic transitions Some benchmarks pointed out the need for undeterministic affectations. The semantics of counter automata has been extended with a special operation $x' =?$ who basically encodes that every information on x is lost. The expressivity power of the input language is increased, as all affine transitions, that is, transitions of the form $Q(x, x')$ with Q a polyhedron, can now be encoded, as shows lemma 3.

Lemma 3 *A general transition with affine relations can be encoded as the sequence of two extended affine transitions.*

Proof Let $t = (k, Q, k')$ be a general affine transition (Q is a polyhedron of size $2n$. Let $\vec{z}$ be a vector of n fresh variables and k_{new} a new control point. Then, the transition t is equivalent to the combination of t_1 and t_2, where:

- $t_1 = (k, true, a_1, k_{new})$ with $a_1(\vec{x}, \vec{z}) = (\vec{x}, (?)^n)$;
- $t_2 = (k_{new}, g_2, a_2, k')$ where $g_2(\vec{x}, \vec{z}) = true$ iff $(\vec{x}, \vec{z}) \in Q$ (affine guard) and $a_2(\vec{x}, \vec{z}) = (\vec{z}, \vec{z})$ (projection).

In other words, the affine relation is encoded in the guard of the second transition. As an illustration, on figure 23, the transition on the left is equivalent to the pair of transitions on the right.

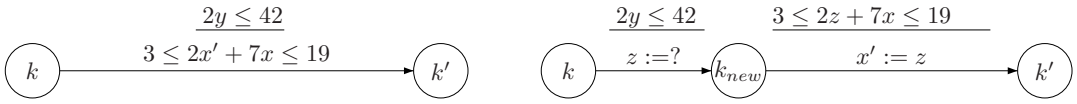


Figure 23: Affine relations encoded with extended affine function

■

Detecting and preprocessing accelerable loops. During the first phase of the analysis the transition functions are preprocessed, an internal structure encodes the type of the action (identity, translation, translation reset, idempotent transition, ...), of the guard (always true, simple, complex, ...), and if the transition is accelerable, and other useful informations that can be precomputed (postconditions, rays to add, ...).

The control structure of the automaton is modified in order to deal with accelerable loops:

- **The unique single loop case** (a unique circuit around the head of the strongly connected subcomponent) is dealt with as follows: if the loop is accelerable, then the control point is split into two points, related with a *meta-transition*, as shown by Fig. 24. This splitting for single loops aims at suppressing the widening at control point q. At q_{split}, the computed polyhedron is $\tau^{\otimes}(P_0)$.

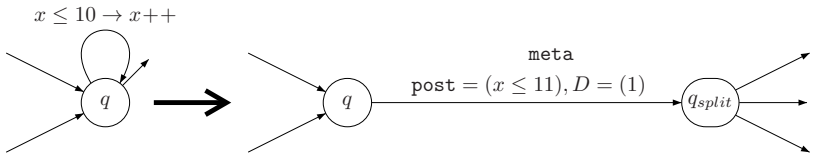


Figure 24: The unique single loop case

- **The multiple single loop case**. For multiple single loops, we also decide to split the control point, as shown in Fig. 25. Multiple loops can be dealt with in two ways:
  - If we have only partial acceleration results, we introduce a return identity edge, which creates a new loop, so a widening node must be chosen among q and q_{split}.
  - If complete acceleration results are available, which means that the multiple loops can be accelerated all together, this return arc is not necessary. This case is similar to the single loop one.

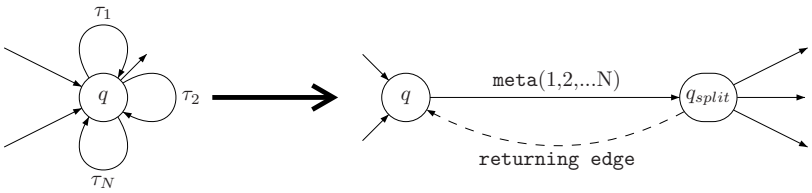


Figure 25: The multiple single loops case

- **The complex loop case**. We deal with this case by (possibly) precomputing the meta transitions associated to the circuits that are detected by the Tarjan algorithm. We compute the associated transformation backward, by composing actions and computing preconditions of guards. For instance, let us consider the following circuit: $(q, \tau_1, q_1)(q_1, \tau_2, q)$ with $q_1 : x \leqslant 7 \rightarrow x := x + 1$ and $q_2 : x \leqslant 4 \rightarrow x := x + 3$. In this case, we compute the transition $q_2 \circ q_1 : x \leqslant 4 \rightarrow x := x + 4$, and it is accelerable, thus we add a meta-transition over the control point q. Both initial transitions are kept in order to preserve the semantic of the CFG. The main drawback of this approach is that not every circuit are detected, in particular in the case where two parallel circuits exists on the same control point (Fig. 26). The detection of every circuit is however too expensive.

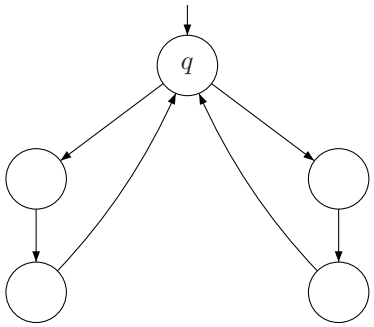


Figure 26: "Parallel" circuits

The choice of widening nodes: Since the first phase modifies the graph structure, the computation of widening control points is done afterwards. Bourdoncle's strategy [16] has been modified as follows: if the head q of a strongly connected subcomponent has been split (with the creation of q_{split}), then q_{split} is chosen as a widening point, instead of q. The reason is that it is better to widen at a control point where the most precise information has been collected. Experiments show that widening after acceleration is a good heuristic.

9 Experimental results

In this section we present some experiments driven with our Aspic tool. These results show that the method we have proposed gives interesting results in terms of precision and efficacity.

9.1 Other approaches to compute invariants

Avoiding widening Some authors study specific cases where a (least) fixpoint can be precisely computed without widening. In the lattice of intervals, the problem is studied in [57], [26] and [33]. [53] proposes a direct method to solve polyhedra fixpoint equations through the use of Farkas lemma and quantifier elimi-

nation. In general, the problem is that these works don't clearly identify the cases where these techniques work.

Using SMT solvers A comparaison to methods using SMT solvers ([55], [60],[61]) is in progress.

9.2 Toy examples

For the toy examples, we compared our method (column "Aspic" in tables) with the following ones:

- Classic Linear Relation Analysis, without widening upto, without the new path algorithm, with two widenings: the standard widening ([39]) and the one proposed in [6]. The first analysis is implemented in Aspic and the StInG tool ([56]) can deal with both of them. The results can be found in column Hal79/BHRZ03. On the toy examples, the invariants are the same.
- The algorithm [53], which is implemented in StInG (column named "STING"). Aspic provides a translator from the Fast language to the input language of StInG.
- The classical LRA with or without widening upto, with the new path algorithm. It has been done with the Aspic tool (column Ch79-v2)
- The LRA with "lookahead widening", which is also implemented in Aspic. The results can be found in column called "lookahead".
- The Fast tool, that implements the exact acceleration techniques described in Section 4. The Fast version we used does not give the fixpoint, but only some information on the accelerated loops. The benchmarks give only an estimation of the computing time.

The description of the automata can be found in Aspic homepage[1]. We present only the results obtained with the different methods on a relevant control point in Fig 27 and 28. No computation time is given because all these analyzes are instantaneous, at the exception of the gas burner and the car analysis with the Fast tool (we stopped these two analysis after 15 min because the Presburger automata were too big at this time (more than 8000 states in each case).

On these examples, we can notice that Aspic often manages to infer more precise invariants than the other tools. When the classic LRA (column CH79-v2) gives the same result, it needs the new path heuristic and the widening upto, and needs more steps to converge, as can be shown in Figure 29. In this last table, we used proof goals and compare the behaviour of classical LRA, Lookahead widening and LRA with acceleration. We indicate the minimum delay used for proving the goal, and the number of global iterations. This table shows the efficiency of Aspic in terms of iteration number.

Example	Proof Goal	CH79_v2	Lookahead	Aspic
swap	$da \leqslant db + 1$	$delay = 4/6it$	$delay = 4/7it$	$delay = 1/1it$
subway	$b \leqslant s \Rightarrow s - b \leqslant 29$	$delay = 1/5it$	$delay = 20/23it$	$delay = 1/4it$
GB	$6\ell \leqslant t + 50$	$delay = 63/65it$	$delay = 63/66it$	$delay = 1/5it$
*wcet*1	$3k \leqslant 10i + 10$	$delay = 11/12it$	$delay = 10/12it$	$delay = 1/4it$
*wcet*2	$20 \leqslant k1$	$delay = 37/39it$	$delay = 37/40it$	$delay = 5/8it$

Figure 29: Some examples with proof goals

9.3 Application to other areas

Reachability analysis on MicMac automaton In [25], the authors propose a new automata formalism to encode SystemC processes. SystemC [50] is a C++ library used to describe and simulate systems on chips, handled as asynchronous compositions of components, managed by a simulation scheduler. For each component, a "Micmac automaton" is generated, in which two kinds of states are distinguished:

- The micro states (in black) represent locations where the process owns the control (in SystemC, the scheduling is not preemptive);
- The macro states (in white) are locations where the process can give back the control to the scheduler.

[1] http://laure.gonnord.org/pro/aspic/benchmarks.html

Name/Method	Ch79/BHRZ03	STING	CH79 v2
*Hal*79*a*	$\left\{ \begin{array}{c} 0 \leqslant j \\ 2j \leqslant i \end{array} \right\}$	idem BHRZ03	$\left\{ \begin{array}{c} 0 \leqslant j \\ 2j \leqslant i \leqslant 104 \end{array} \right\}$
*Hal*79*b*	$\{0 \leqslant y \leqslant x\}$	idem BHRZ03	$\{0 \leqslant y \leqslant x \leqslant 102\}$
SP	$\{x2 + x3 + x4 + x7 = p2, x1 + x2 + x4 + x5 + x6 = p1, \ldots\}$		
GB	$\{0 \leqslant x \leqslant \ell \leqslant t\}$	idem ch79	idem Aspic
*Train*1	$\left\{ \begin{array}{c} d = 9 \\ 20 \leqslant b \\ b \leqslant s + 20 \end{array} \right\}$	$\left\{ \begin{array}{c} \ldots \\ \mathbf{11} \leqslant b \\ 1 \leqslant b - s \leqslant 20 \\ \ldots \end{array} \right\}$	idem BHRZ03
SimpleCar	$\left\{ \begin{array}{c} 0 \leqslant s \leqslant d \\ 0 \leqslant t \end{array} \right\}$	idem BHRZ03	idem Aspic
Car	$\left\{ \begin{array}{c} 0 \leqslant s \leqslant d \\ 0 \leqslant t \end{array} \right\}$	idem BHRZ03	idem Aspic
*Apache*1	idem Aspic	idem Aspic	idem Aspic
*Goubault*1*b*	idem Aspic	idem Aspic	idem Aspic
*Goubault*2*b*	$\{j - i <= 25, j \leqslant 175\}$	$\{\ldots \boldsymbol{i > 150}\}$	idem Aspic
*wcet*1	$\left\{ \begin{array}{c} j \geqslant k, k \leqslant 10 \\ 2k \leqslant a, 0 \leqslant a \leqslant 2k + 5 \end{array} \right\}$	BJRZ03 + $\left\{ \boldsymbol{j \geqslant 10} \right\}$	idem BHRZ03
*wcet*2	idem Aspic	idem Aspic	idem Aspic
*dummy*1	$\{0 \leqslant t_0 \leqslant t\}$	idem CH79	idem CH79
*dummy*4	$\{3t = 3t_0 + z, z \geqslant 0\}$	idem Aspic	idem Aspic
*dummy*6	$\{t + 2a \geqslant 4\}$	$\left\{ \begin{array}{c} a \leqslant 6 \\ \boldsymbol{0 \leqslant t + a - 2} \end{array} \right\}$	$\left\{ \begin{array}{c} 0 \leqslant t \\ 4 \leqslant t + 2a \end{array} \right\}$
*ax*0	$\left\{ \begin{array}{c} t0 = j, 1 \leqslant t0, 1 \leqslant i \\ n - 1 < t0 \\ n - 1 < i \end{array} \right\}$	idem BHRZ03	idem BHRZ03
*t*4*x*0	$\left\{ t0 = j, 1 \leqslant t01 \leqslant i \right\}$	idem BHRZ03	idem BHRZ03

Figure 27: The toy examples with different methods (1)

These automata communicate with each other by means of shared variables. An *ad-hoc* product is then performed, which takes the different possible schedulings into account. The result is an interpreted automaton that encodes all the possible behaviors. The final goal is to check trace inclusion between two such automata. On figure 30, we show a result of such a product. We remark that these automata have no nested loops, the only loops coming from "elapsing of simulation time".

The main limitation of this method is the size of the generated automata. This is why people use existing verification tools in order to discard unreachable states and transitions. Our tool can compute an over approximation of reachable states, and thus can cut off unreachable parts in the resulting product.

A translation of the generated MicMac automata (product) to the Fast format has been given by J. Cornet [25]. The first experiments on toy examples have shown that Aspic permits to make the analysis in a reasonable time and with non trivial results, as we can seen in Fig. 31.

Name/Method	Lookahead	Faster	**ASPIC**
*Hal*79*a*	idem Aspic	*OK*	$\left\{ \begin{array}{c} \mathbf{i + 2j \leqslant 204} \\ i \leqslant 104, 0 \leqslant j \\ 2j \leqslant i \end{array} \right\}$
*Hal*79*b*	idem Aspic	*OK*	$\left\{ \begin{array}{c} 0 \leqslant y \leqslant x \leqslant 102 \\ \mathbf{x + y <= 202} \end{array} \right\}$
SP	$\{x2 + x3 + x4 + x7 = p2, x1 + x2 + x4 + x5 + x6 = p1, \ldots\}$		
GB	$\{0 \leqslant x \leqslant \ell \leqslant t\}$	$> 15min$	$\left\{ \begin{array}{c} \mathbf{6\ell \leqslant t + 5x} \\ 0 \leqslant x \leqslant 10 \\ x \leqslant \ell \\ 0 \leqslant \ell \end{array} \right\}$
*Train*1	idem BHRZ03	*OK*	idem BHRZ03
SimpleCar	$\left\{ \begin{array}{c} 0 \leqslant s \leqslant d \\ \mathbf{s \leqslant 4} \\ 0 \leqslant t \end{array} \right\}$	$> 15min$	$\left\{ \begin{array}{c} 0 \leqslant s \leqslant 4 \\ s \leqslant d \\ \mathbf{d \leqslant 4t + s} \end{array} \right\}$
Car	$\left\{ \begin{array}{c} 0 \leqslant s \leqslant d \\ \mathbf{s \leqslant 2} \\ 0 \leqslant t \end{array} \right\}$	*OK*	$\left\{ \begin{array}{c} 0 \leqslant s \leqslant 2 \\ s \leqslant d \\ \mathbf{d \leqslant 2t + s} \\ t \leqslant 3 \end{array} \right\}$
*Apache*1	idem Aspic	*OK*	$\left\{ c < tk\_sz, 0 \leqslant c \right\}$
*Goubault*1*b*	idem Aspic	*OK*	$\left\{ \begin{array}{c} i + 2j = 21 \\ j \leqslant 8, 6 \leqslant j \end{array} \right\}$
*Goubault*2*b*	idem CH79V2	*OK*(4*s*)	$\left\{ j \leqslant 175, i < 177, 98 \leqslant j \right\}$
*wcet*1	idem CH79V2	*OK*(4*s*)	$\left\{ \begin{array}{c} 0 \leqslant 2k \leqslant a \leqslant 2k + 5 \\ \mathbf{3k < 10i + 10, k \leqslant 5i} \end{array} \right\}$
*wcet*2	$\left\{ \begin{array}{c} \mathbf{5 < k1} \\ \mathbf{20 \leqslant 3k1} \\ \ldots \end{array} \right\}$	*OK*(4*s*)	$\left\{ \begin{array}{c} k = 5, j = 10, i = 5 \\ 2k1 \leqslant a \\ 0 \leqslant k1 \\ a \leqslant 2k1 + 5 \end{array} \right\}$
*dummy*1	idem Aspic	*OK*	$\left\{ 0 \leqslant t_0 \leqslant t \leqslant t_0 + 10 \right\}$
*dummy*4	idem Aspic	*OK*	$\left\{ \begin{array}{c} 3t = 3t_0 + z \\ 0 \leqslant \mathbf{z \leqslant 56} \end{array} \right\}$
*dummy*6	idem CH79V2	*OK*	$\left\{ \begin{array}{c} \mathbf{4 \leqslant t + 2a \leqslant 16} \\ -2 \leqslant t - a \leqslant 1 \end{array} \right\}$
*ax*0	idem CH79V2	$> 3min$	$\left\{ \begin{array}{c} t0 = j, 1 \leqslant \mathbf{t0 \leqslant i} \\ n - 1 < t0 \end{array} \right\}$
*t*4*x*0	idem CH79V2	$> 3min$	$\left\{ \begin{array}{c} t0 = j, 1 \leqslant i, \\ t0 <= n + 1, \\ i <= n + 1, 1 \leqslant t0 \end{array} \right\}$

Figure 28: The toy examples with different methods (2)

Example (nb vars)	Product (#loc,#trans)	Aspic #of acc loops	Aspic (#loc/#trans)	Aspic time	IF Tool time
Ex1 (3)	(47,93)	5	(30,36)	0.036s	some ms
Ex2 (3)	(42,104)	13	(22,35)	0.040s	some ms
Ex3 (4)	(144,653)	1	(144,350)	0.060s	>12min
Ex4 (4)	(180,508)	13	(50,87)	0.044s	>12min

Figure 31: Use of Aspic for reachability analysis in MicMac

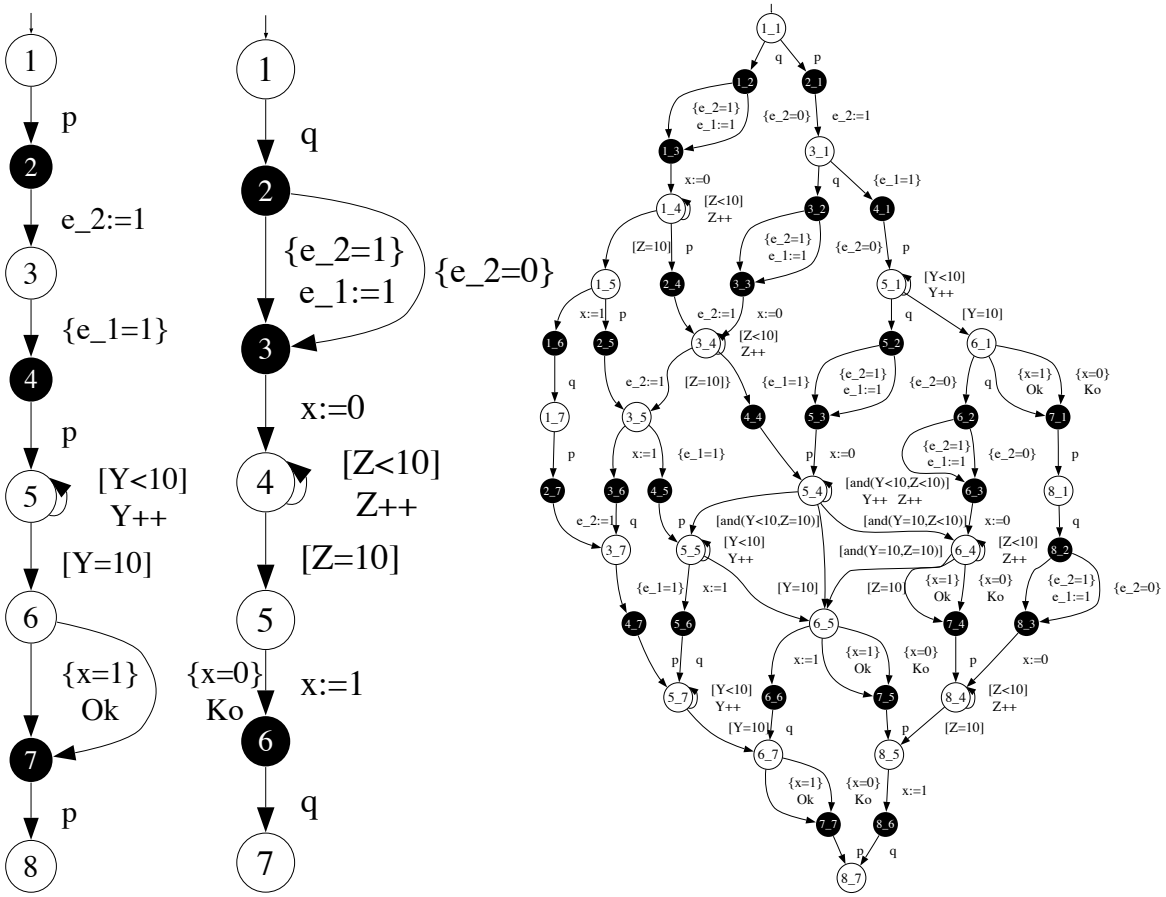


Figure 30: Two MicMac automata and their product

On each example, we give the number of locations and transitions of the generated automaton, then the number of accelerated loops; then the number of the locations/transitions potentially reachable given by Aspic, and then the analysis time. The results given by the IF tool [17, 18] are exact — since IF deals with timed automata —, but the analysis time can be very long (see the last two examples) and the analysis splits the control, so tracing back the results to the source automaton is more difficult.

Programs with lists [15] proposes a translation of programs manipulating lists to counter automata. This translation is made by making abstraction of lists by some "list segments", all the elements of which having the same behaviour. This modeling is often too rough, so some counters are generated in order to take the lists sizes into account. The tool [44] implements this algorithm from programs with lists to the Fast format. Aspic thus permits to verify some properties on program with lists. For the time being, the translation induces several intermediate control points, and the Fast tool does not manage to deal with such programs in the absence of strategy. In many cases, Aspic permits to verify the non reachability of the error states in a reasonable time. However, the generated automata does not involve complex numerical properties for now.

10 Conclusion

When experimenting standard Linear Relation Analysis on various examples, two phenomena become rapidly evident:

1. There is a significant class of programs where the analysis finds the most precise information, *i.e.,* the least abstract fixpoint;
2. On the other hand, it is much more difficult to get precise results about other programs which, however, have an obviously linear behavior: the "discrete derivative" of some variables — *i.e.,* their average variation with respect to loop counters — is constant, or bounded by constants.

This paper explains the former phenomenon, and proposes some solutions to the later one.

We first performed an extended review of the existing proposals in order to improve the results of widened iterations. The message, here, is that while being a heuristic technique, the widening should be applied with some principles in mind. Particularly, the fact that one application of a widening operator is more precise than with another operator, does not ensure that the limit of the iterations will be more precise. Another conclusion, that we used afterwards, is that using only two successive terms of the iteration is often a too strong limitation: looking at the program, *i.e.,* at the way one term is computed from the previous one, can pay off in defining widening strategies.

After this survey of the background, we looked at cases where the least abstract fixpoint should be computable. While being strongly influenced by results of exact acceleration, we decided to stay in the framework of abstract interpretation, because, on one hand, we strongly believe that simplifying sets and formulas is the key to acceptable performances, and on the other hand, we want to keep general applicability: if the exact fixpoint cannot be computed, an extrapolated approximation should still be provided.

We first identified the simple translation loops, which explain the first phenomenon, where standard LRA provides precise results. On this kind of loops, our abstract acceleration computes the result without iteration, thus improving the efficiency w.r.t. classical methods. The second phenomenon often appears when the limitations on derivatives are expressed by means of counter limitations, *i.e.,* by combining translations and resets in loops. We have also studied this case, and identified many situations where the abstract effect of the loop can be computed almost exactly. In these situations, our analysis is more precise, and generally more efficient than existing methods.

References

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science B*, 138:3–34, January 1995. 5, 5

[2] ANALYSER. http://pop-art.inrialpes.fr/people/bjeannet/analyzer/index.html. 8

[3] Eugene Asarin, Oded Maler, and Amir Pnueli. Reachability analysis of dynamical systems having piecewise-constant derivatives. *Theoretical Computer Science*, 138(1):35–65, 1995. 7.1

[4] Eugene Asarin, Gordon J. Pace, Gerardo Schneider, and Sergio Yovine. Algorithmic analysis of polygonal hybrid systems, part ii: Phase portrait and tools. *Theor. Comput. Sci.*, 390(1):1–26, 2008. 5

[5] Eugene Asarin, Gerardo Schneider, and Sergio Yovine. Algorithmic analysis of polygonal hybrid systems, part i: Reachability. *Theor. Comput. Sci.*, 379(1-2):231–265, 2007. 5

[6] R. Bagnara, P. M. Hill, E. Ricci, and E. Zaffanella. Precise widening operators for convex polyhedra. In R. Cousot, editor, *Static Analysis: Proceedings of the 10th International Symposium*, volume 2694 of *Lecture Notes in Computer Science*, pages 337–354, San Diego, California, USA, 2003. Springer-Verlag, Berlin. 3.4.1, 9.2

[7] S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. Fast: Fast acceleration of symbolic transition systems. In *CAV'03*, pages 118–121, Boulder (Colorado), July 2003. LNCS 2725, Springer-Verlag. 1, 7.1.1, 8

[8] Sébastien Bardin. *Vers un model checking avec accélération plate de systèmes hétérogènes*. Thesis, Laboratoire Spécification et Vérification, ENS Cachan, France, October 2005. 4.2, 6.6

[9] Sébastien Bardin, Alain Finkel, Jérôme Leroux, and Philippe Schnoebelen. Flat acceleration in symbolic model checking. In Doron A. Peled and Yih-Kuen Tsay, editors, *Proceedings of the 3rd International Symposium on Automated Technology for Verification and Analysis (ATVA'05)*, volume 3707 of *Lecture Notes in Computer Science*, pages 474–488, Taipei, Taiwan, ROC, October 2005. Springer. 4.2

[10] N. Bjorner, A. Browne, M. Colon, B. Finkbeiner, Z. Manna, H. Sipma, and T. Uribe. Verifying temporal properties of reactive systems: A STeP tutorial. *Formal Methods in System Design*, 16:227–270, 2000. 1

[11] N. Bjorner, I. Anca Browne, and Z. Manna. Automatic generation of invariants and intermediate assertions. *Theoretical Computer Science*, 173(1):49–87, February 1997. 1

[12] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *PLDI 2003, ACM SIGPLAN SIGSOFT Conference on Programming Language Design and Implementation*, pages 196–207, San Diego (Ca.), June 2003. 3.4.2, 3.5.1, 3.5.2

[13] B. Boigelot. Symbolic methods for exploring infinite state spaces. Phd thesis, Université de Liège, 1999. 4.1, 4.2, 6.5, 6.6, 6.6

[14] B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In *CAV'94*, Stanford (Ca.), 1994. LNCS 818, Springer Verlag. 1, 4.1

[15] Ahmed Bouajjani, Marius Bozga, Peter Habermehl, Radu Iosif, Pierre Moro, and Tomáš Vojnar. Programs with lists are counter automata. In Thomas Ball and Robert B. Jones, editors, *Proceedings of the 18th International Conference on Computer Aided Verification (CAV'06)*, volume 4144 of *Lecture Notes in Computer Science*, pages 517–531, Seattle, Washington, USA, August 2006. Springer. 9.3

[16] F. Bourdoncle. Efficient chaotic iterations strategies with widening. In *International Conference on Formal Methods in Programming and their applications*, pages 128–141. LNCS 735, Springer Verlag, 1993. 8, 8

[17] M. Bozga, S. Graf, and L. Mounier. IF-2.0: A validation environment for component-based real-time systems. In *Proceedings of Conference on Computer Aided Verification, CAV'02, Copenhagen*, number 2404 in LNCS. Springer Verlag, 2002. 9.3

[18] Marius Bozga, Susanne Graf, Laurent Mounier, and Iulian Ober. *Real Time Systems 1: Modeling and verification techniques*, chapter Modeling and Verification of Real Time Systems Using the IF Toolbox. Hermes, Lavoisier, 2008. under press. 9.3

[19] T. Bultan, R. Gerber, and W. Pugh. Symbolic model checking of infinite state systems using Presburger arithmetic. In *Computer Aided Verification, CAV'97*. LNCS 1254, Springer Verlag, June 1997. 4

[20] T. Bultan, R. Gerber, and W. Pugh. Model-checking concurrent systems with unbounded integer variables: Symbolic representations, approximations, and experimental results. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 21(4):747–789, July 1999. 4

[21] Z. Chaochen, C.A.R. Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5), 1991. 5

[22] N. V. Chernikova. Algorithm for discovering the set of all solutions of a linear programming problem. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 8(6):282–293, 1968. 2.1

[23] H. Comon and Y. Jurski. Multiple counters automata, safety analysis and Presburger arithmetic. In *CAV'98*, Vancouver (B.C.), 1998. LNCS 1427, Springer Verlag. 1, 4.1

[24] Hubert Comon and Yan Jurski. Timed automata and the theory of real numbers. In *Proc. Conf. on Concurrency theory*, volume 1664 of *lncs*, pages 242–257, Eindhoven, 1999. Springer Verlag. 4.1

[25] Jérôme Cornet, Florence Maraninchi, and Laurent Maillet-Contox. Formalizing systemc transaction-level models of systems-on-chip : a component-based approach. Submitted in, 2007. 9.3, 9.3

[26] A. Costan, S. Gaubert, E. Goubault, M. Martel, and S. Putot. A policy iteration algorithm for computing fixed points in static analysis of programs. In *CAV 2005*, pages 462–475. LNCS 3576, Springer Verlag, 2005. 9.1

[27] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th ACM Symposium on Principles of Programming Languages, POPL'77*, Los Angeles, January 1977. 1, 3.1, 3.3

[28] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *5th ACM Symposium on Principles of Programming Languages, POPL'78*, Tucson (Arizona), January 1978. (document), 1, 3.2, 3.5.1

[29] N. Dor, M. Rodeh, and M. Sagiv. Cleanness checking of string manipulations in C programs via integer analysis. In P. Cousot, editor, *SAS'01*, Paris, July 2001. LNCS 2126. 1

[30] FAST. http://www.lsv.ens-cachan.fr/fast/. 4.2

[31] A. Finkel and J. Leroux. How to compose Presburger-accelerations: Applications to broadcast protocols. In *Proceedings of the 22nd Conf. Found. of Software Technology and Theor. Comp. Sci. (FSTTCS'2002)*, volume 2556 of *Lecture Notes in Computer Science*, pages 145–156, Kanpur, India, December 2002. Springer. 4.1, 6.1, 6.5, 6.6, 6.6

[32] A. Finkel and G. Sutre. An algorithm constructing the semilinear post* for 2-dim reset/transfer vass. In *25th Int. Symp. Math. Found. Comp. Sci. (MFCS'2000)*, Bratislava, Slovakia, August 2000. LNCS 1893, Springer Verlag. 1

[33] T. Gawlitza and H. Seidl. Precise fixpoint computation through strategy iteration. In *ESOP 2007*, pages 300–315. LNCS 4421, Springer-Verlag, March 2007. 9.1

[34] L. Gonnord and N. Halbwachs. Combining widening and acceleration in linear relation analysis. In *13th International Static Analysis Symposium, SAS'06*, Seoul, Korea, August 2006. 7.1.1, 12

[35] D. Gopan and T. Reps. Lookahead widening. In *CAV'06*, Seattle, 2006. 3.5.3

[36] D. Gopan and T. W. Reps. Guided static analysis. In *SAS 2007*, pages 349–365. LNCS 4634, Springer Verlag, 2007. 3.5.3

[37] Eric Goubault. Static analyses of the precision of floating-point operations. In *SAS*, pages 234–259, 2001. 3.5.1

[38] Vesa Halava. Decidable and undecidable problems in matrix theory. Technical Report TUCS-TR-127, University of Turku, 30, 1997. 6.1

[39] N. Halbwachs. Détermination automatique de relations linéaires vérifiées par les variables d'un programme. Thèse de troisième cycle, University of Grenoble, March 1979. (document), 1, 3.2, 9.2

[40] N. Halbwachs. Delay analysis in synchronous programs. In *Fifth Conference on Computer-Aided Verification, CAV'93*, Elounda (Greece), July 1993. LNCS 697, Springer Verlag. 3.4.2, 3.4.2, 3.5.1, 3.5.2

[41] N. Halbwachs, Y.E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, August 1997. 1, 3.4.2, 5

[42] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122, 1997. 1, 5

[43] F. Irigoin, P. Jouvelot, and R. Triolet. Semantical interprocedural parallelization: An overview of the PIPS project. In *ACM Int. Conf. on Supercomputing, ICS'91, Köln*, 1991. 1

[44] L2CA. http://www-verimag.imag.fr/~async/L2CA/l2ca.html. 9.3

[45] LASH. http://www.montefiore.ulg.ac.be/~boigelot/research/lash/. 4.2

[46] J. Leroux. Algorithmique de la vérification des systèmes à compteurs – approximation et accélération – implémentation dans l'outil Fast. Phd thesis, Ecole Normale Supérieure de Cachan, December 2003. 4.2

[47] H. LeVerge. A note on Chernikova's algorithm. Research Report 635, IRISA, February 1992. 2.1

[48] William Miller. The maximum order of an element of a finite symmetric group. *Am. Math. Monthly*, 94(6):497–506, 1987. 4

[49] NewPolka. http://pop-art.inrialpes.fr/people/bjeannet/newpolka/index.html. 8

[50] Open SystemC Initiative. *IEEE 1666: SystemC Language Reference Manual*, 2005. www.systemc.org. 9.3

[51] William Pugh. The omega test: a fast and practical integer programming algorithm for dependence analysis. In *Supercomputing '91: Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, pages 4–13, New York, NY, USA, 1991. ACM Press. 6.2

[52] S. Putot, E. Goubault, and M. Martel. Static analysis-based validation of floating-point computations. In *Numerical Software with Result Verification*, pages 306–313. LNCS 2991, Springer Verlag, 2003. 3.5.1

[53] S. Sankaranarayanan, H. Sipma, and Z. Manna. Constraint-based linear relations analysis. In *International Symposium on Static Analysis, SAS'2004*, pages 53–68. LNCS 3148, Springer Verlag, 2004. 9.1, 9.2

[54] Axel Simon and Andy King. Widening Polyhedra with Landmarks. In *Fourth Asian Symposium on Programming Languages and Systems*, volume 4279 of *Lecture Notes in Computer Science*, pages 166–182. Springer Verlag, November 2006. 3.4.3

[55] Saurabh Srivastava and Sumit Gulwani. Program verification using templates over predicate abstraction. In *PLDI '09: Proceedings of the 2009 ACM SIGPLAN conference on Programming language design and implementation*, pages 223–234, New York, NY, USA, 2009. ACM. 9.1

[56] StInG. http://theory.stanford.edu/~srirams/Software/sting.html. 9.2

[57] Z. Su and D. Wagner. A class of polynomially solvable range constraints for interval analysis without widenings and narrowings. In *TACAS'04*, pages 280–295, Barcelona, 2004. 9.1

[58] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–160, 1972. 8

[59] A. Tiwari. Termination of linear programs. In R. Alur and D. Peled, editors, *Computer-Aided Verification, CAV*, volume 3114 of *LNCS*, pages 70–82. Springer, July 2004. 6.1

[60] Ashish Tiwari. Generating box invariants. In *HSCC'08 : Proceeedings of the 11th International Conference on Hybrid Systems: Computation and Control*, pages 658–661. LNCS 4981, Springer Verlag, 2008. 9.1

[61] Ashish Tiwari and Sumit Gulwani. Constraint-based approach for analysis of hybrid systems. In *Proceedings of the 20th International Conference on Computer Aided Verification (CAV'08)*, volume 5123, pages 190–203, Berlin, Heidelberg, 2008. Springer-Verlag. 9.1

[62] C. Wang, Z. Yang, A. Gupta, and F. Ivančić. Using counterexamples for improving the precision of reachability computation with polyhedra. In *CAV 2007*, pages 352–365. LNCS 4590, Springer-Verlag, July 2007. 3.4.4

[63] D. K. Wilde. A library for doing polyhedral operations. Research Report 785, IRISA, December 1993. 2.1

[64] P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *CAV'98*, pages 88–97, Vancouver, June 1998. LNCS 1427, Springer-Verlag. 1

Aspic input Grammar (Fast Variant)

```
prog: model strat

/*MODEL*/
model : TK_MODEL id TK_LBRACE vars states transitions TK_RBRACE

vars : TK_VAR idlist TK_SEMI

id:
TK_ID

idlist:
id | idlist TK_COMMA id

states : TK_STATES idlist TK_SEMI

transitions : translist

translist :
trans | translist trans

trans : TK_TRANS id TK_ASSIGN TK_LBRACE intrans TK_RBRACE TK_SEMI

intrans : TK_FROM TK_ASSIGN id TK_SEMI TK_TO TK_ASSIGN id  TK_SEMI
          TK_GUARD TK_ASSIGN guard TK_SEMI
          TK_ACTION TK_ASSIGN action TK_SEMI

action :
|  /*do nothing */
| affectation
| action TK_COMMA affectation

affectation :
|id TK_PRIME TK_EQUAL TK_INDET /*x:=?*/
| id TK_PRIME TK_EQUAL numexpr

/*STRATEGY*/

strat : TK_STRATEGY id TK_LBRACE instructionlist TK_RBRACE

instructionlist :
instruction  | instructionlist instruction


instruction :
TK_REGION TK_REGINIT TK_ASSIGN region TK_SEMI
| TK_REGION TK_REGBAD TK_ASSIGN region TK_SEMI
| TK_REGION id TK_ASSIGN region TK_SEMI
| TK_STRATTRANS id TK_ASSIGN  idlistbraq TK_SEMI
| TK_BOOLEAN id TK_ASSIGN boolfastexpr TK_SEMI /*ignore*/
| TK_SETMAXSTATE  TK_LPAREN integ TK_RPAREN TK_SEMI /*ignore*/
| TK_SETMAXACC TK_LPAREN integ TK_RPAREN TK_SEMI /*ignore*/
| TK_PRINT TK_LPAREN chaine  TK_RPAREN TK_SEMI /*ignore*/
```

```
| TK_IF boolfastexpr TK_THEN instructionlist suite  /*ignore*/

suite :
TK_ENDIF
| TK_ELSE instructionlist TK_ENDIF

chaine :
id | TK_DOUBLEQUOTE idandspace TK_DOUBLEQUOTE

idandspace :
id | id idandspace

idlistbraq :
id | TK_LBRACE idlist TK_RBRACE

integ:
| TK_NUM

region :
regionterm /*terminal*/
| region TK_ANDAND region
| region TK_OROR region
| TK_EXCLA region
| TK_LPAREN region TK_RPAREN
| TK_PRE TK_MUL TK_LPAREN region TK_COMMA  idlist TK_RPAREN /*pre
star, ignore*/
| TK_PRE TK_MUL TK_LPAREN region TK_COMMA  idlist TK_COMMA  integ TK_RPAREN /*ignore*/
| TK_POST TK_MUL TK_LPAREN region TK_COMMA  idlist TK_RPAREN /*ignore*/
| TK_POST TK_MUL TK_LPAREN region TK_COMMA  idlist TK_COMMA integ TK_RPAREN /*ignore*/

regionterm :
id | TK_LBRACE guard TK_RBRACE

boolfastexpr :
| id | TK_TRUE | TK_FALSE
| TK_STRATSUBSET TK_LPAREN region TK_COMMA region TK_RPAREN
| TK_STRATEQSET TK_LPAREN region TK_COMMA region TK_RPAREN
| TK_STRATISEMPTY  TK_LPAREN region TK_RPAREN
| boolfastexpr TK_AND boolfastexpr
| boolfastexpr TK_OR boolfastexpr
| TK_EXCLA TK_LPAREN boolfastexpr TK_RPAREN
| TK_LPAREN boolfastexpr TK_RPAREN

/*guards are formula over numerical constraints*/
/*non primed formula*/
guard :
 | TK_LPAREN guard TK_RPAREN
 | guard TK_ANDAND guard
 | guard TK_OROR guard
 | guard TK_IMPLY guard
 | TK_EXCLA  TK_LPAREN guard TK_RPAREN
 | term

term:
```

```
 | TK_TRUE   | TK_FALSE
 | TK_STATE TK_EQUAL id /*bad state*/
 | exprcompare

exprcompare:
 | numexpr TK_GREATEREQUAL numexpr
 | numexpr TK_GREATER numexpr
 | numexpr TK_LESSEQUAL numexpr
 | numexpr TK_LESS numexpr
 | numexpr TK_EQUAL numexpr
 | numexpr TK_NOTEQUAL numexpr

numexpr : /*numerical expressions*/
expr6

expr6:
expr7
| expr6 TK_PLUS expr7
| expr6 TK_MINUS expr7
expr7:
  expr8
| expr7 TK_MUL expr8
| expr7 TK_DIV expr8
expr8:
| numcst
| id
| numcst id
| TK_LPAREN expr6 TK_RPAREN
| TK_MINUS expr8
numcst:
  TK_NUM | TK_RAT
```

Token defs

```
(*keywords*)
|"model"            { TK_MODEL  }
|"var"              { TK_VAR  }
|"states"           { TK_STATES  }
|"transition"       { TK_TRANS  }
|"from"             { TK_FROM  }
|"to"               { TK_TO  }
|"guard"            { TK_GUARD  }
|"action"           { TK_ACTION  }
|"exists"           { TK_EXISTS  }
|"pre"              { TK_PRE  }
|"post"             { TK_POST  }
|"false"            { TK_FALSE  }
|"true"             {  TK_TRUE  }
|"strategy"         {  TK_STRATEGY  }
|"Region"           { TK_REGION  }
|"init"             {  TK_REGINIT  }
|"bad"              {  TK_REGBAD  }
|"state"            {  TK_STATE  }
|"Transitions"      { TK_STRATTRANS  }
```

```
|"boolean"          { TK_BOOLEAN  }
|"True"             {  TK_STRATTRUE  }
|"False"            {  TK_STRATFALSE  }
|"eqSet"            {  TK_STRATSUBSET  }
|"subSet"           {  TK_STRATEQSET  }
|"isEmpty"          {  TK_STRATISEMPTY  }
|"setMaxState"      {  TK_SETMAXSTATE  }
|"setMaxAcc"        {  TK_SETMAXACC  }
|"print"            {  TK_PRINT  }
|"if"               {  TK_IF  }
|"endif"            {  TK_ENDIF  }
|"then"             {  TK_THEN  }
|"else"             {  TK_ELSE  }
|"and"              {  TK_AND  }
|"or"               {  TK_OR  }
(* Delimitors *)
| ","   { TK_COMMA }
| ";"   { TK_SEMI }
| '''   { TK_PRIME }
| "("   { TK_LPAREN }
| ")"   { TK_RPAREN }
| "{"   { TK_LBRACE }
| "}"   { TK_RBRACE }
| "["   { TK_LBRACKET }
| "]"   { TK_RBRACKET }
| ":"   { TK_COLON }
| "\""   { TK_DOUBLEQUOTE }

(* Arithmetic operators *)
| "+"   { TK_PLUS }
| "-"   { TK_MINUS }
| "*"   { TK_MUL }

(* Comparison functions *)
| ">"   { TK_GREATER }
| "<"   { TK_LESS }
| ">="  { TK_GREATEREQUAL }
| "<="  { TK_LESSEQUAL }
| "="   { TK_EQUAL }
| "!="   { TK_NOTEQUAL }

(*assignation*)
| ":="  { TK_ASSIGN }

(*boolean operators*)
| "&&" {TK_ANDAND}
| "||" {TK_OROR}
| "!" {TK_EXCLA}
| "=>" {TK_IMPLY}
| "<=>" {TK_EQUIV}

(*indet*)
| "?" {TK_INDET}
```

Contents