Internship report

Laboratoire d'Informatique Fondamentale de Lille

Lucas Seguinot
August 23, 2013

Improvement of a termination analysis algorithm

Supervisor :	**Team :**
Laure Gonnord	Dart (Dynamic Adaptativity and Real-Time)
Co-supervisor :	**Keywords :**
David Monniaux	static analysis, termination proof, ranking function

Abstract

Deciding whether a C program terminates or not is an undecidable problem. Yet, the termination of some program can be proved by exhibiting a ranking function, i.e., a function from the program states to a well-founded set, which strictly decreases at each program step. The subject of my internship was the improvement an algorithm that proves the termination of C programs that are abstracted as affine interpreted automata by a exhibiting multilinear affine ranking function. My main objectives were to study this algorithm and then to study and implement an improved version, which will fix the scalability issues of the first one. For now, the improved algorithm works with automata which have only one node, but it should not be hard to adapt it to work with any automaton. I implemented this algorithm, corrected several issues, and helped proving its correctness and its termination.

Contents

1 Introduction

As part of my studies as an undergraduate in computer science and telecommunications at the ENS Cachan, Antenne de Bretagne, I have spent two months as an intern in the DART (Dynamic Adaptativity and Real-Time) team, under the supervision of Laure Gonnord. The subject of this internship was the improvement of a termination analysis algorithm. Deciding whether a C program terminates or not is an undecidable problem. Yet, the termination of some program can be proved by exhibiting a ranking function, i.e., a function from the program states to a well-founded set, which strictly decreases at each program step. In [1], the Compsys team has proposed a general algorithm for the computation of linear ranking functions. However, this algorithm has scalability issues. My objectives were first to study this algorithm and then to study and implement the improvement made by Laure Gonnord and David Monniaux.

This report describes the work I have made during this internship, and is organized as follow. In Section 2, we define the concepts and notations used in the rest of the report. Section 3 presents the original algorithm proposed in [1]. Section 4 explains the improvements we made to this algorithm. Section 5 develops the implementation choices I have made. Finally, Section 6 presents the results obtained with the algorithm.

2 Definitions

In this section, we will define the concepts used in the rest of the report. After recalling the definition of closed convex polyhedra, I will present the affine interpreted automata, an abstraction in which we can transform the programs so that they can be easily analyzed, and then the ranking functions, which will allow us to prove that the program represented by an automaton terminates.

We write matrices with capital letters (as A) and column vectors in boldface (as $\boldsymbol{x}$). Sets are represented with calligraphic letters such as $\mathcal{W}$, $\mathcal{K}$, etc.

2.1 Closed convex polyhedra

Definition 1 (Closed convex polyhedron)**.** A set $\mathcal{P}$ is a *closed convex polyhedron* iff it is given as the solutions of a system of non-strict inequalities, i.e. iff there exists a set of couples $(\boldsymbol{v}_i, c_i)$ such that $\mathcal{P} = \{\boldsymbol{x} \mid \bigwedge_i \boldsymbol{v}_i.\boldsymbol{x} \geq c_i\}$.

If bounded, then this closed convex polyhedron is the convex hull of its vertices, but if unbounded one has to include rays as generators:

$$P = \left\{ \left(\sum_i \alpha_i v_i \right) + \left(\sum_i \beta_i r_i \right) \middle| \ \alpha_i \geq 0, \beta_i \geq 0, \sum_i \alpha_i = 1 \right\} \tag{1}$$

In the following, the expressions “convex polyhedron” and “polyhedron” will refer to closed convex polyhedra as defined above.

Example 1 (Closed convex polyhedra).

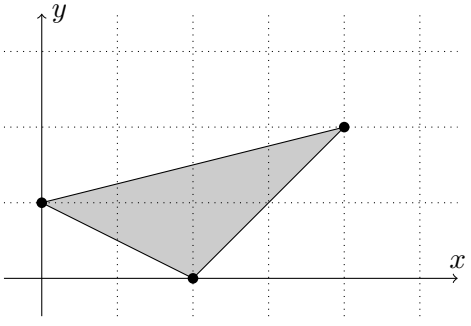

Figure 1 – Bounded polyhedron

The polyhedron represented on Figure 1 can be defined as the couples (x, y) solutions of the following system :

$$\begin{cases} x + 2y \geq 2 \\ -x + y \geq 0 \\ x - 4y \geq 0 \end{cases}$$

It is also the convex hull of the vertices $(0, 1)$, $(2, 0)$ and $(4, 2)$.

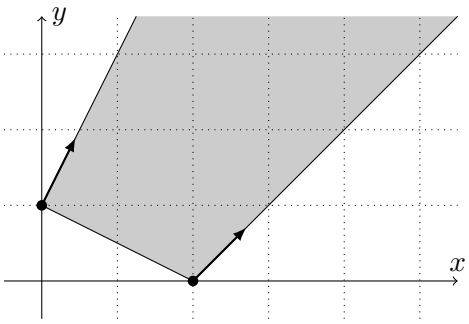

Figure 2 – Unbounded polyhedron

The polyhedron represented on Figure 1 can be defined as the couples (x, y) solutions of the following system :

$$\begin{cases} x + 2y \geq 2 \\ -x + y \geq 0 \\ 2x - y \geq -1 \end{cases}$$

It can also be defined as in Equation (1) by its vertices $\{(0,1),(2,0)\}$ and its ray generators $\{(1,1),(1,2)\}$. ■

2.2 Affine Interpreted Automata

Definition 2 (Affine interpreted automaton)**.** An *affine interpreted automaton* $(\mathcal{K}, n, k_{init}, \mathcal{T})$ is defined by:

- a finite set $\mathcal{K}$ of *control points*
- n rational variables represented by a vector $\boldsymbol{x}$ of size n
- an initial control point $k_{init} \in \mathcal{K}$;
- a finite set $\mathcal{T}$ of 4-tuples (k, g, a, k'), called *transitions*, where $k \in \mathcal{K}$ (resp. $k' \in \mathcal{K}$) is the source (resp. target) control point, $g : \mathbb{Q}^n \mapsto \mathbb{B} = \{\text{true}, \text{false}\}$, the *guard*, is a logical formula (constructed with $\wedge$, $\vee$ and non-strict affine inequalities), and $a : \mathbb{Q}^n \mapsto \mathbb{Q}^n$, the *action*, assigns, to each variable valuation $\boldsymbol{x}$, a vector $\boldsymbol{x}'$ of size n (constructed with affine relations between $\boldsymbol{x}$ and $\boldsymbol{x}'$)

Semantics The set of states is $\mathcal{K} \times \mathbb{Q}^n$. Given a transition $t = (k, g, a, k') \in \mathcal{T}$, the guard g gives a necessary condition on the variables $\boldsymbol{x}$ to traverse t, and the action a gives the value of the variables after traversing t ($x := a(x)$).

A *trace* from $(k_0, \boldsymbol{x}_0)$ to $(k, \boldsymbol{x})$ is a sequence of couples $(k_0, \boldsymbol{x_0}), (k_1, \boldsymbol{x}_1), \ldots, (k_p, \boldsymbol{x}_p)$ such that $k_p = k$, $\boldsymbol{x}_p = \boldsymbol{x}$ and for each i, $0 \leq i < p$, there exists in $\mathcal{T}$ a transition (k_i, g_i, a_i, k_{i+1}) such that $g_i(\boldsymbol{x}_i) = \text{true}$ and $\boldsymbol{x}_{i+1} = a_i(\boldsymbol{x}_i)$. We denote $\mathcal{R}$ the set of reachable states, i.e. the states $(k, \boldsymbol{x})$ such that there exists $\boldsymbol{v} \in \mathbb{Q}^n$ s.t. there exists a trace from $(k_{init}, \boldsymbol{v})$ to $(k, \boldsymbol{x})$, and $\mathcal{R}_k = \{\boldsymbol{x} \in \mathbb{Q}^n \mid (k, \boldsymbol{x}) \in \mathcal{R}\}$ the set of possible valuations $\boldsymbol{x}$ of the variables when the control is in k.

Example 2 (Affine interpreted automaton).

```
y = 0;
x = m;
while(x>=0 && y>=0){
  if(indet()){
    while(y <= m && indet())
      y++;
    x--;
  }
  y--;
}
```

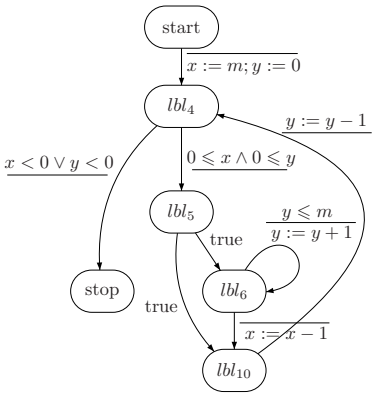

Figure 3 – Illustrating example

Figure 3 gives an example of C code, and of the corresponding affine interpreted automaton. This transformation can be done with the C2FSM tool [5]. The control points are the nodes of the automaton (here, $\mathcal{K} = \{\text{start}, lbl_4, lbl_5, lbl_6, lbl_{10}, \text{stop}\}$). A transition (k, g, a, k') is represented by an arrow from the node k to the node k', labelled by $\frac{g}{a}$ (g is omitted when $g = \text{true}$). Here, we can easily prove that the variables are integers, thus the strict inequalities of the guard of the transition from lbl_4 to stop ($x < 0 \vee y < 0$) are equivalent to non-strict inequalities ($x \leq -1 \vee y \leq -1$). The variable vector is $\boldsymbol{x} = (x, y, m)$. The `indet` function abstracts non-determinism or an intractable test. The outcome of non-determinism is that, in the corresponding automaton, both transitions out of state lbl_5 have a true guard.

If $m = 2$, $((\text{start}, (0,0)), (lbl_4, (2,0)), (lbl_5, (2,0)), (lbl_6, (2,0)), (lbl_6, (2,1)))$ is a trace. ■

The computation of the set $\mathcal{R}_k$ being undecidable in the general case, it is possible to use an over-approximation of this set, thanks to the notion of *invariants*.

Definition 3 (Invariants)**.** An invariant on a control point k is a formula $\phi_k(\boldsymbol{x})$ that is true for all reachable states $(k, \boldsymbol{x})$. It is *affine* if it is the conjunction of a finite number of affine conditions on program variables. The set $\mathcal{R}_k$ is then over-approximated by a polyhedron $\mathcal{P}_k$.

There are several ways to compute invariants. We will use *abstract interpretation* techniques, as proposed by Cousot and Halbwachs in their seminal paper [4], with the help of the ASPIC tool ([5], `http://laure.gonnord.org/aspic/`).

Example 3 (Invariant). The formula $\phi_{lbl_4}(\boldsymbol{x}) = x \geq -1 \wedge y \geq -1 \wedge x \leq m \wedge y \leq m$ is an invariant on the control point lbl_4 of the automaton of Figure 3. ■

2.3 Ranking functions

Let $\succeq$ be the lexicographic order on vectors of $\mathbb{Q}^m$, and for each transition $t = (k, g, a, k') \in \mathcal{T}$

$$\mathcal{Q}_t = \{(\boldsymbol{x}, \boldsymbol{x}') \mid x \in \mathcal{P}_k \text{ and } g(\boldsymbol{x}) = \text{true and } \boldsymbol{x}' = a(\boldsymbol{x})\}$$

and

$$\Delta_t(\rho, \boldsymbol{x}, \boldsymbol{x}') = \rho(k, \boldsymbol{x}) - \rho(k', \boldsymbol{x}')$$

Definition 4 (Ranking function)**.** An affine *weak ranking function* of dimension m is a function $\rho : \mathcal{K} \times \mathbb{Q}^n \to \mathbb{Q}^m$, affine in the second parameter (the variables), positive on all $\mathcal{P}_k$, whose values decrease lexicographically at each transition $t = (k, g, a, k')$:

$$\boldsymbol{x} \in \mathcal{P}_k \Rightarrow \rho(k, \boldsymbol{x}) \geq \mathbf{0} \text{ (component-wise)} \tag{2}$$

$$(\boldsymbol{x}, \boldsymbol{x}') \in \mathcal{Q}_t \Rightarrow \Delta_t(\rho, \boldsymbol{x}, \boldsymbol{x}') \succeq \mathbf{0} \tag{3}$$

We say that a weak ranking function ρ satisfies $(\boldsymbol{x}, \boldsymbol{x}') \in \mathcal{Q}_t$ iff $\Delta_t(\rho, \boldsymbol{x}, \boldsymbol{x}') \succ 0$.

A *strict ranking function* is a weak ranking function ρ that satisfies all $(\boldsymbol{x}, \boldsymbol{x}') \in \mathcal{Q}_t$, i.e. a function such that :

$$(\boldsymbol{x}, \boldsymbol{x}') \in \mathcal{Q}_t \Rightarrow \Delta_t(\rho, \boldsymbol{x}, \boldsymbol{x}') \succ \mathbf{0} \tag{4}$$

Example 4 (Ranking function).

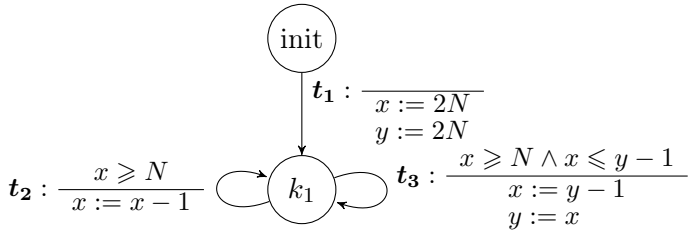

Figure 4 – Monodimensional example

The function ρ defined below is a strict ranking function for the example of Figure 4 :

$$\begin{cases} \rho(\text{init}, \boldsymbol{x}) = 2N + 3 \\ \rho(k_1, \boldsymbol{x}) = 2 + x + y - 2N \end{cases}$$

Indeed, on the state k_1, $x + y$ decreases strictly and is initially equal to $4N$, thus $\rho(k_1, x)$ decreases strictly from $2N + 2 < 2N + 3$. ■

For the first algorithm, the interpreted automata are integer interpreted automata, and we will try to compute a ranking function $\rho : \mathcal{K} \times \mathbb{Z}^n \mapsto \mathbb{N}^m$. $(\mathbb{N}^m, \succeq)$ being a well founded set, the existence of a strict ranking function ensures that there is no infinite trace, i.e. that the corresponding program terminates. However, I had to prove this result in the general case for the improved algorithm.

Proposition 1. *For a given automaton, if there exists a strict ranking function ρ, then there is no infinite trace (i.e. the corresponding program terminates).*

Proof. Let us suppose there exists a strict ranking function $\rho : (k, \boldsymbol{x}) \mapsto L_k\boldsymbol{x} - \boldsymbol{c}_k$ and that there is an infinite trace $(k_i, \boldsymbol{x}_i)_{i\in\mathbb{N}}$. $(\rho(k_i, \boldsymbol{x}_i))$ is a positive and strictly decreasing sequence and thus it has a limit $\boldsymbol{\ell}$.

The set $\mathcal{T}$ being finite, there exists a transition $t = (k, g, a, k') \in \mathcal{T}$ that is traversed an infinite number of times, i.e. there exists a subsequence $(k_{\varphi(i)}, \boldsymbol{x}_{\varphi(i)})$ of $(k_i, \boldsymbol{x}_i)$ such that for all $i \in \mathbb{N}$

$$\begin{cases} k_{\varphi(i)} = k \\ k_{\varphi(i)+1} = k' \\ (\boldsymbol{x}_{\varphi(i)}, \boldsymbol{x}_{\varphi(i)+1}) \in \mathcal{Q}_t \end{cases}$$

As $\mathcal{Q}_t$ is an intersection of closed convex polyhedra, it is a closed convex polyhedron. Let us define

$$\mathcal{S}_t = \mathcal{Q}_t \cap \{(\boldsymbol{x}, \boldsymbol{x}') \mid \rho(k, \boldsymbol{x}) \succeq \boldsymbol{\ell} \wedge \rho(k, \boldsymbol{x}') \succeq \boldsymbol{\ell}\}$$

Since $\rho(k, \boldsymbol{x}) \succeq \boldsymbol{\ell}$ is equivalent to a disjunction of conjunctions of linear inequalities, $\mathcal{S}_t$ is a finite union of closed convex polyhedra $\mathcal{S}_{t,p}$, $1 \leq p \leq m$. Let $(\boldsymbol{v}_{j,p})$ and $(\boldsymbol{r}_{j,p})$ be their generators as defined in Equation (1) :

$$\mathcal{S}_{t,p} = \left\{ \left(\sum_j \alpha_j \boldsymbol{v}_{j,p} \right) + \left(\sum_j \beta_j \boldsymbol{r}_{j,p} \right) \middle| \ \alpha_j \geq 0, \beta_j \geq 0, \sum_j \alpha_j = 1 \right\}$$

For all $i \in \mathbb{N}$, $(\boldsymbol{x}_{\varphi(i)}, \boldsymbol{x}_{\varphi(i)+1}) \in \mathcal{S}_t$, thus we can express $(\boldsymbol{x}_{\varphi(i)}, \boldsymbol{x}_{\varphi(i)+1})$ as a linear combination of the generators of a certain $\mathcal{S}_{t,p_i}$:

$$(\boldsymbol{x}_{\varphi(i)}, \boldsymbol{x}_{\varphi(i)+1}) = \sum_j \alpha_{i,j} \boldsymbol{v}_{j,p_i} + \sum_j \beta_{i,j} \boldsymbol{r}_{j,p_i} \text{ with } \alpha_{i,j} \geq 0, \beta_{i,j} \geq 0 \text{ and } \sum_j \alpha_{i,j} = 1$$

Let $(\boldsymbol{y}_i, \boldsymbol{y}'_i) = \sum_j \alpha_{i,j} \boldsymbol{v}_{j,p_i}$ (see Figure 5). $(\boldsymbol{y}_i, \boldsymbol{y}'_i) \in \mathcal{Q}_t$ which is a closed set, and $(\boldsymbol{y}_i, \boldsymbol{y}'_i)$ is a bounded sequence (because $||(\boldsymbol{y}_i, \boldsymbol{y}'_i)|| \leq \max_p(\sum_j ||\boldsymbol{v}_{j,p}||)$. Thus there exists a subsequence $(\boldsymbol{y}_{\psi(i)}, \boldsymbol{y}'_{\psi(i)})$ of $(\boldsymbol{y}_i, \boldsymbol{y}'_i)$ that has a limit $(\boldsymbol{y}, \boldsymbol{y}') \in \mathcal{Q}_t$.

Moreover, let us denote, for all j and p, $(\boldsymbol{v}^1_{j,p}, \boldsymbol{v}^2_{j,p}) = \boldsymbol{v}_{j,p}$ and $(\boldsymbol{r}^1_{j,p}, \boldsymbol{r}^2_{j,p}) = \boldsymbol{r}_{j,p}$. Then we can notice that necessarily, for all j, p and $a \in \{1, 2\}$, $L_k \boldsymbol{r}^a_{j,p} \succeq \boldsymbol{0}$ (if $L_k \boldsymbol{r}^a_{j,p} \prec 0$, $\rho(k, \beta \boldsymbol{r}^a_{j,p}) \underset{\beta\to\infty}{\longrightarrow} -\infty$, which is in contradiction with the fact that ρ is positive on $\mathcal{P}_k$). Thus

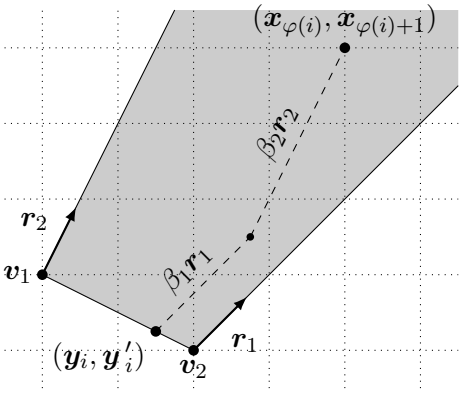

Figure 5 – Construction of $(\boldsymbol{y}_i, \boldsymbol{y}'_i)$

$$\rho(k, \boldsymbol{x}_{\varphi(i)}) = \sum_j \alpha_{i,j}(L_k \boldsymbol{v}^1_{j,p}) + \sum_j \beta_{i,j}(L_k \boldsymbol{r}^1_{j,p}) - \boldsymbol{c} \succeq \sum_j \alpha_{i,j}(L_k \boldsymbol{v}^1_{j,p}) - \boldsymbol{c} = \rho(k, \boldsymbol{y}_i)$$

Finally, $\rho(k, \boldsymbol{x}_{\varphi(\psi(i))}) \succeq \rho(k, \boldsymbol{y}_{\psi(i)}) \succeq \boldsymbol{\ell}$ and, likewise, $\rho(k, \boldsymbol{x}_{\varphi(\psi(i))+1}) \succeq \rho(k, \boldsymbol{y}'_{\psi(i)}) \succeq \boldsymbol{\ell}$. The squeeze theorem gives, with the limits when $i \to \infty$, $\rho(k, \boldsymbol{y}) = \boldsymbol{\ell} = \rho(k, \boldsymbol{y}')$, which is in contradiction with the fact that $(\boldsymbol{y}, \boldsymbol{y}') \in \mathcal{Q}_t$. Thus, there is no infinite trace.

□

Definition 5. A ranking function is said *monodimensional* if its dimension m is 1. Else it is said *multidimensional*.

Example 5 (Monodimensional). The function ρ of Example 4 is a monodimensional ranking function. ■

Example 6 (Multidimensional).

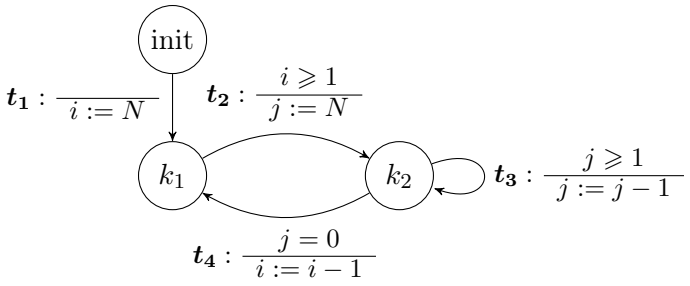

Figure 6 – Multidimensional example

The function ρ defined below is a multidimensional ranking for the example of Figure 6 on the left:

$$\begin{cases} \rho(\text{init}, x) &= (2N+1, 0) \\ \rho(k_1, x) &= (2i, 0) \\ \rho(k_2, x) &= (2i-1, j) \end{cases}$$

■

3 A first algorithm to compute affine ranking functions

In order to find a multidimensional affine ranking function ρ, Alias et al. [1] proposed a greedy algorithm which, at each iteration, computes (via linear programming) a weak ranking function σ that maximizes the number of transitions that strictly makes σ decrease. It then adds σ as a component of ρ and continue on the other transitions. The algorithm terminates when all transitions are satisfied by ρ or when there are no monodimensional weak affine ranking function that satisfies the remaining transitions.

Here, we suppose that the automata are integer interpreted automata, i.e. that the variable are in $\mathbb{Z}$ and we compute a ranking function $\rho : \mathcal{K} \times \mathbb{Z}^n \to \mathbb{N}^m$.

In the following, I will present the algorithms relying on LP-programming, as well as some examples, and the main drawbacks in terms of scalability.

This section is a summary of [2]. The examples have been directly copied from it.

3.1 Computing a monodimensional ranking function

We will rely on the affine form of the Farkas Lemma [7].

Lemma 1 (Farkas lemma, affine form)**.** *An affine form $\phi : \mathbb{R}^n \to \mathbb{R}$ with $\phi(\vec{x}) = \boldsymbol{c}.\boldsymbol{x} + c_0$ is nonnegative everywhere in a non-empty polyhedron $\{\boldsymbol{x} \mid A\boldsymbol{x} + \boldsymbol{a} \geq \boldsymbol{0}\}$ iff:*

$$\exists \boldsymbol{\lambda} \in (\mathbb{R}^+)^n, \lambda_0 \in \mathbb{R}^+ \text{ such that } \phi(\boldsymbol{x}) \equiv \boldsymbol{\lambda}.(A\boldsymbol{x} + \boldsymbol{a}) + \lambda_0$$

The notation $\equiv$ is a formal equality, which means that $\boldsymbol{x}$ can be eliminated and coefficients identified. In other words:

$$\exists \boldsymbol{\lambda} \in (\mathbb{R}^+)^n, \lambda_0 \in \mathbb{R}^+ \text{ such that } \boldsymbol{c} = \boldsymbol{\lambda}.A \text{ and } c_0 = \boldsymbol{\lambda}.\boldsymbol{a} + \lambda_0$$

As the computed ranking functions have integer values, we can notice that a transition t is satisfied by a monodimensional weak ranking function σ if and only if

$$(\boldsymbol{x}, \boldsymbol{x}') \in \mathcal{Q}_t \Rightarrow \Delta_t(\sigma, \boldsymbol{x}, \boldsymbol{x}') \geq 1 \tag{5}$$

Thus, to compute a weak ranking function σ that maximizes the number of transitions of T (where $T \subset \mathcal{T}$ are the transitions unsatisfied by ρ), we can maximize $\sum_{t \in T} \epsilon_t$, such that

$$(\boldsymbol{x}, \boldsymbol{x}') \in \mathcal{Q}_t \Rightarrow \Delta_t(\sigma, \boldsymbol{x}, \boldsymbol{x}') \geq \epsilon_t \text{ with } 0 \leq \epsilon_t \leq 1. \tag{6}$$

With $\mathcal{P}_k = \{\boldsymbol{x} \mid P_k\boldsymbol{x} + \boldsymbol{p_k} \geq \boldsymbol{0}\}$ and $\mathcal{Q}_t = \{\boldsymbol{y} = (\boldsymbol{x}, \boldsymbol{x}') \mid Q_t\boldsymbol{y} + \boldsymbol{q_t} \geq \boldsymbol{0}\}$, we can prove (using the Farkas lemma) that the inequalities (2) and (6) are respectively equivalent to (7) and (8) :

$$\exists \boldsymbol{\lambda}_k \in (\mathbb{R}^+)^n, \lambda_k^0 \in \mathbb{R}^+ \text{ s.t. } \sigma(k, \boldsymbol{x}) = \boldsymbol{\lambda}_k.(P_k\boldsymbol{x} + \boldsymbol{p_k}) + \lambda_k^0 \tag{7}$$

$$\exists \boldsymbol{\mu}_t \in (\mathbb{R}^+)^n, \mu_t^0 \in \mathbb{R}^+ \text{ s.t. } \Delta_t(\sigma, \boldsymbol{x}, \boldsymbol{x}') - \epsilon_t = \boldsymbol{\mu}_t.(Q_t\boldsymbol{y} + \boldsymbol{q_t}) + \mu_t^0 \tag{8}$$

Finally, by substituting (7) in (8), we obtain that "(2) and (6) for as many ϵ_t as possible" is equivalent to a set of linear inequalities. Thus, the computation of a monodimensional affine weak ranking function is done by solving a linear programming instance.

Example 7.

Aspic provides the following invariants for the automaton of Example 4 (the initial values are denoted by N_0, x_0 and y_0) :

$$\begin{aligned} P_{\text{init}} &= \{1 \leq N_0 = N, y = y_0, x = x_0\} \\ P_{k_1} &= \{x \leq 2N, y \leq 2N, N \leq y, \\ &\quad N - 1 \leq x, 1 \leq N = N_0\} \\ P_{\text{end}} &= \{x < N, y \leq 2N, N \leq y, \\ &\quad N - 1 \leq x, 1 \leq N = N_0\} \end{aligned}$$

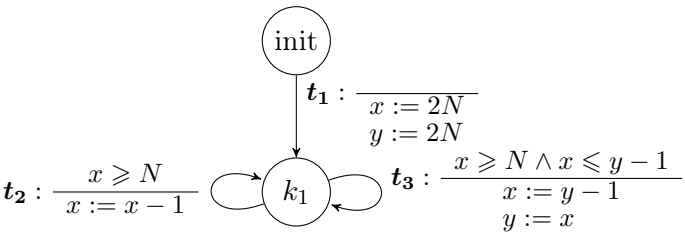

Figure 7 – Monodimensional example

For $i \geq 1$ and $k \in \mathcal{K}$, we denote λ_k^i the i^{th} coordinate of $\boldsymbol{\lambda}_k$, same for $\boldsymbol{\mu}$. Also, $\boldsymbol{x}$ stands for the vector (x, x_0, y, y_0, N, N_0). The subsystem (7) is obtained by applying Farkas lemma to P_{init}, P_{k_1} and P_{end} (here, with no simplification):

$$\begin{aligned} \rho(\text{init}, \boldsymbol{x}) = \lambda_{\text{init}}^0 &+ \lambda_{\text{init}}^1(N-1) + \lambda_{\text{init}}^2(N_0-1) + \lambda_{\text{init}}^3(N_0-N) + \lambda_{\text{init}}^4(N-N_0) \\ &+ \lambda_{\text{init}}^5(y_0-y) + \lambda_{\text{init}}^6(y-y_0) + \lambda_{\text{init}}^7(x_0-x) + \lambda_{\text{init}}^8(x-x_0) \\ &\text{with } \lambda_{\text{init}}^i \geq 0 \text{ for all } i \in [0,8] \end{aligned} \tag{7i}$$

$$\begin{aligned} \rho(k_1, \boldsymbol{x}) = \lambda_{k_1}^0 &+ \lambda_{k_1}^1(2N-x) + \lambda_{k_1}^2(2N-y) + \lambda_{k_1}^3(N-1) + \lambda_{k_1}^4(y-N) \\ &+ \lambda_{k_1}^5(x-N-1) + \lambda_{k_1}^6(N_0-N) + \lambda_{k_1}^7(N-N_0) \\ &\text{with } \lambda_{k_1}^i \geq 0 \text{ for all } i \in [0,7] \end{aligned} \tag{7ii}$$

$$\begin{aligned} \rho(\text{end}, \boldsymbol{x}) = \lambda_{\text{end}}^0 &+ \lambda_{\text{end}}^1(N-x) + \lambda_{\text{end}}^2(2N-y) + \lambda_{\text{end}}^3(N-1) + \lambda_{\text{end}}^4(y-N) \\ &+ \lambda_{\text{end}}^5(x-N-1) + \lambda_{\text{end}}^6(N_0-N) + \lambda_{\text{end}}^7(N-N_0) \\ &\text{with } \lambda_{\text{end}}^i \geq 0 \text{ for all } i \in [0,7] \end{aligned} \tag{7iii}$$

To get the subsystem (8), we first compute $\mathcal{Q}_t$ (here in logical form) :

- $\mathcal{Q}_{t_1} = \boldsymbol{x} \in P_{\text{init}} \wedge x' = 2N \wedge y' = 2N \wedge N' = N$
- $\mathcal{Q}_{t_2} = \boldsymbol{x} \in P_{k_1} \wedge N \leq x \wedge x' = x - 1 \wedge y' = y \wedge N' = N$
- $\mathcal{Q}_{t_3} = \boldsymbol{x} \in P_{k_1} \wedge N \leq x \wedge x < y \wedge x' = y - 1 \wedge y' = x \wedge N' = N$

And, finally, we obtain:

$$\begin{aligned} \rho(\text{init}, \boldsymbol{x}) - \rho(k_1, \boldsymbol{x}') - \epsilon_{t_1} = \mu_{t_1}^0 &+ \mu_{t_1}^1(N_0-1) + \mu_{t_1}^2(N-N_0) + \mu_{t_1}^3(N_0-N) + \dots \\ &+ \mu_{t_1}^7(x'-2N) + \mu_{t_1}^8(2N-x') + \mu_{t_1}^9(y'-2N) \\ &+ \mu_{t_1}^{10}(2N-y') + \dots \text{ with } \mu_{t_1}^i \geq 0 \text{ for all } i \end{aligned} \tag{8i}$$

$$\rho(k_1, \boldsymbol{x}) - \rho(k_1, \boldsymbol{x}') - \epsilon_{t_2} = \dots \tag{8ii}$$

$$\rho(k_1, \boldsymbol{x}) - \rho(k_1, \boldsymbol{x}') - \epsilon_{t_3} = \dots \tag{8iii}$$

In the expressions coming from (8), we replace the expressions involving ρ by the values obtained in (7). We obtain a conjunction of conditions involving the different $\boldsymbol{\lambda}_k$ and $\boldsymbol{\mu}_k$. The solver finally produces:

- $\rho(\text{init}, \boldsymbol{x}) = 2N_0 + 3$
- $\rho(k_1, \boldsymbol{x}) = 2 + x + y - 2N$
- $\rho(\text{end}, \boldsymbol{x}) = 0$

which means that the loop terminates in at most $2N_0 + 3$ steps (including at most $2N_0 + 1$ iterations of the while loop). The quantity $i = 2 + x + y - 2N$, which is automatically extracted, can be viewed as a kind of counter for the while loop. ■

As shown in the example above, the value of the ranking function at the control point k_{init} gives an upper bound of the number of transitions taken, i.e. an upper bound of the worst case complexity of the algorithm. The ranking function being affine and monodimensional, only the termination of programs of linear complexity can be proven.

In order to prove the termination of more complex programs with affine ranking functions, we need to find multidimensional ranking functions.

3.2 Multidimensional algorithm

To compute a mutidimensional ranking function, Alias et al. [1] apply a greedy algorithm which maximizes the number of transitions satisfied by each dimensions of the ranking function :

Algorithm 1 Multidimensional algorithm

1: $i = 0$; $T = \mathcal{T}$; ▷ Initialize T to the set of all transitions
2: **while** T is not empty **do**
3: Find a monodimensional affine function σ and values ϵ_t such that all inequalities (2) and (6) are satisfied and as many ϵ_t as possible are equal to 1; ▷ This means maximizing $\sum_{t \in T} \epsilon_t$
4: Let $\rho_i = \sigma$; $i = i + 1$; ▷ σ defines the i-th component of ρ
5: If no transition t with $\epsilon_t = 1$, **return** false ▷ No multidimensional affine ranking.
6: Remove from T all transitions t such that $\epsilon_t = 1$; ▷ The transitions have level i
7: **end while**;
8: $d = i$; **return** true; ▷ There is a d-dimensional ranking

Example 8.

Consider the automaton of Example 6 with the $N \geq 0$ in control point init. ASPIC finds the following invariants :

$$\begin{aligned} P_{\text{init}} &= \{0 \leq N = N_0, j = j_0, i = i_0\} \\ P_{k_1} &= \{0 \leq N = N_0, 0 \leq i \leq N\} \\ P_{k_2} &= \{0 \leq N = N_0, 1 \leq i \leq N, \\ &\qquad 0 \leq j \leq N\} \end{aligned}$$

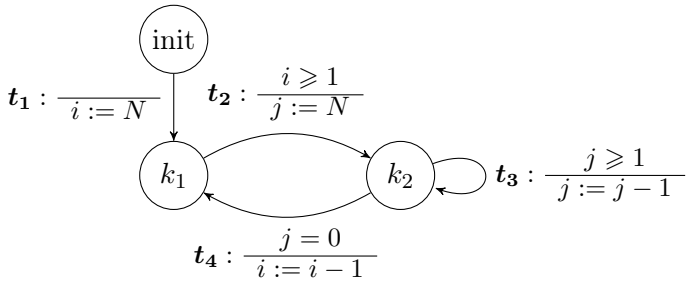

Figure 8 – Multidimensional example

Solving the system while maximizing $\epsilon_{t_1} + \epsilon_{t_2} + \epsilon_{t_3} + \epsilon_{t_4}$ leads to $\epsilon_{t_1} = \epsilon_{t_2} = \epsilon_{t_4} = 1$ and $\epsilon_{t_3} = 0$. The corresponding function (first dimension of the ranking) is

$$\begin{cases} \rho(\text{init}, x)[0] &= 2N + 1 \\ \rho(k_1, x)[0] &= 2i \\ \rho(k_2, x)[0] &= 2i - 1 \end{cases}$$

We keep the transition t_3 for the next dimension, and we get $\epsilon_{t_3} = 1$ with $\rho(k_2, x)[1] = j$. The complete ranking function is thus

$$\begin{cases} \rho(\text{init}, x) &= 2N + 1 \\ \rho(k_1, x) &= 2i \\ \rho(k_2, x) &= (2i - 1, j) \end{cases}$$

Note that, for init and k_1 , the ranking is only one-dimensional. If a globally 2D ranking function is desired, it can be extended arbitrarily, for example as

$$\begin{cases} \rho(\text{init}, x) &= (2N+1, 0) \\ \rho(k_1, x) &= (2i, 0) \\ \rho(k_2, x) &= (2i-1, j) \end{cases}$$

■

Some benchmarks can be found at `http://compsys-tools.ens-lyon.fr/wtc/index.html`.

Yet, as the underlying LP instances consider the integer interpreted automaton globally, there is a challenging scaling issue. This algorithm cannot prove the termination of an iterative implementation of the mergesort. Some work has been done in [3] to enhance the scalability of the algorithm on C programs with while loops, by ignoring the statements that don't have an impact on the loop condition, and by proving the termination of each loop separately when it is possible. However, in order to have a significant improvement of the scalability, we have to modify the algorithm so that it does not consider the whole automaton.

4 A proposition for improvement

In order to improve the scalability of this algorithm, Laure Gonnord and David Monniaux proposed in an unpublished work to solve the LP instances incrementally by using SMT (satisfiability modulo theories)-queries for selecting "pertinent" constraints to be added to the current system-to-solve. I will first present some theoretical results, then a non-terminating algorithm to compute monodimensional ranking functions, a corrected version, and finally an algorithm to compute multidimensional ranking functions.

The proposed algorithm was almost working. I had to make a few corrections and adapt its termination and correctness proofs to the corrections I have made.

4.1 Notations and propositions

This part is a summary of a larger research report which has not been published yet.

The algorithm presented in this section uses the same approach as the previous one : it greedily computes a strict multidimensional ranking function by adding monodimensional ranking functions as new dimensions until the ranking is strict. We will here give a few properties on monodimensional ranking functions which will enable us to improve the algorithm.

From now on, we consider an affine interpreted automaton $(\{k\}, n, k, \mathcal{T})$ with only one control point.

Remark 1. This hypothesis simplifies the problem without loss of generality, in so far as the algorithm can be adapted to work with any automaton as in [6].

We denote $\mathcal{I} = \mathcal{P}_k$ an invariant of the control point and τ the *transition relation*, with

$$\tau = \bigcup_{(k,g,a,k)\in\mathcal{T}} \{(\boldsymbol{x}, \boldsymbol{x}') \mid g(\boldsymbol{x}) = \text{true} \wedge \boldsymbol{x}' = a(\boldsymbol{x})\}$$

We will also use the following notations :

- If σ is an affine monodimensional weak ranking function, $\sigma(\boldsymbol{x}) = \boldsymbol{l}.\boldsymbol{x} - c$
- $\mathcal{I} = \{\boldsymbol{x}, \bigwedge_{i=1}^{m} \boldsymbol{l}_i.\boldsymbol{x} \geq c_i\}$ and $\mathcal{L}_\mathcal{I} = \{\boldsymbol{l}_i, i \in [1, m]\}$

Example 9 (Generators for polyhedral invariants). Let us consider the following automaton :

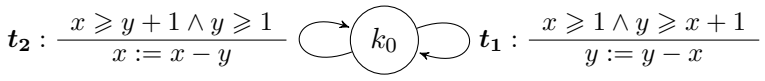

Figure 9

The associated transition relation is :

$$\tau = \Big\{(x, y, x', y'),\ \big(x \geq 1 \wedge y \geq x + 1 \wedge x' = x \wedge y' = y - x\big) \\ \vee \big(x \geq y + 1 \wedge y \geq 1 \wedge x' = x - y \wedge y' = y\big) \Big\}$$

An invariant generator gives the following inductive invariant for $\mathcal{I}$:

$$\mathcal{I} = \{0 \leq x, 0 \leq y\}$$

From these invariants we compute the $\boldsymbol{l}_i$s :

$$\boldsymbol{l}_1 = \begin{pmatrix}1\\0\end{pmatrix}, \boldsymbol{l}_2 = \begin{pmatrix}0\\1\end{pmatrix}$$

■

Proposition 2. *σ is a monodimensional weak ranking function iff*

$$\forall \boldsymbol{x}, \boldsymbol{x'}\ \boldsymbol{x} \in \mathcal{I} \wedge (\boldsymbol{x}, \boldsymbol{x'}) \in \tau \implies \boldsymbol{l}.(\boldsymbol{x} - \boldsymbol{x'}) \geq 0 \tag{9}$$

and

$$\exists \lambda_1 \geq 0, \ldots, \lambda_m \geq 0,\ \boldsymbol{l} = \sum_{i=1}^{m} \lambda_i \boldsymbol{l_i} \tag{10}$$

Condition 10 means that we look for ranking function that is a linear combination of the invariants.

Proof. We can first notice that $\tau \cap \mathcal{I} \times \mathbb{Q} = \bigcup_{t \in \mathcal{T}} \mathcal{Q}_t$, thus $\sigma : \mathbb{Q}^n \to \mathbb{Q}$ is a monodimensional weak ranking function iff

$$\forall \boldsymbol{x}, \boldsymbol{x'} \quad , \quad \boldsymbol{x} \in \mathcal{I} \wedge (\boldsymbol{x}, \boldsymbol{x'}) \in \tau \implies \sigma(\boldsymbol{x}) \geq \sigma(\boldsymbol{x'}) \tag{11}$$

$$\forall \boldsymbol{x} \in \mathcal{I} \quad , \quad \sigma(\boldsymbol{x}) \geq 0 \tag{12}$$

Condition 9 is then obtained by rewriting condition 11 using linearity.

Secondly, the Farkas Lemma applied on the polyhedron I where inequality scheme 12 holds, gives :

$$\exists \lambda_1 \geq 0, \ldots, \lambda_m \geq 0,\ \boldsymbol{l} = \sum_{i=1}^{m} \lambda_i \boldsymbol{l}_i \wedge C \leq \sum_{i=1}^{m} \lambda_i c_i$$

Once the condition $\boldsymbol{l} = \sum_{i=1}^{m} \lambda_i \boldsymbol{l}_i$ is met then an appropriate choice of C can always be made, thus condition 10. □

Let $\mathcal{P}_{\mathcal{I},\tau} = \{\boldsymbol{x} - \boldsymbol{x'} \mid \boldsymbol{x} \in \mathcal{I} \wedge (\boldsymbol{x}, \boldsymbol{x'}) \in \tau\}$ be the set of all reachable 1-step differences. Let us reexamine the condition that $\boldsymbol{l}.\boldsymbol{p} \geq 0$ for all $p \in \mathcal{P}_{\mathcal{I},\tau}$. Remark that this condition is left unchanged if we replace $\mathcal{P}_{\mathcal{I},\tau}$ by its convex hull. Because $\mathcal{I}$ and τ are (finite unions of) convex closed polyhedra, so is $\mathcal{P}_{\mathcal{I},\tau}$, and thus its convex hull $\mathcal{P}_{\mathcal{H}}$ is a closed convex polyhedron. Thanks to Equation 1, we can deduce that there is a finite set $\mathcal{V} = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m\}$ of generators of $\mathcal{P}_{\mathcal{I},\tau}$ such that Condition 9 is equivalent to $\forall 1 \leq i \leq m,\ \boldsymbol{l}.\boldsymbol{v}_i \geq 0$.

As we did for the previous algorithm, we are interested by monodimensional weak ranking functions σ which are "as strict as possible", i.e. that satisfy $\boldsymbol{l}.\boldsymbol{v} > 0$ for as many $v \in \mathcal{V}$ as possible. Let $\pi_{\mathcal{V}}(\boldsymbol{l}) = \{\boldsymbol{v} \in \mathcal{V} \mid \boldsymbol{v}.\boldsymbol{l} > 0\}$; $\pi_{\mathcal{V}}$ measures the distance to the ideal. We would like $\boldsymbol{l}$ of maximal $|\pi_{\mathcal{V}}(\boldsymbol{l})|$.

Proposition 3. *Given $\mathcal{V} = (\boldsymbol{v}_1, \ldots \boldsymbol{v}_k)$ a set of vectors and $\boldsymbol{l}$ a vector, $\pi_{\mathcal{V}}(\boldsymbol{l})$ is maximal with respect to cardinality if and only if it is maximal with respect to inclusion, i.e.*

$$\forall \boldsymbol{l'}, |\pi_{\mathcal{V}}(\boldsymbol{l'})| \leq |\pi_{\mathcal{V}}(\boldsymbol{l})| \Leftrightarrow \forall \boldsymbol{l'}, \pi_{\mathcal{V}}(\boldsymbol{l'}) \subseteq \pi_{\mathcal{V}}(\boldsymbol{l})$$

Proof. It is obvious that $\forall \boldsymbol{l'}, \pi_{\mathcal{V}}(\boldsymbol{l'}) \subseteq \pi_{\mathcal{V}}(\boldsymbol{l}) \Rightarrow \forall \boldsymbol{l'}, |\pi_{\mathcal{V}}(\boldsymbol{l'})| \leq |\pi_{\mathcal{V}}(\boldsymbol{l})|$. Let us now prove the other implication.

Let $\boldsymbol{l}$ be a vector such that $\forall \boldsymbol{l'}, |\pi_{\mathcal{V}}(\boldsymbol{l'})| \leq |\pi_{\mathcal{V}}(\boldsymbol{l})|$ and $\boldsymbol{l'}$ a vector. Let us suppose that $\pi_{\mathcal{V}}(\boldsymbol{l'}) \not\subseteq \pi_{\mathcal{V}}(\boldsymbol{l})$. Then $\pi_{\mathcal{V}}(\boldsymbol{l'}) \backslash \pi_{\mathcal{V}}(\boldsymbol{l})$ is not empty. Since $\pi_{\mathcal{V}}(\boldsymbol{l}) \cup \pi_{\mathcal{V}}(\boldsymbol{l'}) \subseteq \pi_{\mathcal{V}}(\boldsymbol{l} + \boldsymbol{l'})$, $|\pi_{\mathcal{V}}(\boldsymbol{l} + \boldsymbol{l'})| \geq |\pi_{\mathcal{V}}(\boldsymbol{l}) \cup \pi_{\mathcal{V}}(\boldsymbol{l'})| = |\pi_{\mathcal{V}}(\boldsymbol{l'})| + |\pi_{\mathcal{V}}(\boldsymbol{l'}) \backslash \pi_{\mathcal{V}}(\boldsymbol{l})| > |\pi_{\mathcal{V}}(\boldsymbol{l'})|$, which is absurd. Thus, $\forall \boldsymbol{l'}, |\pi_{\mathcal{V}}(\boldsymbol{l'})| \leq |\pi_{\mathcal{V}}(\boldsymbol{l})|$. □

4.2 First monodimensional algorithm

Definition 6. Given $\mathcal{V} = (\boldsymbol{v}_1, \dots \boldsymbol{v}_k)$ and $\mathcal{L} = (\boldsymbol{l}_1, \dots \boldsymbol{l}_m)$, we denote by $LP(\mathcal{V}, \mathcal{L})$ the following Linear Programming instance where λ_i and δ_i are the unknowns:

$$\begin{cases} \textit{Maximize} \sum_i \delta_i \text{ s.t.} & \\ \lambda_1, \dots, \lambda_m \geq 0 & \\ 0 \leq \delta_j \leq 1 & \text{for all } 1 \leq j \leq N \\ \sum_{i=1}^m \lambda_i (\boldsymbol{v}_j . \boldsymbol{l}_i) \geq \delta_j & \text{for all } 1 \leq j \leq N \end{cases} \tag{13}$$

In the following, we denote as $\boldsymbol{\lambda}$ (resp. $\boldsymbol{\delta}$) the vector with λ_i (resp. δ_i) components.

Proposition 4. *Let $\mathcal{V} = (\boldsymbol{v}_1, \dots \boldsymbol{v}_k)$ and $\mathcal{L} = (\boldsymbol{l}_1, \dots \boldsymbol{l}_m)$. Then :*

- *$LP(\mathcal{V}, \mathcal{L})$ is always feasible.*
- *$LP(\mathcal{V}, \mathcal{L})$ gives λ_is such that $\boldsymbol{l} = \sum_{i=1}^m \lambda_i \boldsymbol{l}_i$ is a weak ranking function of maximal $\pi_{\mathcal{V}}$.*

Proof. Let us start by noticing that $(\boldsymbol{\lambda}, \boldsymbol{\delta}) = (\boldsymbol{0}, \boldsymbol{0})$ is a solution of the inequalities, and $\sum_i \delta_i \leq N$, thus the set $\{\sum_i \delta_i$ s.t. $(\boldsymbol{\lambda}, \boldsymbol{\delta})$ is a solution of the above inequalities$\}$ has a least upper bound. Moreover, in the optimal result of $LP(\mathcal{V}, \mathcal{L})$, each δ_j is either 0 or 1 : if $0 < \delta_j < 1$, then by scaling up $\boldsymbol{l}$ by a factor $1/\delta_j$ we obtain a solution $\boldsymbol{l'}$ with $\delta'_j = 1$. Therefore, the least upper bound is a maximum. It ensues that $LP(\mathcal{V}, \mathcal{L})$ is always feasible.

$\delta_i = 1$ iff $\boldsymbol{v}_i \in \pi_{\mathcal{V}}(l)$, thus $|\pi_{\mathcal{V}}(l)| = \sum_i \delta_i$. It implies that $|\pi_{\mathcal{V}}(l)|$ is maximal □

The difficulty is that computing this set $\mathcal{V}$ may be expensive. Obviously, we could compute a disjunctive normal form (DNF) for $\mathcal{I} \wedge \tau$, each disjunct denoting a convex polyhedron, compute their generators (x, x') and collect all resulting $x - x'$; yet computing the DNF has exponential cost and the full DNF may be unnecessary.

Let us propose instead an incremental algorithm (somewhat wrong, but it will be corrected later): the idea is to incrementally augment a set $\mathcal{C}$ with elements of $\mathcal{P}_{\mathcal{H}}$ that negate the fact that the current $\boldsymbol{l}$ could be a strict ranking function. As soon as such an element $(\boldsymbol{x}, \boldsymbol{x'})$ is found (by an SMT-query), the set $\mathcal{C}$ is used to compute a new ranking-function candidate. The algorithm is described in Alg.2.

Algorithm 2 Initial Algorithm

Require: $\mathcal{I}$ the invariant polyhedra and τ the transition function
Ensure: When it terminates, returns $\boldsymbol{l} = \sum_{i=1}^m \lambda_i \boldsymbol{l}_i$ a weak ranking function of maximal π and a boolean which is true iff $\boldsymbol{l}$ is a strict ranking function

```
C ← ∅
λ ← 0
l ← 0
finished ← false
while not(finished) and ( I ∧ τ ∧ (x − x').l ≤ 0 satisfiable ) do
    (x, x') ← a model for the above SMT test
    C ← C ∪ {x − x'}
    (λ, δ) ← LP(C, L_I)
    if λ = 0 then
        finished ← true
    else
        l ← Σ_{i=1}^m λ_i l_i
    end if
end while
Return (l, ⋀_i δ_i = 1).
```

Proof of correctness [Monniaux/Seguinot]. It is easy to see that, if this algorithm terminates, then $\boldsymbol{l}$ is a weak ranking function (possibly null). When the algorithm terminates, either $finished$ is true,

or $\mathcal{I} \wedge \tau \wedge (\boldsymbol{x} - \boldsymbol{x'}).\boldsymbol{l} \leq 0$ is not satisfiable. In the first case, proposition 4 shows that $\boldsymbol{l} = \boldsymbol{0}$ is a weak ranking function of maximal π_C, which means that $\boldsymbol{0}$ is the only weak ranking function on $\mathcal{C}$. Therefore it is a ranking function of maximal $\pi_{\mathcal{V}}$, because $\mathcal{C} \subseteq \mathcal{V}$. In the second case, let $\boldsymbol{l'}$ be a weak ranking function of maximal $\pi_{\mathcal{C}}$. Then property 3 gives $\pi_{\mathcal{C}}(\boldsymbol{l}) \subseteq \pi_{\mathcal{C}}(\boldsymbol{l'})$. If $\boldsymbol{u} \in \pi_{\mathcal{C}}(\boldsymbol{l'}) \setminus \pi_{\mathcal{C}}(\boldsymbol{l})$, then either $\boldsymbol{u} \in \mathcal{C}$, which means that $\pi_{\mathcal{C}}(\boldsymbol{l})$ is not maximal on $\mathcal{C}$ which is a contradiction, or $\boldsymbol{u} \notin \mathcal{C}$ and $\boldsymbol{u} \cdot \boldsymbol{l} \leq 0$, so $\mathcal{I} \wedge \tau \wedge (\boldsymbol{x} - \boldsymbol{x'}).\boldsymbol{l} \leq 0$ is satisfiable, which is absurd too. It ensues that $\pi_{\mathcal{C}}(\boldsymbol{l'}) \subset \pi_{\mathcal{C}}(\boldsymbol{l})$, i.e. $\boldsymbol{l}$ is of maximal $\pi_{\mathcal{C}}$.

Moreover, if $finished$ is true, then $\boldsymbol{\lambda} = \boldsymbol{0}$ which implies $\boldsymbol{\delta} = \boldsymbol{0}$, thus $\bigwedge_i \delta_i = 0$, which is coherent with the fact that $\boldsymbol{l}$ is not strict. Else, $\boldsymbol{l}$ is strict iff $\pi_{\mathcal{V}}(\boldsymbol{l}) = \mathcal{V}$ i.e. iff $\bigwedge_i \delta_i = 1$. □

Example 10. On Example 9, we already computed the generators $\boldsymbol{l}_i$ for $\mathcal{I}$. These vectors are given as input to Algorithm 2, as well as the transition function :

$$\tau = \Big\{(x, y, x', y'),\ \big(x \geq 1 \wedge y \geq x + 1 \wedge x' = x \wedge y' = y - x\big) \\ \vee \big(x \geq y + 1 \wedge y \geq 1 \wedge x' = x - y \wedge y' = y\big) \Big\}$$

Recall that we are looking for $\boldsymbol{l}$ as a linear combination of the $\boldsymbol{l}_i$s.

- Step 1. beginning with $\boldsymbol{l} = \boldsymbol{0}$, we SMT-solve the constraint $\mathcal{I} \wedge \tau \wedge 0 \leq 0$. We are given a first model $x = 1, x' = 1, y = 2, y' = 1$, thus $\mathcal{C} \leftarrow \{\boldsymbol{c}_1\}$ where $\boldsymbol{c}_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$. Then, an LP-solver is given the following LP problem :
$$\begin{cases} \textit{Maximize } \delta_1 \text{ s.t.} \\ \lambda_1, \lambda_2 \geq 0 \\ 0 \leq \delta_1 \leq 1 \\ \lambda_2 \geq \delta_1 \end{cases}$$
(The last contraint comes from the scalar products of $\boldsymbol{c}_1$ with all $\boldsymbol{l}_i$.). The solver gives $\lambda_1 = 0$ and $\lambda_2 = 1$, . Then $\boldsymbol{l} = \boldsymbol{l}_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$.

- Step 2. The new SMT-constraint is now $\mathcal{I} \wedge \tau \wedge \begin{pmatrix} x' - x \\ y - y' \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} \leq 0$. The solver gives us the model $x = 2, x' = 1, y = 1, y' = 1$, thus $\mathcal{C} \leftarrow \{\boldsymbol{c}_1, \boldsymbol{c}_2\}$ where $\boldsymbol{c}_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$. The new LP instance is thus now :
$$\begin{cases} \textit{Maximize } \delta_1 + \delta_2 \text{ s.t.} \\ \lambda_1, \lambda_2 \geq 0 \\ 0 \leq \delta_1, \delta_2 \leq 1 \\ \lambda_2 \geq \delta_1 \\ \lambda_1 \geq \delta_2 \end{cases}$$
whose optimal is $\lambda_1 = \lambda_2 = 1$ with $\delta_1 = \delta_2 = 1$ Then $\boldsymbol{l} = \boldsymbol{l}_1 + \boldsymbol{l}_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$.

- Step 3. The SMT-constraint is now $\mathcal{I} \wedge \tau \wedge x' - x + y' - y \leq 0$, and the SMT solver gives UNSAT.

- The function returns $(\boldsymbol{l} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \mathsf{true})$.

To compute the final ranking function $\sigma = \boldsymbol{l} \cdot \boldsymbol{x} - c = x + y - c$, we have to compute a suitable c. Combining the $\mathcal{I}$ constraints using the last $\lambda_i s$ give the constraint $x + y \geq 0$, thus $\sigma = x + y$ is a strict ranking function for $(\tau, \mathcal{I})$. ■

Unfortunately, this simple algorithm suffers from two problems (noted by D.Monniaux and L. Gonnord in their report) which may prevent its termination.

Non finiteness of the smt models First, termination is guaranteed only if the models provided by the SMT tests come from a finite set (then the number of iterations is bounded by the cardinal of that set). Depending on the implementation of the SMT-solver, it may be the case that all models $(\boldsymbol{x}, \boldsymbol{x'})$ would be vertices constructed by intersection of the linear atomic constraints in $\mathcal{I} \wedge \tau$, of which there are finitely (exponentially) many; but this will also produce vertices that do not lie on the boundary of the convex hull $\mathcal{P}_{\mathcal{H}}$, resulting in unoptimally tight constraints. A perhaps better option is to impose that the model for the SMT-test should minimize $(\boldsymbol{x} - \boldsymbol{x'}) \cdot \boldsymbol{l}$, which advocates for the use of state-of-the-art "optimization modulo theory" tools, like OptiMathSAT [8].

If $\inf(\boldsymbol{x} - \boldsymbol{x'}) \cdot \boldsymbol{l} = -\infty$, i.e. if $\mathcal{P}_{\mathcal{H}}$ has a ray generator $\boldsymbol{r}$ such that $\boldsymbol{r} \cdot \boldsymbol{l} < 0$, the SMT-solver used (OptiMathSAT [8]), returns a model which satisfies $(\boldsymbol{x} - \boldsymbol{x'}) \cdot \boldsymbol{l} = -K$, where K is a "big" constant. I saw that this prevents the algorithm from terminating, since the problem $\mathcal{I} \wedge \tau \wedge (\boldsymbol{x} - \boldsymbol{x'}).\boldsymbol{l} \leq 0$ is always satisfiable ($\boldsymbol{u} + \beta \boldsymbol{r}$, with $\beta \geq -\frac{\boldsymbol{u} \cdot \boldsymbol{l}}{\boldsymbol{r} \cdot \boldsymbol{l}}$, is a model).

Example 11.

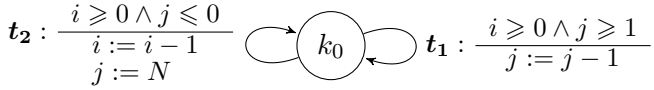

Figure 10 – Non-terminating example

For example, the algorithm won't terminate on the automaton of Example 11. Indeed, on the transition $\boldsymbol{t_2}$, $j - j' \leq -N$, and there is no constraint on N. Thus, if we denote $\boldsymbol{x} = (i, j, N)$ the variable vector, $\boldsymbol{r} = (0, -1, 0)$ is a ray generator of $\mathcal{P}_{\mathcal{H}}$. We can see that as long as $\boldsymbol{l} \cdot \boldsymbol{r} < 0$, the SMT solver can return an infinite number of models as $\mathcal{I} \wedge \tau \wedge (\boldsymbol{x} - \boldsymbol{x'}).\boldsymbol{l} \leq 0$ is always satisfiable. ■

Non termination due to weak ranking functions Second, even if the first issue is fixed, the above algorithm terminates only if there is a strict ranking function ($\boldsymbol{l}.\boldsymbol{v} > 0$ for all $v \in \mathcal{V}$). Indeed, termination is ensured by the algorithm never choosing twice the same $(\boldsymbol{x}, \boldsymbol{x'})$, which is the case if there exists a (weak) ranking function such that $\boldsymbol{l}.(\boldsymbol{x} - \boldsymbol{x'}) > 0$. But what if the SMT-solver picks $(\boldsymbol{x}, \boldsymbol{x'})$ such that *all* weak ranking functions $\boldsymbol{l}$ satisfy $\boldsymbol{l}.(\boldsymbol{x} - \boldsymbol{x'}) = 0$? In this case, the algorithm may not terminate, always picking the same $(\boldsymbol{x}, \boldsymbol{x'})$.

4.3 Corrected monodimensional algorithm

Non termination due to weak ranking functions We can notice that the set $\{\boldsymbol{u} \mid \boldsymbol{u} \cdot \boldsymbol{l} = 0 \ \forall \text{ ranking function } \boldsymbol{l}\}$ is a linear subspace of $\mathbb{Q}^n$. To prevent elements from this subspace to be returned by the SMT-solver, we maintain a free family $\{\boldsymbol{B}_1, \ldots, \boldsymbol{B}_d\} \subseteq \mathbb{Q}^n$ (initially empty), such that we search for solutions $\boldsymbol{l}$ such that $\boldsymbol{B}_i.\boldsymbol{l} = 0$ for all $1 \leq i \leq d$. Every time a new model $(\boldsymbol{x}, \boldsymbol{x'})$ is found, a vector $\boldsymbol{u} = \boldsymbol{x} - \boldsymbol{x'}$ is added to $\mathcal{C}$, and a new optimal solution $(\boldsymbol{l}, \boldsymbol{\delta})$ is computed, we check whether $\delta_{\boldsymbol{u}}$ is 0 or 1. If it is 0, which means that *all* weak ranking functions will satisfy $\boldsymbol{l}.\boldsymbol{u} = 0$, thus $\boldsymbol{u}$ is added to $\mathcal{B}$. We then add the constraint $\boldsymbol{u} \notin \text{Span}(\mathcal{B})$ to the SMT-test. The computation of this constraint can be implemented as either:

1. Complete $\mathcal{B}$ into a basis $(\mathcal{B}, \mathcal{B}')$ of $\mathbb{Q}^n$, then

$$\boldsymbol{u} \notin \text{Span}(\mathcal{B}) \Leftrightarrow \boldsymbol{u} = \sum_{\boldsymbol{v}_i \in B} \beta_i \boldsymbol{v}_i + \sum_{\boldsymbol{v}_i \in B'} \gamma_i \boldsymbol{v}_i \wedge \bigwedge_i \gamma_i \neq 0$$

2. Compute a basis $(\boldsymbol{w}_j)$ of the orthogonal of the vector space generated by $\mathcal{B}$, then

$$\boldsymbol{u} \notin \text{Span}(\mathcal{B}) \Leftrightarrow \bigvee_i u \cdot \boldsymbol{w}_i \neq 0$$

I have implemented it with the first method, by maintaining the matrix representing the basis $\mathcal{B}$ in row echelon form (which can be obtained by computing the Gauss-Jordan elimination of the matrix). This is done with Algorithm 3.

Algorithm 3 AvoidSpace

Require: $\boldsymbol{u}$ and $\mathcal{B}$
  $\mathcal{B}' \leftarrow \emptyset$
  $i \leftarrow 1$
  $j \leftarrow 2$
  while $j \leq m$ **do**
    if $\boldsymbol{B}_i[j] \neq 0$ **then**
      $\mathcal{B}' \leftarrow \mathcal{B}' \cup \{\boldsymbol{e_i}\}$
    else
      $i \leftarrow i + 1$
    end if
    $j \leftarrow j + 1$
  end while
  for $i = j \to n$ **do**
    $\mathcal{B}' \leftarrow \mathcal{B}' \cup \{\boldsymbol{e_i}\}$
  end for
  Return $\boldsymbol{u} = \sum_{\boldsymbol{v}_i \in B} \beta_i \boldsymbol{v}_i + \sum_{\boldsymbol{v}_i \in B'} \gamma_i \boldsymbol{v}_i \wedge \bigwedge_i \gamma_i \neq 0$

Non finiteness of the smt models In the light of Example 11, I suggested that adding the constraints $\boldsymbol{l} \cdot \boldsymbol{r} \geq 0$ for all ray generator $\boldsymbol{r}$ of $\mathcal{P}_{\mathcal{H}}$ seems to correct the first issue (we will prove later that it does). As we have seen before, computing the generators of $\mathcal{P}_{\mathcal{H}}$ may be expensive. Instead, we maintain a set $\mathcal{R}$ of rays which are generators of the polyhedron $\mathcal{P}_{\mathcal{H}}$, and we define the $LP(\mathcal{C}, \mathcal{L}_{\mathcal{I}}, \mathcal{R})$ instance as :

$$\begin{cases} \textit{Maximize} \sum_i \delta_i \text{ s.t.} & \\ \lambda_1, \ldots, \lambda_m \geq 0 & \\ 0 \leq \delta_j \leq 1 & \text{for all } 1 \leq j \leq N \\ \sum_{i=1}^m \lambda_i (\boldsymbol{v}_j . \boldsymbol{l}_i) \geq \delta_j & \text{for all } 1 \leq j \leq N \\ \sum_{i=1}^m \lambda_i (\boldsymbol{r} . \boldsymbol{l}_i) \geq 0 & \text{for all } \boldsymbol{r} \in \mathcal{R} \end{cases} \tag{14}$$

Proposition 4 is still true, since $\sum_{i=1}^m \lambda_i (\boldsymbol{r} . \boldsymbol{l}_i) \geq 0$ for all $\boldsymbol{r} \in R$ is a necessary condition for $\boldsymbol{l}$ to be a weak ranking function.

We then add the ray generators $\boldsymbol{r}$ incrementally, when the SMT-solver returns a model of the form $\boldsymbol{u} = \cdots + \beta \boldsymbol{r}$. When a model is of this form, $\boldsymbol{u} \cdot \boldsymbol{l} = -K$. We can ensure that there is actually an unbounded component by checking if $\boldsymbol{u} \cdot \boldsymbol{l} = -(K+1)$ is satisfiable. We then compute $\boldsymbol{r}$ and add it to $\mathcal{R}$. This is implemented as follow :

Algorithm 4 UnboundedRay

Require: $\boldsymbol{u}$, $\boldsymbol{l}$ and $\boldsymbol{r}$
  if $\boldsymbol{u} \cdot \boldsymbol{l} = -K$ **then**
    Generalize the model $\boldsymbol{u}$ with a conjunction of equalities P constructed with the saturated inequalities of $I \wedge \tau \wedge \boldsymbol{u} \cdot \boldsymbol{l} \leq 0$
    if $P \wedge \mathit{AvoidSpace}(\boldsymbol{u}, \mathcal{B}) \wedge \boldsymbol{u} \cdot \boldsymbol{l} = -(K+1)$ is satisfiable **then**
      $\boldsymbol{u'} \leftarrow$ a model for $\boldsymbol{u}$ in the above SMT test
      $\boldsymbol{r} \leftarrow (\boldsymbol{u'} - \boldsymbol{u})$
      Return `true`
    end if
  end if
  Return `false`

These remarks and solutions lead to the following algorithm :

Algorithm 5 Modified (terminating) Algorithm MONODIMENSIONAL

Require: $\mathcal{I}$ and τ
  $\mathcal{C} \leftarrow \emptyset$
  $\mathcal{B} \leftarrow \emptyset$
  $\mathcal{R} \leftarrow \emptyset$
  finished $\leftarrow$ false
  $\boldsymbol{r} \leftarrow \mathbf{0}$
  while $\neg$*finished* and $\mathcal{I} \wedge \tau \wedge \boldsymbol{u} = \boldsymbol{x} - \boldsymbol{x'} \wedge \mathit{AvoidSpace}(\boldsymbol{u}, \mathcal{B})$ satisfiable (in the unknowns $\boldsymbol{u}, \boldsymbol{x}, \boldsymbol{x'}$) with minimization for $\boldsymbol{u} \cdot \boldsymbol{l} \leq 0$ **do**
    $\boldsymbol{u} \leftarrow$ a model for $\boldsymbol{u}$ in the above SMT test
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{\boldsymbol{u}\}$
    if $\mathit{UnboundedRay}(\boldsymbol{u}, \boldsymbol{l}, \boldsymbol{r})$ **then** ▷ $\boldsymbol{u} = \cdots + \beta\boldsymbol{r}$, with $\boldsymbol{r}$ a ray generator of $\mathcal{P}_\mathcal{H}$
      $\mathcal{R} \leftarrow \mathcal{R} \cup \{\boldsymbol{r}\}$
    end if
    $(\boldsymbol{\lambda}, \boldsymbol{\delta}) \leftarrow LP(\mathcal{C}, \mathcal{L}_\mathcal{I}, \mathcal{R})$
    if $\boldsymbol{\lambda} = \mathbf{0}$ **then**
      finished $\leftarrow$ true
    else
      $\boldsymbol{l} \leftarrow \sum_{i=1}^{m} \lambda_i \boldsymbol{l}_i$
      if $\delta_{\boldsymbol{u}} = 0$ **then**
        $\mathcal{B} \leftarrow \mathcal{B} \cup \{\boldsymbol{u}\}$
      end if
    end if
  end while
  Return $(\boldsymbol{l}, \bigwedge_i \delta_i = 1 \wedge (\mathcal{I} \wedge \tau \wedge \boldsymbol{x} = \boldsymbol{x}')$ unsatisfiable)

Adding "$(\mathcal{I} \wedge \tau \wedge \boldsymbol{x} = \boldsymbol{x}')$ unsatisfiable" is necessary in the case where there is a vector $\boldsymbol{x}$ such that $(\boldsymbol{x}, \boldsymbol{x}) \in \tau$. Indeed, the constraint $\mathit{AvoidSpace}(\boldsymbol{u}, \mathcal{B})$ prevents the SMT-solver to return the model $(\boldsymbol{x}, \boldsymbol{x})$. Thus, if $\mathcal{B} = \emptyset$ at the end, the algorithm would return $(\boldsymbol{l}, \text{true})$ instead of $(\boldsymbol{l}, \text{false})$.

Proof (termination of Algorithm 5) [Monniaux/Seguinot]. The number of vectors in $\mathcal{B}$ can only increase or stay the same, and it is bounded by n, thus $\mathcal{B}$ is stationary. Once it is stationary, so is the SMT formula (save for the optimization constraint $\boldsymbol{u} \cdot \boldsymbol{l} \leq 0$).

We argue that when $\mathcal{B}$ is stationary, then once $\delta_i = 1$ at one iteration, then it stays so at all future iterations. Suppose the opposite: we had a system $\mathcal{C}$ of constraints (that is, $\bigwedge_i \boldsymbol{c}_i \cdot \boldsymbol{l} \geq 0$) such that $\delta_{i_0} = 1$ at the preceding iteration, and we are adding a new constraint C' ($\boldsymbol{c'} \cdot \boldsymbol{l} \geq 0$). If $LP(\mathcal{C} \cup \{C'\}, \mathcal{L}_\mathcal{I})$ yields $\delta_{i_0} = 0$ for some i_0, this means, by Farkas' lemma, that $-\boldsymbol{c}_{i_0}$ can be expressed as a combination of the other constraints in $\mathcal{C}$ and of C', otherwise said there exist nonnegative λ_i, λ' such that

$$-c_{i_0} = \sum_{i \neq i_0} \lambda_i c_i + \lambda' c' \tag{15}$$

If $\lambda' = 0$, then $\mathcal{C}$ (without C') already implies $\boldsymbol{c}_{i_0} \cdot \boldsymbol{l} \leq 0$ and thus δ_{i_0} is already null at the preceding iteration; contradiction. Therefore, there exists a nonnegative μ_i such that $-\boldsymbol{c'} = \sum_i \mu_i \boldsymbol{c}_i$, otherwise said any solution of $\mathcal{C}$ satisfies $\boldsymbol{c'} \cdot \boldsymbol{l} \leq 0$; then this means, in our algorithm, that $\delta_{\boldsymbol{u}} = 0$, and thus $\mathcal{B}$ is updated; but we have assumed $\mathcal{B}$ is stationary so this cannot occur.

Once $\mathcal{B}$ is stationary, all vectors $\boldsymbol{u}$ in $\mathcal{C}$ either belong to the vector space spanned by $\mathcal{B}$ (case $\delta_u = 0$), either satisfy $\delta_u = 1$. All $\boldsymbol{l}$ obtained thus verify $\boldsymbol{u} \cdot \boldsymbol{l} > 0$ for all $\boldsymbol{u}$ in $\mathcal{C}$ but not in the span of $\mathcal{B}$. A $\boldsymbol{u}$ that was already given as solution by the SMT solver cannot be given again.

Let us now prove that if $\boldsymbol{v}_i$ (resp. $\boldsymbol{r}_i$) are the vertices (resp. ray) generators of $\mathcal{P}_\mathcal{H}$, and $\boldsymbol{v} = \sum_i \alpha_i \boldsymbol{v}_i$, the SMT solver cannot return an infinite number of vertices of the form $\boldsymbol{v} + \sum_i \beta_i \boldsymbol{r}_i$. Indeed, if the SMT solve returns $\boldsymbol{v} + \sum_i \beta_i \boldsymbol{r}_i$ with at least one of the $\beta_i > 0$, then one of the ray generators $\boldsymbol{r}_{i_0}$ (with $\beta_{i_0} > 0$) will be added to $\mathcal{R}$. If it returns $\boldsymbol{u} = \boldsymbol{v} + \sum_i \beta'_i \boldsymbol{r}_i$ after that, then $\beta'_{i_0} = 0$ (or else $\boldsymbol{u} \cdot \boldsymbol{l}$ is not minimal). Thus, only $|\{\boldsymbol{r}_i\}| + 1$ vectors of the form $\boldsymbol{v} + \sum_i \beta_i \boldsymbol{r}_i$ can be returned by the SMT solver.

Since we have assumed that the solutions of the form $\sum_i \alpha_i \boldsymbol{v}_i$ returned by the SMT solver belong to a finite set, and for each of these solutions, only a finite number of solutions with an unbounded component can be returned, the SMT solver yields solution belonging to a finite set of vertices and the termination ensues. □

Lemma 2. *Let $\boldsymbol{l}$ be the weak ranking function produced by the algorithm and $\boldsymbol{l'}$ be another weak ranking function. Then, for all $(\boldsymbol{x}, \boldsymbol{x}') \in \mathcal{I} \wedge \tau$, if $\boldsymbol{l'} \cdot (\boldsymbol{x} - \boldsymbol{x}') > 0$ then $\boldsymbol{l} \cdot (\boldsymbol{x} - \boldsymbol{x}') > 0$.*

Proof. $\boldsymbol{x} - \boldsymbol{x}'$ can be expressed as a linear combination of a subset of generators of $\mathcal{P}_{\mathcal{H}}$: $\boldsymbol{x} - \boldsymbol{x}' = \sum_{\boldsymbol{v} \in \mathcal{V}'} \alpha_{\boldsymbol{v}} \boldsymbol{v}$ with $\sum_{\boldsymbol{v} \in \mathcal{V}'} \alpha_{\boldsymbol{v}} = 1$ and for all $\boldsymbol{v} \in \mathcal{V}'$, $\alpha_{\boldsymbol{v}} > 0$, where $\mathcal{V}' \subseteq \mathcal{V}$. Thus $\boldsymbol{l} \cdot (\boldsymbol{x} - \boldsymbol{x}') = \sum_{\boldsymbol{v}} \alpha_{\boldsymbol{v}} \boldsymbol{l} \cdot \boldsymbol{v}$ (similarly for $\boldsymbol{l'}$). Thus if $\boldsymbol{l} \cdot (\boldsymbol{x} - \boldsymbol{x}') = 0$, then for all $\boldsymbol{v} \in \mathcal{V}'$, $\boldsymbol{l} \cdot \boldsymbol{v} = 0$. Since $\boldsymbol{l}$ has maximal $\pi_{\mathcal{V}}(\boldsymbol{l})$, it follows that $\boldsymbol{l'} \cdot \boldsymbol{v} = 0$ for any $\boldsymbol{v} \in \mathcal{V}'$. But then $\boldsymbol{l'} \cdot (\boldsymbol{x} - \boldsymbol{x}') = 0$. We thus have proved the contrapose of the result. □

4.4 Multidimensional algorithm

Like in Algorithm 1, we incrementally apply the monodimensional algorithm to complete a multidimensional ranking function until it is strict.

Algorithm 6 Multidimensional Algorithm MULTIDIMENSIONAL

Require: $\mathcal{I}$ and τ
  $\rho \leftarrow \emptyset$
  failed $\leftarrow$ false
  repeat
    $(\boldsymbol{l}, \mathit{strict}) \leftarrow$ MONODIMENSIONAL $\left(\mathcal{I}, \tau \wedge \bigwedge_{\boldsymbol{l} \in \rho} (\boldsymbol{x} - \boldsymbol{x}') \cdot \boldsymbol{l} = \boldsymbol{0}\right)$
    if $\neg$*strict* **then**
      if $\boldsymbol{l}$ is in the span of ρ **then**
        failed $\leftarrow$ true
      else
        $\rho \leftarrow \rho \cup \{\boldsymbol{l}\}$
      end if
    end if
  until *strict* $\vee$ *failed*
  return if *failed* then "None" else ρ

This algorithm terminates in at most n iterations because ρ is a free family of strictly increasing size in $\mathbb{Q}^n$.

Lemma 3. *Let ρ be the multidimensional weak ranking function produced by the algorithm and ρ' be another multidimensional weak ranking function. Then, for all $(\boldsymbol{x}, \boldsymbol{x}') \in \mathcal{I} \wedge \tau$, if $\rho'(\boldsymbol{x}) \succ \rho'(\boldsymbol{x}')$ then $\rho(\boldsymbol{x}) \succ \rho(\boldsymbol{x}')$.*

Proof. Let us assume that for some $(\boldsymbol{x}, \boldsymbol{x}') \in \tau \wedge \mathcal{I}$ we have $\rho'(\boldsymbol{x}) \succ \rho'(\boldsymbol{x}')$ but $\rho(\boldsymbol{x}) = \rho(\boldsymbol{x}')$. Then, there must be a least i such that $\rho'_i \cdot (\boldsymbol{x} - \boldsymbol{x}') > 0$; and for $j < i$, $\rho'_j \cdot (\boldsymbol{x} - \boldsymbol{x}') = 0$. Without loss of generality, we choose such a couple $(\boldsymbol{x}_0, \boldsymbol{x}'_0) \in \tau \wedge \mathcal{I}$ of least i. Thus, for all $(\boldsymbol{x}, \boldsymbol{x}')$, if $\forall j < i \ \rho_j \cdot (\boldsymbol{x} - \boldsymbol{x}') = 0$ then $\forall j < i \ \rho'_j \cdot (\boldsymbol{x} - \boldsymbol{x}') = 0$; since otherwise it would mean that there would be a $j < i$ such that $\rho'_j \cdot (\boldsymbol{x} - \boldsymbol{x}') > 0$ and $\rho_j \cdot (\boldsymbol{x} - \boldsymbol{x}') = 0$, which would contradict the minimality of i.

It follows that over $\{(\boldsymbol{x}, \boldsymbol{x}') \mid (\boldsymbol{x}, \boldsymbol{x}') \in \mathcal{I} \wedge \tau \wedge \bigwedge_{j<i} \rho_j \cdot (\boldsymbol{x} - \boldsymbol{x}') = 0\}$, ρ'_i is also a weak ranking function; but so is ρ_i. By lemma 2, for all $(\boldsymbol{x}, \boldsymbol{x}')$ in this set, if $\rho'_i \cdot (\boldsymbol{x} - \boldsymbol{x}') > 0$ then $\rho_i \cdot (\boldsymbol{x} - \boldsymbol{x}') > 0$. This is true in particular for $(\boldsymbol{x}_0, \boldsymbol{x}'_0)$, which leads to contradiction. □

5 Implementation choices

In this section, I will describe the implementation and the different choices I had to make during this internship. I will first explain which programming language I chose an why I chose it. I will then

present the input format used to describe the automata, followed by the different libraries and tools used. Finally, I will detail the implementation of the three algorithms, and the output format.

5.1 Choice of implementation language

I had the choice between OCaml and C++. I chose the OCaml language because I am more familiar with it and because there already exists OCaml libraries for most of the functionalities I needed.

5.2 Input format

Integer interpreted automaton description language I could have used the FAST language (`http://www.lsv.ens-cachan.fr/Software/fast/`), which is used for verification and invariant computation, and was used in the implementation of the algorithm 1. Instead, I chose the NTS language (`http://nts.imag.fr/index.php/Main_Page`), whose representation is more adapted to the new algorithm (with FAST, each transition is labelled by its guard and its action, whereas with NTS, it is labelled by the formula $\mathcal{I} \wedge \tau$). It also provides an OCaml API.

Pre-processing These interpreted automata are obtained by preprocessing a C code. The C2FSM tool translates a C program into an interpreted automaton in the FAST language. For now, there is no such tool to obtain automata in the NTS language, thus the algorithm only proves termination of NTS interpreted automata. However, David Monniaux is currently working on a tool base on `llvm` (`http://llvm.org`) which will do this, `llvm2nts`, and will enable the algorithm to prove the termination of C programs.

Input format restrictions For now, the algorithm only works for automata with one node. It should however not be hard to adapt it to any automaton, in a similar way as in [6].

5.3 Choices of libraries

Arbitrary precision arithmetic Even if there is an arbitrary precision arithmetic module in OCaml (`Num`), I used Zarith (`http://forge.ocamlcore.org/projects/zarith`), which gives better performances.

Smt-solvers I first used the Z3 SMT-solver (`http://z3.codeplex.com/`) which is open-source and provides an OCaml API. However, the API being outdated, I had to use the SMTLIB interface. As SMTLIB is a textual standard, I had to parse the output of the solver to obtain the results of the SMT-tests. I then reused this parser with OptiMathSat ([8]) to make SMT-tests with a minimization.

Linear programming solver There are several linear programming solvers which provide an OCaml API, but here we work in arbitrary precision arithmetic. None of the solvers I found could work with arbitrary precision numbers, so I used Pip (`http://www.piplib.org/`) as an external tool, and parsed its output.

5.4 Implementation of the first monodimensional algorithm

I will here give some details about the implementation the first monodimensional Algorithm 2 (which doesn't always terminate).

The data structures I used are :

- Arrays of rationals for the column vectors : $\boldsymbol{\lambda}, \boldsymbol{\delta}, \boldsymbol{l}, \boldsymbol{l}_i, \boldsymbol{x}, \boldsymbol{x}'$
- Lists of arrays for the set $\mathcal{C}$
- Trees for the formulas $\mathcal{I}$ and τ

To represent $\mathcal{I}$, I used both the formula, and an array of couples $(\boldsymbol{l}_i, c_i)$. The first one is used for the SMT tests, whereas the second one is used in the LP tests and the computation of $\boldsymbol{l}$.

The tree type used to represent the formulas is defined as :

```
(**
   Type representing an arithmetic expression
*)
type expression =
| Cst of Q.t
| ProgramVar of bool * int (* Variable used in the program :
  (false,i) = "x_i", (true,i) = "x'_i" *)
| OtherVar of string (* Other variables (cost, ...) *)
| Neg of expression
| Add of expression * expression
...

(**
   Type representing a formula
*)
type relation =
| Eq of expression * expression
| Neq of expression * expression
...
| And of relation * relation
| Or of relation * relation
| Not of relation
| True
| False
```

I implemented two functions that make the interface with the LP and SMT solvers, which are called within the OCaml main program with :

- `Lp.lp c inv lambda delta`
- `Smtsolver.check_sat prob (x,x')`

The `Lp.lp` function first generates an instance of the $LP(\mathcal{C}, \mathcal{L}_\mathcal{I})$ problem. The generation of the SMT-test instance `prob` is done before calling `Smtsolver.check_sat`. These two functions then write the problems in the format of the solver (PIP or SMTLIB), call it, parse the output, and write the results to the last arguments (`lambda` and `delta` or `(x,x')`). `Smtsolver.check_sat` also returns a boolean which is `true` iff the problem is satisfiable.

```
x = [1; 2; ]
y = [1; 1; ]
x - y = [0; 1; ]
lambda = [0; 1; ]
delta = [1; ]
l = [0; 1; ]
------
x = [2; 1; ]
y = [1; 1; ]
x - y = [1; 0; ]
lambda = [1; 1; ]
delta = [1; 1; ]
l = [1; 1; ]
------
l = [1; 1; ], true
```

(a) Example 10 (`gcd`)

```
x = [0; 0; 0; ]
y = [-1; 0; 0; ]
x - y = [1; 0; 0; ]
lambda = [1; 0; 0; 0; ]
delta = [1; ]
l = [1; 0; 0; ]
------
x = [0; 1; 1; ]
y = [0; 0; 1; ]
x - y = [0; 1; 0; ]
lambda = [1; 1; 0; 0; ]
delta = [1; 1; ]
l = [1; 1; 0; ]
------
x = [0; 0; 1; ]
y = [-1; 1; 1; ]
x - y = [1; -1; 0; ]
lambda = [2; 1; 0; 0; ]
delta = [1; 1; 1; ]
l = [2; 1; 0; ]
------
```

```
x = [0; 0; 2; ]
y = [-1; 2; 2; ]
x - y = [1; -2; 0; ]
lambda = [3; 1; 0; 0; ]
delta = [1; 1; 1; 1; ]
l = [3; 1; 0; ]
------
x = [0; 0; 3; ]
y = [-1; 3; 3; ]
x - y = [1; -3; 0; ]
lambda = [4; 1; 0; 0; ]
delta = [1; 1; 1; 1; 1; ]
l = [4; 1; 0; ]
------
x = [0; 0; 4; ]
y = [-1; 4; 4; ]
x - y = [1; -4; 0; ]
lambda = [5; 1; 0; 0; ]
delta = [1; 1; 1; 1; 1; 1; ]
l = [5; 1; 0; ]
------
...
```

(b) Non-terminating example (`cousot9`)

Figure 11 – Output (in verbose mode)

Figure 11 presents two examples of output that the program can return in verbose mode. For each iteration, the values of the vectors `x`, `y` ($\boldsymbol{x}'$), `x - y` ($\boldsymbol{x} - \boldsymbol{x}'$), $\boldsymbol{\lambda}$, $\boldsymbol{\delta}$ and $\boldsymbol{l}$ are given. The iterations are

separated by `------`. In (a), the last line is the value returned by the algorithm. More details on the `cousot9` example can be found in appendix A.A.

5.5 Implementation of the second monodimensional algorithm

AvoidSpace To represent a subspace of $\mathbb{Q}^n$, I used an upper triangular matrix B of size $n \times n$, such that all the non-null lines ($\{B_i \mid B_{i,i} \neq 0\}$) of the matrix form a basis of the subspace. To add a vector in this basis while keeping the matrix's properties, we add this vector in a null line, we compute the Gauss-Jordan elimination of the matrix, and we reorganize the lines to make the matrix upper triangular (the implementation was optimized, but the result is equivalent). To implement AvoidSpace, we then need to compute a basis of the complementary of the subspace. All we have to do is complete the basis by adding the vector e_i to the complementary basis if $B_{i,i} = 0$ (where e_i is the i-th vector of the canonical basis). For example, if

$$B = \begin{pmatrix} 1 & 2 & 0 & 3 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Then $\{(0,1,0,0),(0,0,0,1)\}$ is a complementary basis of B.

We did not have any performance problems, and the matrix B being sparse, we think that their won't be any. However, if it turns out that this algorithm is not scalable, it is possible to improve it, for example by trying the second method proposed in 4.3, or by using a linear algebra library.

UnboundedRay The computation of ray generators is implemented with Algorithm 4. In order to generalize a model (x, y) with a conjunction of equalities P constructed with the saturated inequalities of $I \wedge \tau \wedge \boldsymbol{u} \cdot \boldsymbol{l} \leq 0$, we traverse the tree representing the formula $\mathcal{I} \wedge \tau \wedge \boldsymbol{u} \cdot \boldsymbol{l} \leq 0$ (which is composed only of $\wedge$, $\vee$, $\leq$, $\geq$ and $=$). If the formula is $A \wedge B$, we return $A' \wedge B'$ (where A' is the generalized model extracted from A). If it is $A \vee B$, we return the generalized model extracted from the formula that is `true` when evaluated with (x, y) (if A and B are `true`, we return A). If the formula is an (in)equality ($A = B$, $A \leq B$ or $A \geq B$), we return $A = B$ if the model is saturated by the (in)equality (i.e. if the evaluation of $A = B$ is `true`), else we return `true`.

5.6 Implementation of the multidimensional algorithm

The implementation of this algorithm follows the pseudo-code given in Algorithm 6, and I won't develop it.

6 Results

I have tested the algorithms on some programs extracted from the WTC toolsuite benchmarks (see Appendix A for more details). The tool translating C programs in interpreted automata not being available yet, I chose the examples that could be translated into an interpreted automaton with one node and did it manually, as well as the computation of the invariants.

On all these examples, a correct result is obtained instantaneously, and the sizes of the LP problems are reasonable (less than 10 unknowns, and less than 10 inequalities).

The ranking functions found for these programs are listed in table 1.

Program	Algorithm 2	Algorithm 5	Algorithm 6
`cousot9`	-	i (ns)	(i, j)
`cousot9_cst`	$11 \times i + j$	$11 \times i + j$	$11 \times i + j$
`easy1`	$-1/2 \times x$	$-1/2 \times x$	$-1/2 \times x$
`easy2`	z	z	z
`example1`	$-x$	$-x$	$-x$
`example2`	$-x - y - z$	$-x - y - z$	$-x - y - z$
`exemple3`	-	$1/2 \times j$ (ns)	$(1/2 \times j, i)$
`exmini`	$-1/2 \times i - 1/2 \times j + 1/2 \times k$	$-1/4 \times i - 1/4 \times j + 1/4 \times k$	$-1/4 \times i - 1/4 \times j + 1/4 \times k$
`gcd`	$x + y$	$x + y$	$x + y$
`random1d`	$-x + \max$	$-x + \max$	$-x + \max$
`real`	0 (ns)	x (ns)	None
`wcet2`	$-11 \times i - j$	$-11 \times i - j$	$-11 \times i - j$
`wise`	0 (ns)	0 (ns)	None

Table 1 – Results obtained with the three implemented algorithms. In the first column, "-" means that the algorithm does not terminate on this automaton. "(ns)" means that the ranking found is not strict.

7 Conclusion and future work

In conclusion, the main objectives of this internship (understanding and implementing the algorithms, testing them, and finishing to prove them) have been met. However, there is still some work remaining in order to have a fully functional termination analysis program. First, implementing the algorithm for several control point. Second, finishing the `llvm2nts` frontend which will be used to translate C programs into interpreted automata, and third, implementing the computation of the invariants. The next step would be to test the algorithm on larger programs, to compare the execution time of the improved algorithm and the previous one, and, more importantly, the sizes of the LP instances generated to ensure that the scalability has been improved.

The work this internship was based on has not been published yet. Laure Gonnord and David Monniaux are currently writing a research report and an article that should be submitted soon.

Appendices

A Benchmarks

In this appendix, I present the benchmarks used to test the algorithms. All the C programs are extracted from the WTC toolsuite benchmarks (`http://compsys-tools.ens-lyon.fr/wtc/index.html`). I do not provide C codes for `example1`, `example2`, `example3` and `real` in so far as these examples have been created directly as automata in order to test the behaviour of the algorithms in particular cases that may have been problematic.

A.A cousot9

```
int cousot9(){
  int i,j,N;

  i=N;
  while(i>0) {
    if(j>0) {j--;}
    else
      {
j=N;i--;
      }
  }

  return 0;
}
```

$$\tau: \begin{array}{l}\big(i \geq 0 \wedge j \geq 1 \wedge i' = i \wedge j' = j-1 \wedge N' = N\big) \vee \\ \big(i \geq 0 \wedge j \leq 0 \wedge i' = i-1 \wedge j' = N \wedge N' = N\big)\end{array}$$

$$\mathcal{I}: \begin{cases} i \geq 0 \\ j \geq 0 \\ N \geq 0 \\ -j + N \geq 0 \end{cases}$$

$$\rho(\boldsymbol{x}) = \begin{pmatrix} i \\ j \end{pmatrix}$$

A.B cousot9_cst

```
int cousot9(){
  int i,j;

  i=10;
  while(i>0) {
    if(j>0) {j--;}
    else
      {
j=10;i--;
      }
  }

  return 0;
}
```

$$\tau: \begin{array}{l}\big(i \geq 0 \wedge j \geq 1 \wedge i' = i \wedge j' = j-1\big) \vee \\ \big(i \geq 0 \wedge j \leq 0 \wedge i' = i-1 \wedge j' = 10\big)\end{array}$$

$$\mathcal{I}: \begin{cases} i \geq 0 \\ j \geq 0 \\ -j \geq -10 \end{cases}$$

$$\rho(\boldsymbol{x}) = 11 \times i + j$$

A.C easy1

```
int easy1() {
  int x, y, z;
  x=0;
  y=100;
  while (x<40) {
    if (z==0) {
      x=x+1;
    } else {
      x=x+2;
    }
  }
}
```

$$\tau: \begin{array}{l}\big(x \leq 39 \wedge z = 0 \wedge x' = x+1 \wedge z' = z \wedge y' = y\big) \vee \\ \big(x \leq 39 \wedge z \leq -1 \wedge x' = x+2 \wedge z' = z \wedge y' = y\big) \vee \\ \big(x \leq 39 \wedge z \geq 1 \wedge x' = x+2 \wedge z' = z \wedge y' = y\big)\end{array}$$

$$\mathcal{I}: \begin{cases} x \geq 1 \\ -x \geq -41 \\ y \geq 100 \\ -y \geq -100 \end{cases}$$

$$\rho(\boldsymbol{x}) = -1/2 \times x$$

A.D easy2

```
int easy2() {
  int x=12;
  int y=0;
  int z;
  while (z>0) {
    x=x+1;
    y=y-1;
    z=z-1;
}
return 0;
}
```

$$\tau: \big(z \geq 1 \wedge x' = x+1 \wedge y' = y-1 \wedge z' = z-1\big)$$

$$\mathcal{I}: \begin{cases} x \geq 12 \\ -y \geq 0 \\ z \geq 0 \end{cases}$$

$$\rho(\boldsymbol{x}) = z$$

A.E example1

$$\tau : \begin{array}{l} \big(x \leq 10 \wedge y \geq 0 \wedge x' = x + 1 \wedge y' = y + 1\big) \vee \\ \big(x \leq 0 \wedge y \geq 0 \wedge x' = x + 1 \wedge y' = y - 1\big) \end{array}$$

$$\mathcal{I} : \begin{cases} x \geq -1 \\ -x \geq -11 \end{cases}$$

$$\rho(\boldsymbol{x}) = -x$$

A.F example2

$$\tau : \begin{array}{l} \big(z \geq 0 \wedge z \leq 10 \wedge x' = x \wedge y' = y \wedge z' = z + 1\big) \vee \\ \big(y \geq 0 \wedge y \leq 10 \wedge x' = x \wedge y' = y + 1 \wedge z' = z\big) \vee \\ \big(x \geq 0 \wedge y \leq 10 \wedge x' = x + 1 \wedge y' = y \wedge z' = z\big) \end{array}$$

$$\mathcal{I} : \begin{cases} x \geq 0 \\ y \geq 0 \\ z \geq 0 \\ -x \geq -10 \\ -y \geq -10 \\ -z \geq -10 \end{cases}$$

$$\rho(\boldsymbol{x}) = -x - y - z$$

A.G example3

$$\tau : \begin{array}{l} \big(i \geq 1 \wedge i' = i - 1 \wedge j' = j\big) \vee \\ \big(i = 0 \wedge i' = j \wedge j' = j - 1\big) \end{array}$$

$$\mathcal{I} : \begin{cases} i \geq 0 \\ j \geq 0 \\ -i + j \geq -1 \end{cases}$$

$$\rho(\boldsymbol{x}) = \begin{pmatrix} 1/2 \times j \\ i \end{pmatrix}$$

A.H exmini

```
int exmini()
{
  int i,j,k,tmp;

  while(i<=100 && j<=k){
    tmp = i;
    i=j;
    j=tmp+1;
    k=k-1;
  }

  return 0;
}
```

$$\tau : \big(i \leq 100 \wedge j \leq k \wedge i' = j \wedge j' = i + 1 \wedge k' = k - 1\big)$$

$$\mathcal{I} : \begin{cases} -i \geq -100 \\ -j + k \geq 0 \end{cases}$$

$$\rho(\boldsymbol{x}) = -1/4 \times i - 1/4 \times j + 1/4 \times k$$

A.I gcd

```
int gcd(int x, int y){
  if(x<=0) return;
  if(y<=0) return;
  while(x != y)
    if(x<y) y = y-x;
    else x = x-y;
}
```

$$\tau: \begin{array}{l}\big(x \geq 1 \wedge y \geq x+1 \wedge x' = x \wedge y' = y-x\big) \vee \\ \big(x \geq y+1 \wedge y \geq 1 \wedge x' = x-y \wedge y' = y\big)\end{array}$$

$$\mathcal{I}: \begin{cases} x \geq 0 \\ y \geq 0 \end{cases}$$

$$\rho(\boldsymbol{x}) = x + y$$

A.J random1d

```
int random1d() {
  int a,x,max;
  if (max>0) {
    a=0;
    x=1;
    while (x<=max) {
      if (random()) a=a+1; else a=a-1;
      x=x+1;
    }
  }
}
```

$$\tau: \begin{array}{l}\big(x \leq max \wedge a' = a+1 \wedge x' = x+1 \wedge max' = max\big) \vee \\ \big(x \leq max \wedge a' = a-1 \wedge x' = x+1 \wedge max' = max\big)\end{array}$$

$$\mathcal{I}: \begin{cases} -a + x \geq 1 \\ x \geq 1 \\ max \geq 1 \\ -x + max \geq 0 \end{cases}$$

$$\rho(\boldsymbol{x}) = -x + max$$

A.K real

$$\tau: \big(x \geq 0 \wedge x' = 1/2 \times x\big)$$

$$\mathcal{I}: x \geq 0$$

$$\rho(\boldsymbol{x}) = \text{None}$$

A.L wcet2

```
int wcet2 () {
  int i,j;

  while(i<5){
    j=0;
    while(i>2 && j<=9) j++;
    i++;
  }
  return 0;
}
```

$$\tau: \begin{array}{l}\big(i \leq 4 \wedge i \geq 3 \wedge j \leq 9 \wedge j' = j+1 \wedge i' = i\big) \vee \\ \big(i \leq 4 \wedge j \geq 10 \wedge j' = 0 \wedge i' = i+1\big) \vee \\ \big(i \leq 4 \wedge j \leq 2 \wedge j' = 0 \wedge i' = i+1\big)\end{array}$$

$$\mathcal{I}: \begin{cases} j \geq 0 \\ -i \geq -5 \\ -j \geq -10 \end{cases}$$

$$\rho(\boldsymbol{x}) = -11 \times i - j$$

A.M wise

```
void wise(){
  int x, y;
  if(x<0 || y<0) return;
  while(x-y>2 || y-x>2)
    if(x < y) ++x;
    else ++y;
}
```

$$\tau: \begin{array}{l}\big(x \geq 0 \wedge y \geq 0 \wedge x - y \geq 3 \wedge y' = y+1 \wedge x' = x\big) \vee \\ \big(x \geq 0 \wedge y \geq 0 \wedge y - x \geq 3 \wedge x' = x+1 \wedge y' = y\big)\end{array}$$

$$\mathcal{I}: \begin{cases} x \geq 0 \\ y \geq 0 \end{cases}$$

$$\rho(\boldsymbol{x}) = \text{None}$$

References

[1] C. Alias, A. Darte, P. Feautrier, and L. Gonnord. Multi-dimensional Rankings, Program Termination, and Complexity Bounds of Flowchart Programs. In *17th International Static Analysis Symposium, SAS'10*, Perpignan, France, September 2010.

[2] Christophe Alias, Alain Darte, Paul Feautrier, Laure Gonnord, and Clément Quinson. Program termination and worst-time complexity with multi-dimensional affine ranking functions. Research Report 7037, INRIA, November 2009.

[3] Guillaume Andrieu, Christophe Alias, and Laure Gonnord. Modular termination of C programs. Research Report RR-8166, INRIA, December 2012.

[4] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 84–97, Tucson, Arizona, 1978. ACM Press, New York, NY.

[5] Paul Feautrier and Laure Gonnord. Accelerated Invariant Generation for C Programs with Aspic and C2fsm. In *Tools for Automatic Program AnalysiS*, page ., Perpignan, France, 2010.

[6] D. Monniaux and L. Gonnord. Using bounded model checking to focus fixpoint iterations. In *18th International Static Analysis Symposium, SAS'11*, Venice, Italy, September 2011.

[7] A. Schrijver. *Theory of linear and integer programming.* Wiley, NewYork, 1986.

[8] Roberto Sebastiani and Silvia Tomasi. Optimization in smt with la(q) cost functions. In *Proceedings of the 6th international joint conference on Automated Reasoning*, IJCAR'12, pages 484–498, Berlin, Heidelberg, 2012. Springer-Verlag.