Synthesis of ranking functions using extremal counterexamples

Gabriel Radanne*

Under the supervision of David Monniaux† and Laure Gonnord‡

June 27, 2014

We present a complete method for synthesizing lexicographic linear ranking functions, supported by inductive invariants, in the case where the transition relation includes disjunctions and existentials (large block encoding of complex control flow).

Previous work would expand the transition relation into disjunctive normal form, or equivalently would expand the block transition into (exponentially many) elementary transitions, prior to computing the ranking function, resulting in a very large global constraint system. In contrast, our algorithm sees elementary transitions only as needed, and builds a global constraint system lazily.

keywords Static analysis, Termination proof, Ranking function, LLVM.

Forewords

This report aims to present the work accomplished during my 5 month internship in Verimag, under the supervision of David Monniaux and Laure Gonnord. A first version of the algorithm described in this report was proposed by David Monniaux and Laure Gonnord. Lucas Seguinot improved it and implemented a first version during a previous internship. Part of the first three sections comes from this previous work while the last section describes my contribution. Proofs are available in a separate document, in [18].

1 Introduction

Program termination is shown by exhibiting a ranking function — a function from program states to a well-founded ordering, such that taking any transition in the program makes the function decrease.

Because the problem of finding a ranking function is equivalent to that of proving termination, which is undecidable, automated approaches are incomplete: they typically search for ranking functions in restricted classes: if a ranking function is found, then the program necessarily terminates, but it may still terminate even though no function is found within the class. One popular class is *linear ranking functions*: such a function ρ maps the vector $\vec{x} \in \mathbb{Z}^n$ of variables of the program to an integer linear combination $\sum_i \alpha_i \vec{x}_i$ such that $\rho(\vec{x}') < \rho(\vec{x})$ for any possible program step $\vec{x} \to \vec{x}'$, and such that $\rho(\vec{x}) \geq 0$ for any reachable program state (remark that this entails proving an auxiliary invariant property). Furthermore, the α_i may be allowed to depend on the program point k, thus the unknowns are $\alpha_{i,k}$: the decreasing condition becomes $\sum_i \alpha_{i,k} \vec{x}_i < \sum_i \alpha_{i,k'} \vec{x}'_i$ for any transition $(k, \vec{x}) \to (k, \vec{x'})$.

Various methods for the automated synthesis of such functions have been proposed [16]; they build a constraint system in the unknowns $\alpha_{i,p}$ and solve it. This class is extended to *lexicographic linear ranking functions*: instead of a single function $\rho(\vec{x})$, one uses a tuple of them $\langle \rho_1(\vec{x}), \ldots, \rho_m(\vec{x}) \rangle$, which is shown to be strictly decreasing with

*ENS Rennes
†CNRS, VERIMAG
‡Université Lyon1, LIP (UMR CNRS / ENS Lyon / UCB Lyon 1 / INRIA)

respect to lexicographic ordering. Again, the function ρ can be allowed to depend on the program point, and complete automated synthesis methods exist for this class of functions [1].

A common weakness of many previous methods is the need to solve for linear coefficients at all program points, or at least at all basic blocks: if the program has K basic blocks, n integer variables and one searches for a m-dimensional ranking function, then the number of unknowns in the constraint system is $K.m.n$, which may lead to scalability issues [1]. It is therefore desirable to limit the set of points to consider to heads of loops, or, more generally, a *cut-set* of program points: a set such that removing these points cuts all cycles in the program (e.g. all heads of loops in a structured program without recursive functions). Furthermore, there exists loops such that there is a linear lexicographic ranking function that decreases along each path inside the loop, from one loop iteration to the next, but such that there is no lexicographic linear ranking function that decreases at each step along these paths. For these reasons, it is tempting to treat each path inside a loop as a single transition [11]; unfortunately the number of paths may be exponential in the size of the program (e.g. if the loop consists in t successive if-then-else tests, the number of paths is 2^t), thus the constraint system may become very large, even though it features fewer variables.

Our new approach is based on the following insight: even though the number of paths may be large, it is possible that few of them actually matter in the constraint system (we shall make this concept precise by giving a characterization as geometric extremal points). Our algorithm therefore builds the constraint system lazily, taking paths into account as needed.

Contents This paper is organized as follows. After recalling the main definitions and notations (Section 2), we present the main ideas of a first algorithm in Section 3, and then the workarounds to ensure its own termination (Section 4). We will describe the implementation Section 5.

2 Preliminary definitions

In this section, we will define the concepts used in the rest of the article. After recalling the definition of closed convex polyhedra, we 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. Theses concepts will be illustrated in Example 1.

We write column vectors in boldface (as $\boldsymbol{x}$). Sets are represented with calligraphic letters such as $\mathcal{W}$, $\mathcal{P}$, 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 pairs $(\boldsymbol{a}_i, b_i)$ such that $\mathcal{P} = \{\boldsymbol{x} \mid \bigwedge_i \boldsymbol{a}_i.\boldsymbol{x} \geq b_i\}$.*

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

$$\mathcal{P} = \left\{ \left(\sum_i \alpha_i \boldsymbol{v}_i \right) + \left(\sum_i \beta_i \boldsymbol{r}_i \right) \middle| \alpha_i \geq 0, \beta_i \geq 0, \sum_i \alpha_i = 1 \right\} \tag{1}$$

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

2.2 Transitions

We consider programs over a state space $\mathcal{W} \times \mathbb{Q}^n$, where $\mathcal{W}$ is the finite set of control states, defined by an initial state and a transition relation τ. We denote by R_k the set of all values of $\boldsymbol{x}$ when the flow is in k.

Definition 2 (Invariants) *An* invariant *on a control point $k \in \mathcal{W}$ is a formula $\phi_k(\boldsymbol{x})$ that is true for all reachable states $(k, \boldsymbol{x})$. It is* affine *if it is a conjunction of a finite number of affine conditions on program variables: the set $\mathcal{R}_k$ is then over-approximated by a convex polyhedron $\mathcal{P}_k$.*

We do not require the invariant to be inductive. We also assume that some external tool provides us with invariants: either affine invariants provided by e.g. polyhedral analysis [4] or its refinements as in the Aspic[1] [10, 8] or the Pagai[2] [12] tools ; or arbitrary invariants, obtained by whatever approach (predicate abstraction etc.) in a theory such that its combination with linear algebra is decidable.[3]

Definition 3 *A 2-invariant at a pair of control points (k, k') is a set $\Phi_{k,k'}$ containing all $(\boldsymbol{x}, \boldsymbol{x}')$ such that $\boldsymbol{x}$ is reachable and $(k, \boldsymbol{x}, k', \boldsymbol{x}') \in \tau$.*[4]

2.3 Ranking functions

We suppose we have an invariant $\mathcal{I}$ and a 2-invariant $\Phi = \cup\Phi_{k,k'}$.

Definition 4 (Ranking function) *A* (strict) linear ranking function *is a function $\rho : \mathcal{W} \times \mathbb{Q}^n \to \mathbb{Q}$, such that*

1. *for any state $k \in \mathcal{W}$, $\boldsymbol{x} \mapsto \rho(k, \boldsymbol{x})$ is affine linear;*
2. *for any transition $(k, \boldsymbol{x}, k', \boldsymbol{x}')$ in the 2-invariant Φ, $\rho(k', \boldsymbol{x}') \leq \rho(k, \boldsymbol{x}) - 1$;*
3. *for any state $(k, \boldsymbol{x})$ in the invariant $\mathcal{I}$, $\rho(k, \boldsymbol{x}) \geq 0$;*

A weak linear ranking function *replaces condition 2 by $\rho(k', \boldsymbol{x}') \leq \rho(k, \boldsymbol{x})$.*

Definition 5 *A* lexicographic (strict) linear ranking function *of dimension m is a function $\rho : \mathcal{W} \times \mathbb{Q}^n \to \mathbb{Q}^m$, such that*

1. *for any state $k \in \mathcal{W}$, $\boldsymbol{x} \mapsto \rho(k, \boldsymbol{x})$ is affine linear;*
2. *for any transition $(k, \boldsymbol{x}, k', \boldsymbol{x}')$ in the 2-invariant Φ, $\rho(k', \boldsymbol{x}') \prec \rho(k, \boldsymbol{x})$*
   where $\langle x_1, \ldots, x_m \rangle \prec \langle y_1, \ldots, y_m \rangle$ if and only if there exists an i such that $x_j = y_j$ for all $j < i$ and $x_i \leq y_i - 1$;
3. *for any state $(k, \boldsymbol{x})$ in the invariant $\mathcal{I}$, all coordinates of $\rho(k, \boldsymbol{x})$ are nonnegative.*

A weak lexicographic linear ranking function *replaces condition 2 by $\rho(k', \boldsymbol{x}') \preceq \rho(k, \boldsymbol{x})$ where $\preceq$ is the lexicographic ordering.*

As the strict orderings $<$ and $\prec$ are well-founded within the invariant, the existence of a strict ranking function entails the termination of the program.

Definition 6 *A lexicographic linear ranking function is said to be* monodimensional *if its dimension m is* 1 *(and then it is the same as a linear ranking function); else it is said to be* multidimensional.

In this report, for the sake of brevity and simplicity, we will restrict ourselves to monodimensional ranking functions. however this work as already been extended to lexicographic ranking functions [18].

3 Monodimensional weak ranking functions of maximal termination power

In a nutshell, finding a monodimensional ranking function amounts to finding a vector in the orthogonal cone of the cone generated by a projection of an invariant on successive states of the system. Let us see why, and then examine a naive (but wrong) algorithm to compute such a vector: this algorithm constructs the desired cone as the solution set of a constraint system, generated through a counterexample-guided approach.

[1] http://laure.gonnord.org/aspic/
[2] http://pagai.forge.imag.fr/
[3] The general condition is that it should be possible to compute a polyhedron enclosing the projection on the numerical variables of the solution of a conjunction of theory predicates.
[4] This notion resembles that of "transition invariants" but these consider arbitrary runs between k and k'.

3.1 Weak ranking functions expressed in terms of cones

We first recall the definition of convex cones and orthogonality [9, §4.2].

Definition 7 *A convex cone of $\mathbb{Q}^n$ is a subset Δ of $\mathbb{Q}^n$ such that :*

- *For all $\boldsymbol{x} \in \Delta$, $\alpha > 0$, $\alpha\boldsymbol{x}$ is also in Δ.*
- *For all $\boldsymbol{x}, \boldsymbol{y} \in \Delta$, $\boldsymbol{x} + \boldsymbol{y}$ is also in Δ.*

Definition 8 *The orthogonal $\Delta^\perp$ of Δ is defined as $\{\boldsymbol{u} \mid \forall \boldsymbol{v} \in \Delta,\ \boldsymbol{u}.\boldsymbol{v} \geq 0\}$. If Δ is a convex cone, $\Delta^\perp$ is also a convex cone.*

In the following, we will make the following assumptions, some of which will be relaxed later:

- We deal with a unique control point k with $\mathcal{I}$ its invariant and τ its transition relation.[5]
- $\mathcal{I}$ is a closed convex polyhedron given by a set of inequalities $\mathcal{I} = \{\boldsymbol{x} \mid \bigwedge_{i=1}^{m} \boldsymbol{a}_i.\boldsymbol{x} + b_i \geq 0\}$, $\text{Cons}_\mathcal{I}$ will denote the set $\{a_i, i \in [1, m]\}$.
- τ is a finite union of closed convex polyhedra. Since from an algorithmic point of view, not all such descriptions are equivalent with respect to complexity, let us assume that τ is given as the solution over $x_1, \ldots, x_n$ and $x'_1, \ldots, x'_n$, the values before and after the transition (all other variables being considered as implicitly existentially quantified), of a formula built from $\wedge$, $\vee$ and non-strict linear inequalities and linear equalities (we thus exclude negation and strict inequalities).
- We are looking for linear affine ranking functions $\rho(\boldsymbol{x}) = \boldsymbol{l}.\boldsymbol{x} + \ell$.

Then, the set of weak ranking functions forms a closed convex cone, because the sum of two ranking functions is a ranking function, and multiplying all coefficients of a given ranking function by the same positive constant also provides a ranking function. Moreover, $\mathbf{0}$ is a weak ranking function.

Example 1 (Generators for polyhedral invariants) *As a first example, we consider the following transition relation on $\mathbb{Q}^2$:*

$$\boldsymbol{t_2} : \frac{0 \leq x \wedge 0 \leq y}{\begin{array}{l} x := x - 1 \\ y := y - 1 \end{array}} \quad k_0 \quad \boldsymbol{t_1} : \frac{x \leq 10 \wedge 0 \leq y}{\begin{array}{l} x := x + 1 \\ y := y - 1 \end{array}}$$

Under initial assumptions $x = 5, y = 10$, an invariant generator (Aspic) gives the following inductive invariant for $\mathcal{I}$:

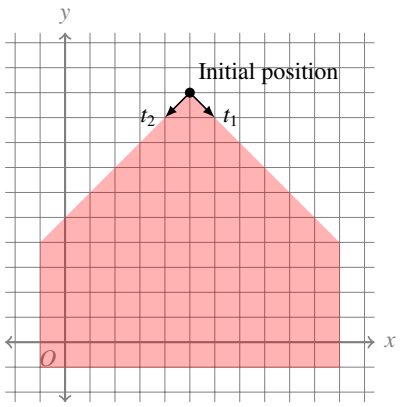
$$\mathcal{I} = \{0 \leq x + 1, x \leq 11, 0 \leq y + 1, y \leq x + 5, x + y \leq 15\}$$

from these invariants we compute the $\boldsymbol{a}_i$:

$$\boldsymbol{a}_1 = (1, 0)^\top ; \boldsymbol{a}_2 = (-1, 0)^\top ; \boldsymbol{a}_3 = (0, 1)^\top ; \boldsymbol{a}_4 = (1, -1)^\top ; \boldsymbol{a}_5 = (1, 1)^\top$$

$$b_1 = 1 ; b_2 = 11 ; b_3 = 1 ; b_4 = 5 ; b_5 = 15$$

A possible monodimensional affine ranking function for this automaton is $\rho(x, y) = y + 1$.



The rest of the section is a reformulation of properties obtained in the field of polyhedral scheduling [6, 7].

[5] This also emulates the situation where one looks for the exact same linear function at all control points.

Proposition 1 *Under the previous assumptions, $\rho = \boldsymbol{l}.\boldsymbol{x} + \ell$ is a ranking function if and only if the following two conditions hold :*

$$\forall \boldsymbol{x}, \boldsymbol{x'}\ \boldsymbol{x} \in \mathcal{I} \wedge (\boldsymbol{x}, \boldsymbol{x'}) \in \tau \implies \boldsymbol{l}.(\boldsymbol{x} - \boldsymbol{x'}) \geq 0 \tag{2}$$

$$\exists \lambda_1 \geq 0, \ldots, \lambda_m \geq 0,\ \boldsymbol{l} = \sum_{i=1}^{m} \lambda_i \boldsymbol{a_i} \tag{3}$$

Equation 3 means that $\boldsymbol{l}$ belongs to the cone generated by the $\boldsymbol{a}_i$ constraints, denoted as $cone_{cons}(\{\boldsymbol{a}_i\})$ or $cone_{cons}(\mathcal{I})$ in the sequel. Equation 2 is also equivalent to stating that $\boldsymbol{l}$ belongs to some (other) cone, as we will see in the following. Let $\mathcal{P}_{I,\tau} = \{\boldsymbol{x} - \boldsymbol{x'} \mid \boldsymbol{x} \in \mathcal{I} \wedge (\boldsymbol{x}, \boldsymbol{x'}) \in \tau\}$ be the set of all reachable 1-step differences (This polyhedra has also been used in [2] to compute affine invariants). Equation 2 is then equivalent to stating that $\boldsymbol{l}$ belongs to the orthogonal of the convex cone generated by $\mathcal{P}_{\mathcal{I},\tau}$.

Let us reexamine the condition that $\boldsymbol{l}.\boldsymbol{u} \geq 0$ for all $\boldsymbol{u} \in \mathcal{P}_{\mathcal{I},\tau}$. Remark that this condition is left unchanged if we replace $\mathcal{P}_{\mathcal{I},\tau}$ by its convex hull (a linear inequality is true over a set if and only if it is true over the convex hull of this set). Because $\mathcal{I}$ and τ are (finite unions of) convex closed polyhedra, so is $\mathcal{P}_{\mathcal{I},\tau}$, and thus the closure of its convex hull $\mathcal{P}^H_{\mathcal{I},\tau}$ is a closed convex polyhedron.[6] If bounded, then this convex polyhedron is just the convex hull of its vertices, but if unbounded one has to include rays and lines as generators, as in Equation 1.

The condition $\boldsymbol{l}.\boldsymbol{u} \geq 0$ for all $\boldsymbol{u} \in \mathcal{P}^H_{\mathcal{I},\tau}$ can thus be replaced by $\forall i\ \boldsymbol{l}.\boldsymbol{v}_i \geq 0 \wedge \forall i\ \boldsymbol{l}.\boldsymbol{r}_i \geq 0$ For the sake of simplicity, we shall group these three conditions into one: there is a finite set $V = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m\}$ such that Equation 2 is equivalent to $\forall 1 \leq i \leq m,\ \boldsymbol{l}.\boldsymbol{v}_i \geq 0$, otherwise said $\boldsymbol{l}$ belongs to the polyhedral convex cone with faces defined by the $\boldsymbol{v}_i$, denoted by $cone_{gen}(\{\boldsymbol{v_j}\})$ or $cone_{gen}(V_{\mathcal{I},\tau})$.

We have expressed both conditions of Definition 4 as membership of $\boldsymbol{l}$ in polyhedral cones, respectively given by constraints and generators:

Proposition 2 *Let $Cons_{\mathcal{I}}$ the set of constraints of $\mathcal{I}$ and $V_{\mathcal{I},\tau}$ a set of generators of $\mathcal{P}^H_{\mathcal{I},\tau}$. Then $\boldsymbol{l}$ is a weak ranking function iff $\boldsymbol{l} \in cone_{cons}(\mathcal{I}) \cap cone_{gen}(V_{I,\tau})$.*

Thanks to this proposition, we can restrict the search of a ranking function to vectors that are linear combinations of $\text{Cons}_{\mathcal{I}}$ and of $V_{\mathcal{I},\tau}$, which we will do in Definition 10.

3.2 Maximal "termination power" and strict ranking functions

We are however interested in more than *weak* ranking functions (otherwise, $\boldsymbol{l} = \boldsymbol{0}$ would suffice!). In the simplest cases, we would like $\forall 1 \leq i \leq m,\ \boldsymbol{l}.\boldsymbol{v}_i > 0$: by appropriate scaling of $\boldsymbol{l}$, we can ensure that $\forall 1 \leq i \leq m,\ \boldsymbol{l}.\boldsymbol{v}_i \geq 1$ and thus $\boldsymbol{l}$ is a ranking function, proving termination. Remark that this condition $\forall i\ \boldsymbol{l}.\boldsymbol{v}_i > 0$ is otherwise expressed by "$\boldsymbol{l}$ lies in the interior $V^{\perp\circ}$ of $V^{\perp}$".

In case there is no strict ranking function, we will settle for what is the closest best:

Definition 9 *$\boldsymbol{l}$ is said to have* maximal termination power *if $\boldsymbol{l}.\boldsymbol{v}_i > 0$ for as many $\boldsymbol{v}_i$ as possible. In the framework of [1], this would be equivalent to maximizing the number of transition where the ranking function decrease strictly.*

The notations and results of this section are also adapted from Feautrier [6, 7].

For $\boldsymbol{l} \in V^{\perp}$, we define the set $\pi_{\mathcal{V}}(\boldsymbol{l}) = \{\boldsymbol{v} \in V \mid \boldsymbol{v}.\boldsymbol{l} > 0\}$; we would like $\boldsymbol{l}$ of maximal $|\pi_{\mathcal{V}}(\boldsymbol{l})|$. This last condition is equivalent to $\pi_{\mathcal{V}}(\boldsymbol{l})$ being maximum with respect to inclusion:

Proposition 3 *Given $\mathcal{V} = (\boldsymbol{v}_1, \ldots \boldsymbol{v}_N)$ 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 maximum with respect to inclusion, i.e.*

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

[6] If $\mathcal{P}_{\mathcal{I},\tau}$ is bounded, then its convex hull $\mathcal{P}^H_{\mathcal{I},\tau}$ is a closed convex polyhedron, but it might not be closed if $\mathcal{P}_{\mathcal{I},\tau}$ is unbounded: e.g. the convex hull of $\{(0,0)\}$ and $\{(1,x) \mid x \in \mathbb{R}\}$ is $\{(0,0)\} \cup (0,1] \times \mathbb{R}$, which is not closed.

Definition 10 *Given $V = (\boldsymbol{v}_1, \dots \boldsymbol{v}_N)$ a set of generators of (the convex hull of) $\mathcal{P}_{I,\tau}$ and $Cons_{\mathcal{I}} = (\boldsymbol{a}_1, \dots \boldsymbol{a}_m)$ a set of vectors (constraints of I), we denote by $LP(V, Cons_{\mathcal{I}})$ the following linear programming instance where λ_i and δ_i are the unknowns:*

$$\begin{cases} \text{Maximize } \sum_i \delta_i \text{ s.t.} & \\ \lambda_1, \dots, \lambda_m \geq 0 & \\ 0 \leq \delta_j \leq 1 & \text{for all } 1 \leq j \leq N \\ \sum_{i=1}^{m} \lambda_i (\boldsymbol{v}_j . \boldsymbol{a}_i) \geq \delta_j & \text{for all } 1 \leq j \leq N \end{cases} \tag{4}$$

The result of such an LP problem is *None* if the problem is unfeasible, or a valuation of the λ_i and δ_i variables maximizing the objective function. In the following, we denote as $\boldsymbol{\lambda}$ the vector with λ_i components. Thanks to the previous section, we have the following result:

Proposition 4 *Let $V = (\boldsymbol{v}_1, \dots \boldsymbol{v}_N)$ be a set of generators of the convex hull of $P_{I,\tau}$ and $Cons_{\mathcal{I}} = (\boldsymbol{a}_1, \dots \boldsymbol{a}_m)$ the constraints of I. Then :*

- *$LP(V, Cons_{\mathcal{I}})$ is always feasible (since $\boldsymbol{\lambda} = \mathbf{0}$ is always a solution).*
- *$LP(V, Cons_{\mathcal{I}})$ gives λ_is such that $\boldsymbol{l} = \sum_{i=1}^{m} \lambda_i \boldsymbol{a}_i$ is a weak ranking function of maximal $\pi_{\mathcal{V}}$.*

The difficulty is that computing this set V (or, equivalently, any finite set between it and $\mathcal{P}_{\mathcal{I},\tau}$) may be expensive (and the cardinal of V may be exponential in the number of constraints). Obviously, we could compute a disjunctive normal form (DNF) for $\tau \wedge \mathcal{I}$, each disjunct denoting a convex polyhedron, compute their generators $(\boldsymbol{x}, \boldsymbol{x}')$ and collect all resulting $\boldsymbol{x} - \boldsymbol{x}'$; computing the DNF has exponential cost but our lazy approach will only computes extremal points as needed, as opposed to eagerly expanding the DNF.

In the rest of the paper, when there is no ambiguity, the convex hull of $\mathcal{P}_{\mathcal{I},\tau}$ will be denoted as $\mathcal{P}_H$.

3.3 A wrong but intuitive algorithm

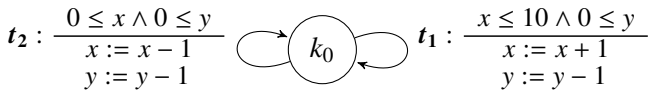
Let us propose an incremental algorithm (somewhat wrong, but it will be corrected later):

1. Consider $\boldsymbol{l}$ a ranking function.
2. Find an element $\boldsymbol{u} = \boldsymbol{x} - \boldsymbol{x}'$ in $\mathcal{P}^H_{\mathcal{I},\tau}$ which contradicts that $\boldsymbol{l}$ is a *strict* ranking function.
3. Add it to a set C and call $LP(C, \text{Cons}_{\mathcal{I}})$ to compute a new ranking function that maximises the number of stricly positive vertices.
4. Go back to step 2 until a counter example can't be found.

The algorithm is described in Algorithm 1.[7]

Proposition 5 *If Algorithm 1 terminates, then it returns a weak ranking function of maximal termination power (Definition 9).*

Example 2 *On Example 1, we already computed the generators $\boldsymbol{a}_i$ for $\mathcal{I}$. These vectors are given as input to Algorithm 1, as well as the transition function :*



$$\tau = \left\{(x, y, x', y'), x \leq 10 \wedge 0 \leq y \wedge x' = x + 1 \wedge y' = y - 1 \ \vee \ 0 \leq x \wedge 0 \leq y \wedge x' = x - 1 \wedge y' = y - 1\right\}$$

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

[7] It is important that the condition $(\boldsymbol{x} - \boldsymbol{x}').\boldsymbol{l} \leq 0$ be non strict. Otherwise, $\boldsymbol{l} = \mathbf{0}$ could be accepted immediately; more generally, identifying couples $(\boldsymbol{x}, \boldsymbol{x}')$ such that $(\boldsymbol{x} - \boldsymbol{x}').\boldsymbol{l} = 0$ is important for maximizing "termination power": we wish that the function decrease strictly as much as possible.

Algorithm 1 Initial Algorithm

Require: $\mathcal{I}$ the invariant polyhedra and τ the transition relation, over $(\boldsymbol{x}, \boldsymbol{x'})$ and (optionally) existentially quantified variables

Ensure: When it terminates, returns the components $\boldsymbol{l}$ and ℓ of a weak ranking function $\rho(\boldsymbol{x}) = \boldsymbol{l}.\boldsymbol{x} + \ell$ of maximal $\pi_{\mathcal{V}}$ and a boolean which is true iff ρ is a strict ranking function.

$C \leftarrow \emptyset$
$\boldsymbol{l} \leftarrow \boldsymbol{0}$
$\ell \leftarrow 0$
$finished \leftarrow false$
while $not(finished)$ and ($\mathcal{I} \wedge \tau \wedge (\boldsymbol{x} - \boldsymbol{x'}).\boldsymbol{l} \leq 0$ satisfiable) **do**
    $(\boldsymbol{x}, \boldsymbol{x'}) \leftarrow$ a model for the above SMT test
    $C \leftarrow C \cup \{\boldsymbol{x} - \boldsymbol{x'}\}$
    $(\boldsymbol{\lambda}, \boldsymbol{\delta}) \leftarrow LP(C, \text{Cons}_{\mathcal{I}})$
    if $\boldsymbol{\lambda} = \boldsymbol{0}$ **then**
        $finished \leftarrow true$
    else
        $\boldsymbol{l} \leftarrow \sum_{i=1}^{m} \lambda_i \boldsymbol{a}_i$
        $\ell \leftarrow \sum_{i=1}^{m} \lambda_i b_i$
    end if
end while
Return $(\boldsymbol{l}, \ell, \bigwedge_i \delta_i = 1)$.

Step 1. Beginning with $\boldsymbol{l} = \boldsymbol{0}$, the SMT-solver is given the constraint $\mathcal{I} \wedge \tau \wedge 0 \leq 0$. We are given a first model $x = -1, x' = 0, y = 0, y' = -1$, thus $C \leftarrow \{(-1, 1)^\top\}$. Then, an LP-solver is given the following LP problem :

$$\begin{cases} \textit{Maximize } \delta_1 \textit{ s.t.} \\ \lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5 \geq 0 \\ 0 \leq \delta_1 \leq 1 \\ -\lambda_1 + \lambda_2 + \lambda_3 - 2\lambda_4 \geq \delta_1 \end{cases}$$

(The last constraint comes from the scalar products of $\boldsymbol{c}_1 = (-1, 1)^\top$ with all $\boldsymbol{a}_i$.). The solver gives $\lambda_2 = 1$, $\lambda_1 = \lambda_3 = \lambda_4 = \lambda_5 = 0$. Then $\boldsymbol{l} = \boldsymbol{a}_2 = (-1, 0)^\top$.

Step 2. The new SMT-constraint is now $\mathcal{I} \wedge \tau \wedge \begin{pmatrix} x' - x \\ y - y' \end{pmatrix} \cdot \begin{pmatrix} -1 \\ 0 \end{pmatrix} \leq 0$. The solver gives us the model $x = 11, x' = 10, y = 0, y' = -1$, thus $C \leftarrow \{\boldsymbol{c}_1, \boldsymbol{c}_2\}$ where $\boldsymbol{c}_2 = (1, 1)^\top$.
The new LP instance is thus now :

$$\begin{cases} \textit{Maximize } \delta_1 + \delta_2 \textit{ s.t.} \\ \lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5 \geq 0 \\ 0 \leq \delta_1, \delta_2 \leq 1 \\ -\lambda_1 + \lambda_2 + \lambda_3 - 2\lambda_4 \geq \delta_1 \\ \lambda_1 - \lambda_2 + \lambda_3 - 2\lambda_5 \geq \delta_2 \end{cases}$$

whose optimal is $\lambda_3 = 1$, $\lambda_1 = \lambda_2 = \lambda_4 = \lambda_5 = 0$. Then $\boldsymbol{l} = \boldsymbol{a}_3 = (0, 1)^\top$ and $\ell = b_3 = 1$.

Step 3. The SMT-constraint is now $\mathcal{I} \wedge \tau \wedge y - y' \leq 0$, which is unsat.

Return. The function returns $(\boldsymbol{l} = (0, 1)^\top, \ell = 1, \mathsf{true})$.

We obtain $\rho = y + 1$, a strict ranking function for $(\tau, \mathcal{I})$.

3.4 Termination problems

Unfortunately, this simple algorithm suffers from three problems which may prevent its termination, as shows Example 3.

Infiniteness of the set of 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 cardinality of that set). Depending on the implementation of the SMT-solver, it may be the case that all models $(\boldsymbol{x}, \boldsymbol{x'})$ are vertices constructed by intersection of the linear atomic constraints in $\mathcal{I} \wedge \tau$, of which there are finitely (exponentially) many, which would ensure termination.

Note that, even in this favorable case, the algorithm will produce vertices that do not lie on the boundary of the convex hull $\mathcal{P}^H_{\mathcal{I},\tau}$, and thus accumulate unoptimally tight constraints, which may eventually become redundant. A better option is to impose that the model for the SMT-test should minimize $(\boldsymbol{x} - \boldsymbol{x'}) \cdot \boldsymbol{l}$, as in "optimization modulo theory" [15, 17], which ensures that vertices are on the boundary of $\mathcal{P}^H_{\mathcal{I},\tau}$; we shall explore this idea in Section 4.

No strict ranking function Second, even if the first issue is resolved, the above algorithm terminates only if there is a strict ranking function ($\boldsymbol{l}.\boldsymbol{v} > 0$ for all $\boldsymbol{v} \in \mathcal{V}$). 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 $(\boldsymbol{x}, \boldsymbol{x'})$.

Example 3 *The algorithm does not terminate on the automaton of Figure 1.*

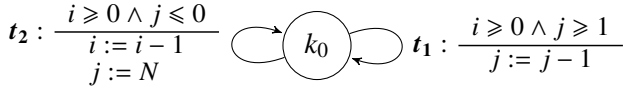


Figure 1: Non-terminating example

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 $\mathcal{P}^H_{\mathcal{I},\tau}$. 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. ■

4 Corrected mono-dimensional algorithm

Let us now address the termination issues raised in Section 3.4. We will first assume, as said in Section 3.4 that we use an optimizing SMT solver to minimize $(\boldsymbol{x} - \boldsymbol{x'}) \cdot \boldsymbol{l}$ in order to discover tight bounds of $\mathcal{P}^H_{\mathcal{I},\tau}$.

4.1 Non termination due to the absence of a strict ranking function

The set $\{\boldsymbol{u} \mid \boldsymbol{u} \cdot \boldsymbol{l} = 0 \ \forall$ weak ranking function $\boldsymbol{l}\}$ is a linear subspace of $\mathbb{Q}^n$. To prevent elements from this subspace from being returned again and again by the SMT-solver, we maintain a linearly independent 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 C, and a new optimal solution $(\boldsymbol{l}, \ell, \boldsymbol{\delta})$ 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 decision procedure AvoidSpace$(\boldsymbol{u}, \mathcal{B})$ encoding the constraint $\boldsymbol{u} \notin \text{Span}(\mathcal{B})$ to the SMT-test. This constraint can be implemented as either:

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

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

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

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

We implemented 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).

4.2 SMT formulas with unbounded domain

In light of Example 3, the issue could be corrected by adding the constraints $\boldsymbol{l} \cdot \boldsymbol{r} \geq 0$ for all ray generators $\boldsymbol{r}$ of $\mathcal{P}_H$ (we will prove later that it does). However, computing all generators of $\mathcal{P}_H$ may be expensive. Instead, we add generators on demand, as we do for vertices.

In addition to the vertices, we add to C the ray generators $\boldsymbol{r}$ incrementally, when the SMT-solver returns a model of the form $\boldsymbol{u} = \cdots + \beta \boldsymbol{r}$ such that $\boldsymbol{u} \cdot \boldsymbol{l}$ unbounded, we compute $\boldsymbol{r}$ and add it to C.[8]

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

4.3 Final algorithm

These remarks and solutions finally lead to Algorithm 2.

Algorithm 2 Modified (terminating) Algorithm MONODIMENSIONAL

Require: $\mathcal{I}$ and τ
  $C \leftarrow \emptyset$
  $\mathcal{B} \leftarrow \emptyset$
  finished $\leftarrow$ false
  $\boldsymbol{l} \leftarrow \boldsymbol{0}$
  $\ell \leftarrow 0$
  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 $\boldsymbol{u} \cdot \boldsymbol{l}$ (≤ 0) **do**
    $(\boldsymbol{u}, \mathit{unbounded}) \leftarrow$ a model for $\boldsymbol{u}$ in the above SMT test
    $C \leftarrow C \cup \{\boldsymbol{u}\}$
    if *unbounded* **then** ▷ $\boldsymbol{u} = \cdots + \beta \boldsymbol{r}$, with $\boldsymbol{r}$ a ray generator of $\mathcal{P}_H$
      $C \leftarrow C \cup \{\boldsymbol{r}\}$
    end if
    $(\boldsymbol{\lambda}, \boldsymbol{\delta}) \leftarrow LP(C, \mathrm{Cons}_{\mathcal{I}})$
    if $\boldsymbol{\lambda} = \boldsymbol{0}$ **then**
      finished $\leftarrow$ true
    else
      $\boldsymbol{l} \leftarrow \sum_{i=1}^{m} \lambda_i \boldsymbol{l}_i$
      $\ell \leftarrow \sum_{i=1}^{m} \lambda_i b_i$
      if $\delta_{\boldsymbol{u}} = 0$ **then**
        $\mathcal{B} \leftarrow \mathcal{B} \cup \{\boldsymbol{u}\}$
      end if
    end if
  end while
  Return $(\boldsymbol{l}, \ell, \bigwedge_i \delta_i = 1 \wedge (\mathcal{I} \wedge \tau \wedge \boldsymbol{x} = \boldsymbol{x'})$ 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 from returning the model $(\boldsymbol{x}, \boldsymbol{x})$. Without this additional test, if $\mathcal{B} = \emptyset$ at the end, the algorithm would return $(\boldsymbol{l}, \text{true})$ instead of $(\boldsymbol{l}, \text{false})$.

Proposition 6 *Algorithm 2 always terminates and returns a weak ranking function of maximal termination power (Definition 9)*

[8] A perhaps more satisfying solution would be to modify the linear programming optimization step in the optimization modulo theory algorithm [15, 17] so that it detects that it is moving along an unbounded edge, in which case it should return the last vertex and the direction of the edge.

5 Implementation

This section describes the implementation of these algorithms and the interaction with other tools.

A previous implementation has been done by Lucas Seguinot during his internship in 2013, however this implementation suffered several limitations, the main one being that it analyzes programs in the NTS format[13]. The NTS format describes precisely an automaton, which makes it very convenient for analysis purposes. However there seems to be no reliable tools to transform real programs (typically, C programs) to NTS and to infer invariants.

We decided to use as input format the LLVM intermediate representation.

5.1 LLVM and SSA

LLVM is a compiler infrastructure with multiple front-ends and back-ends organized around a common Intermediate Representation (IR). In particular, Clang is a C/C++ front end for LLVM. Using this input format makes the analysis possible on a very broad variety of existing programs. It also allows us to use Pagai [12] to infer invariants.

Unlike NTS, which describes the program as a counter automaton (with multiple affectations for each variable), LLVM IR is in SSA form. In Static Single Assignment form[5] each variable is assigned only *once*. A program is decomposed as a graph of basic blocks. Each basic block is a node of the control flow graph of the program and contains a list of assignments, each assignment being associated to an instruction.

In Figure 2, the C program on the left is transformed into the LLVM representation on the right. The function possess three parts: an entry point, an implicit output point, and the loop. In the LLVM representation, the entry point correspond to the block `%0`, the output point is the block `%6` and the loop is composed of the blocks `%1`, `%3` and `%4`.

Let us consider the instruction `i++` in the loop, which we can rewrite as $i = i + 1$. It corresponds to the LLVM instruction "`%5 = add i32 %i.0, 1`" which means "add the 32-bits integers `%i.0` and `1` and name the result `%5`". `%i.0` represents i and `%5` is a new variable distinct from i. This makes following uses and modification of variables easier and simplifies the presentation of most algorithms.

In order to define variables which values depends on the control flow, like i, SSA introduces ϕ-functions. A ϕ-function is placed at the beginning of a basic block. For example "`%i.0 = phi i32 [ 0, %0 ], [ %5, %4 ]`" in the block `%1` means that `%i.0` as either the value of `0` (if the previous block is `%0`) or `%5` (if the previous block is `%4`).

To move from one block to another, LLVM puts terminator instructions at the end of blocks. Terminators are either unconditional jumps like "`br label %1`", which means that the successor is the block `%1`, or conditional jumps like "`br i1 %2, label %3, label %6`", which means "if `%2` then jump to `%3` else to `%6`".

`%2` is the boolean (hence the type `i1`) result of the comparison "`icmp ult i32 %i.0, 10`", which is the translation of "`i<10`", the unsigned less than comparison on 32-bits integers.

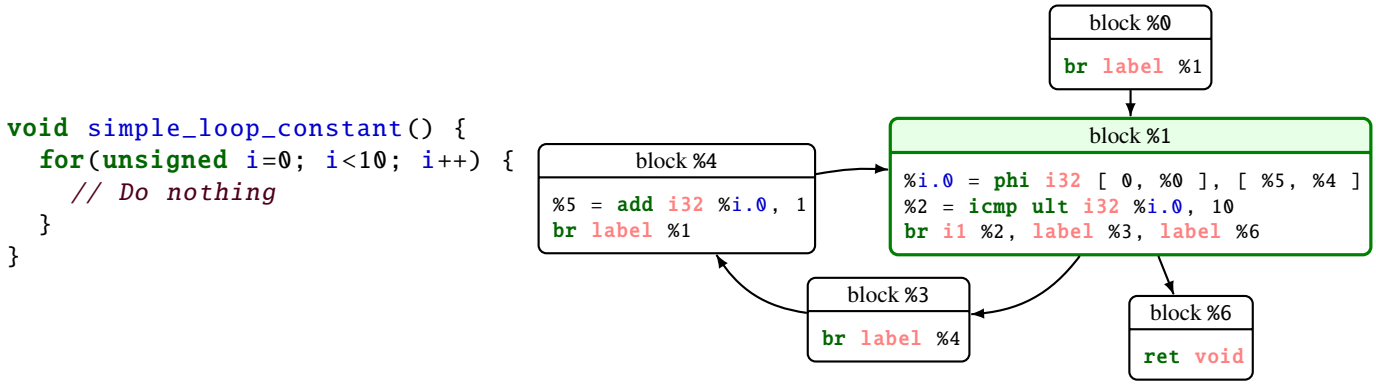


Figure 2: A C function and the LLVM representation

5.2 Algorithm modifications for SSA

Since SSA is not in the form of an automaton, as is NTS, the transition relation τ is not directly accessible. Moreover, the set of control states and variable, as part of the definition of an automaton Section 2.2 is not clearly specified for a program in SSA form.

In the LLVM IR, the transition relation is not defined on the edges but inside the nodes. For example $x' = x + 1$, in on one of the edges of Example 1, would be represented in the LLVM IR as an instruction of the form `%x.1 = add i32 %x.0, 1` in a basic block. Transition conditions are placed at the beginning and the end of a block as ϕ-functions and branching instructions.

To build the transition relation, we encode each instruction into an SMT formula. The transition function is the conjunction of all these formulas. This encoding has been proposed by Henry et al. [12] and is used in Pagai. Each LLVM variable is associated to an SMT variable and each basic block (resp. edge) is associated to a boolean SMT variable to symbolize the fact that the program goes or goes not through this block (resp. edge). We encode edge conditions by two means. On one hand, block terminators (conditional and unconditional jumps) are encoded by making the boolean associated to a block equal to the boolean of the edge taken. On the other hand, each block variable is equal to the disjunction of the incoming edges: if one of these edges is accessed, then the block is accessed.

There is an issue in this encoding: SMT variables at the beginning and the end of the loop are going to be the same, which is not desirable since our goal is to capture the progression during a loop. In order to avoid that, we decompose the graph into a DAG, as proposed by Monniaux and Gonnord [14]. We do so by breaking the strongly connected component of the graph[9]. Each node we break is separated into the up part, with the input edges and the ϕ-functions, and the down part, with the output edges and the rest of the instructions. The result of such separation applied to Figure 2 is shown Figure 3, the block %1 colored in green is decomposed into two parts. We also create different smt variables for the value of the variables before and after a loop. In Figure 3 i_0 represents the value before the loop and i'_0 after.

Figure 3 shows on the left the LLVM control flow graph after breaking up the strongly connected components. The translation to an SMT formula is shown on the right, represented as a graph for readability purposes. The final SMT formula is the conjunction of the formulas inside each nodes.

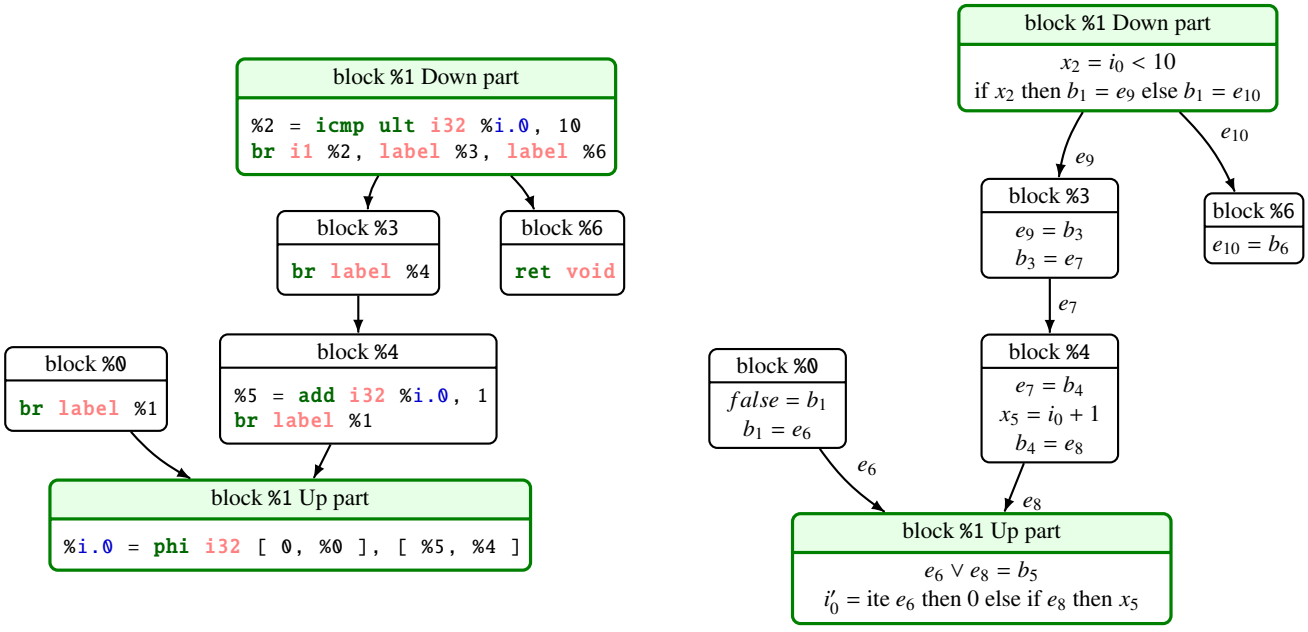


Figure 3: Follow up of Figure 2: breaking of the cycles and transformation to smt formula

Even if we have an SMT formula representing the transition relation, we still didn't specify properly the set of control points and the set of variables we considered. The set of control points $\mathcal{W}$ is the cutset used to break the strongly connected component. The set of loop headers is a valid cutset but not always a minimal one. With this definition of $\mathcal{W}$, a transition does not correspond to an edge in the control flow graph but is the complete trace from one control point to another, abstracting away the various possible path between two control points, minimizing the size of the problem considered.

We use the same set of variables as Pagai [12]: the set of *alive* variables that are not linear combination of other

[9]Even if finding a minimal vertex cutset is NP-complete in general, It is solvable in linear time on reducible graphs[19] such as control flow graphs. It is also approximable using a simple DFS.

variables. In other words, it's the minimal set of variables needed to describe the invariant polyhedron. For example, if $z = x + y$ and that z is alive, only x and y will be in the set of variables.

In the Figure 2, the only control point is the block `%1` represented in green. The set of variables is { `%i.0` }.

5.3 Architecture

The architecture of the software we developed and its interaction with external software is shown Figure 4. It can be separated into various parts.

- A library to read LLVM IR and produce the transition relation, which is called "LLVM to SMT" on the Figure 4.
- An interface to gather the invariants inferred by Pagai. Pagai takes as input an LLVM IR and produces a annotated LLVM IR containing the invariants, which we can then collect.
- An interface to an optimizing SMT solver. We choose Z3 as our optimizing SMT solver. Z3[10] is open source software developed by Microsoft which possess an OCaml API and an (experimental) optimizing SMT solver.
- An interface to a linear programming solver. There is no special requirement on the linear programming solver except that it must handle rationals. The current implementation uses the fact that the SMT solver can perform optimization to emulate an LP solver, but this may change later.
- The various algorithms presented in this report, which are gathered into a library called "Smt Terminate".

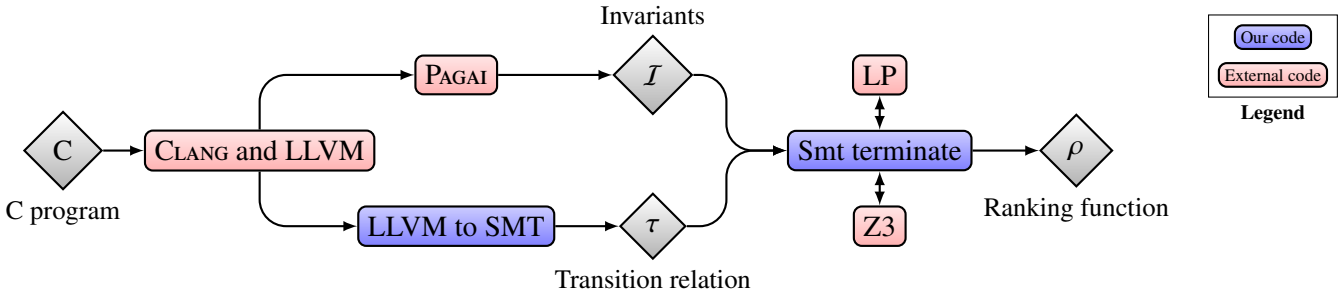


Figure 4: The complete architecture of our code and its interaction with external programs

6 Extensions and Future works

Several extensions of this work have already been proposed in [18].

Multidimensional ranking functions We only presented an algorithm for monodimensional ranking functions. It's possible to extend the second algorithm to compute multidimensional ranking functions by computing a family of linearly independent monodimensional ranking functions iteratively and stopping when a non-strict ranking function is encountered. This algorithm has not yet been implemented.

Multiple control points In this report, we restricted our approach to program with only one control points. It is possible to extend the proposed approach to handle multiple control points by considering a family of $\boldsymbol{l}$ indexed by the control points in $\mathcal{W}$. Even if the intuition is not much complicated, the encoding of the problem as an SMT formula becomes significantly more complicated. This has not yet been implemented either.

We plan to implement both algorithm in the near future.

Other extensions are possible :

1. Consider non-linear relations. This would allows to infer the ranking function for more programs but at the expense of the termination of the algorithm.
2. Use the same technique of counter-example guided inference to generate invariants.

[10] https://z3.codeplex.com/

7 Conclusion

We presented a method to infer ranking functions in an iterative fashion using extremal counter-examples. There has been ample literature on this on related topics (see [1], Sec. 6 and [3] for a large panel of proposed methods). Our approach differs in several ways :

1. Several existing approaches do not necessarily terminate. In contrast, our approach always terminates, and always outputs a linear ranking function if one exists.
2. Since our approach consider the transition lazily, the size of the LP problems considered are much smaller than other tools, allowing better performances.

This internship was the occasion for me to explore a different territory which I haven't previously experienced: static analysis and more precisely termination checking. It was very enlightening and gave me a better knowledge of the various tools used in static analysis, which I'm sure will be fruitful while working on programming languages.

References

[1] C. Alias, A. Darte, P. Feautrier, and L. Gonnord. Multi-dimensional rankings, program termination, and complexity bounds of flowchart programs. In *Static analysis (SAS)*, Perpignan, France, Sept. 2010. doi: 10.1007/978-3-642-15769-1. URL http://hal.inria.fr/inria-00523298.

[2] C. Ancourt, F. Coelho, and F. Irigoin. A modular static analysis approach to affine loop invariants detection. *Electronic Notes in Theoretical Computer Science*, 267(1):3 – 16, 2010. ISSN 1571-0661. doi: http://dx.doi.org/10.1016/j.entcs.2010.09.002.

[3] B. Cook, A. Podelski, and A. Rybalchenko. Proving program termination. *Commun. ACM*, 54(5):88–98, May 2011. ISSN 0001-0782. doi: 10.1145/1941487.1941509. URL http://doi.acm.org/10.1145/1941487.1941509.

[4] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 84–97. ACM, 1978.

[5] R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, and F. K. Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM transactions on programming languages and systems*, 13:451–490, 1991.

[6] P. Feautrier. Some efficient solutions to the affine scheduling problem, part I, one-dimensional time. *International Journal of Parallel Programming*, 21(5):313–348, Oct. 1992.

[7] P. Feautrier. Some efficient solutions to the affine scheduling problem, part II, multi-dimensional time. *International Journal of Parallel Programming*, 21(6):389–420, Dec. 1992.

[8] P. Feautrier and L. Gonnord. Accelerated Invariant Generation for C Programs with Aspic and C2fsm. In *Tools for Automatic Program AnalysiS (TAPAS'10)*, Perpignan, France, 2010. doi: 10.1016/j.entcs.2010.09.014. URL http://hal.inria.fr/inria-00523320.

[9] B. Gärtner and J. Matoušek. *Approximation Algorithms and Semidefinite Programming*. Springer, 2012. ISBN 978-3-642-22014-2. doi: 10.1007/978-3-642-22015-9.

[10] L. Gonnord and N. Halbwachs. Combining widening and acceleration in linear relation analysis. In *Static analysis (SAS)*, volume 4134 of *LNCS*, pages 144–160. Springer, Aug. 2006. doi: 10.1007/11823230_10.

[11] S. Gulwani and F. Zuleger. The reachability-bound problem. In *ACM symposium on programming language design and implementation (PLDI)*, pages 292–304. ACM, 2010. ISBN 978-1-4503-0019-3. doi: 10.1145/1806596.1806630.

[12] J. Henry, D. Monniaux, and M. Moy. Pagai: A path sensitive static analyser. *Electr. Notes Theor. Comput. Sci.*, 289:15–25, 2012. doi: 10.1016/j.entcs.2012.11.003.

[13] H. Hojjat, F. Konecný, F. Garnier, R. Iosif, V. Kuncak, and P. Rümmer. A verification toolkit for numerical transition systems - tool paper. pages 247–251, 2012.

[14] D. Monniaux and L. Gonnord. Using bounded model checking to focus fixpoint iterations. In E. Yahav, editor, *SAS*, volume 6887 of *Lecture Notes in Computer Science*, pages 369–385. Springer, 2011. ISBN 978-3-642-23701-0.

[15] R. Nieuwenhuis and A. Oliveras. On SAT modulo theories and optimization problems. In *SAT*, volume 4121 of *LNCS*, pages 156–169. Springer, 2006. ISBN 3-540-37206-7. doi: 10.1007/11814948_18.

[16] A. Podelski and A. Rybalchenko. A complete method for the synthesis of linear ranking functions. In *Verification, Model Checking and Abstract Interpretation (VMCAI'04)*, volume 2937 of *LNCS*, pages 239–251. Springer, 2004. ISBN 3-540-20803-8.

[17] R. Sebastiani and S. Tomasi. Optimization in SMT with $\mathcal{LA}(\mathbb{Q})$ cost functions. In *Proceedings of the 6th international joint conference on Automated Reasoning*, (IJCAR'12), pages 484–498. Springer, 2012. ISBN 978-3-642-31364-6. doi: 10.1007/978-3-642-31365-3_38.

[18] L. Seguinot, D. Monniaux, and L. Gonnord. Synthesis of ranking functions using extremal counterexamples. Technical Report xxx, Verimag Research Report, 2014.

[19] A. Shamir. A linear time algorithm for finding minimum cutsets in reducible graphs. *SIAM Journal on Computing*, (8): 645–655, 1979.