Synthesis of ranking functions using extremal counterexamples

Laure Gonnord, David Monniaux, Gabriel Radanne

Synthesis of ranking functions using extremal counterexamples*

Laure Gonnord
LIP, Univ. Lyon-1, France
Laure.Gonnord@ens-lyon.fr

David Monniaux
VERIMAG, CNRS, Grenoble, France
David.Monniaux@imag.fr

Gabriel Radanne
PPS, Univ. Paris. Diderot, France
gabriel.radanne@pps.univ-paris-diderot.fr

April 20, 2015

Abstract

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

Previous work would either synthesize a ranking function at every basic block head, not just loop headers, which reduces the scope of programs that may be proved to be terminating, or expand large block transitions including tests into (exponentially many) elementary transitions, prior to computing the ranking function, resulting in a very large global constraint system. In contrast, our algorithm incrementally refines a global linear constraint system according to extremal counterexamples: only constraints that exclude spurious solutions are included.

Experiments with our tool Termite show marked performance and scalability improvements compared to other systems.

1 Introduction

Program termination can be shown by providing a *ranking function*: a function from program states to a well-founded ordering, decreasing along any transition in the program.

In formal methods by assisted proofs (e.g. B method, Frama-C) the user typically has to provide a ranking function for each loop; automation may relieve that burden — but that is not all! In compilation, certain optimizations, such as *lazy code motion*, need to ensure that certain fragments of code terminate to be correct Tristan and Leroy [2009]; yet termination analysis is often considered too inefficient to be integrated into a compiler[1] — we show it needs not be. Also, from a ranking function and an invariant one may compute a bound (constant or parametric) on the number of iterations, which is necessary for *worst-case execution time* (WCET)[2] or for *high-level synthesis* ("C-to-silicon"). These applications need a fully automated, efficient algorithm providing the ranking function in most cases.

Because finding a ranking function is equivalent to 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 $\boldsymbol{x} \in \mathbb{Z}^n$ of program variables to an integer linear combination $\sum_i \alpha_i \boldsymbol{x}_i$ such that $\rho(\boldsymbol{x}') < \rho(\boldsymbol{x})$ for any possible program step $\boldsymbol{x} \to \boldsymbol{x}'$, and such that $\rho(\boldsymbol{x}) \geq 0$ for any reachable program state (remark that this entails proving an auxiliary invariant property). Furthermore, the α_i

*The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement nr. 306595 "STATOR"

[1] Between the 1999 and 2011 C standards an exception (§6.8.5:6) was added to allow compilers to assume that every non trivially infinite computation loop always terminates — a brutal way to avoid termination analysis!

[2] Tools such as Absint's aiT (http://www.absint.com/ait/) and OTAWA (http://otawa.fr/OTAWA) need loop bounds, by user annotation if necessary.

may be allowed to depend on the program point k, thus the unknowns are $\alpha_{i,k}$: the decreasing condition becomes $\sum_i \alpha_{i,k'} \boldsymbol{x}'_i < \sum_i \alpha_{i,k} \boldsymbol{x}_i$ for any transition $(k, \boldsymbol{x}) \to (k', \boldsymbol{x}')$.

Various methods for the automated synthesis of such functions have been proposed Podelski and Rybalchenko [2004]; they build a constraint system in the unknowns $\alpha_{i,k}$ and solve it. This class is extended to *lexicographic linear ranking functions*: instead of a single function $\rho(\boldsymbol{x})$, one uses a tuple of them $\langle \rho_1(\boldsymbol{x}), \ldots, \rho_m(\boldsymbol{x}) \rangle$, which is shown to be strictly decreasing with 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 Alias et al. [2010], Ben-Amram and Genaim [2014].

A common weakness of many existing approaches is the need to solve for linear coefficients at all basic blocks, which leads to scalability and precision issues.

First, 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 might be too big Alias et al. [2010]. Thus, it is desirable to limit the set of points to consider to heads of loops, or, more generally, a *cut-set* of K' program points — a set such that removing these points cuts all cycles in the program Shamir [1979].

Second, considering each transition separately might also lead to a lack of precision: a loop might have a linear lexicographic ranking function that decreases along each of its paths as a whole, but none that decreases *at each step* of these paths, e.g., Listing 1. Thus, following Gulwani and Zuleger [2010], we will treat each path inside a loop as a single transition. 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. We propose a new approach which avoids the explicit computation of (all) the paths.

Contribution We present an approach for computing lexicographic linear ranking functions when all paths inside a loop are treated separately. We lazily build a constraint system according to extremal counterexamples (a counterexample is a path where a candidate ranking function increases) — a naive approach with arbitrary counterexamples could fail to terminate. We implemented our approach and benchmarked it against existing analyzers.

Contents This article 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). Section 5 and section 6 discuss extensions to multidimensional ranking functions and multi control points. Section 7 discusses coNP-completeness and worst-case complexity. Section 8 gives some possible extensions w.r.t. the expressiveness of the method. Section 9 discusses implementation and experimental results. Finally, Section 10 highlights the improvements brought by our contribution.

Related Work The use of ranking functions (a nonnegative strictly decreasing integer quantity) for proving the termination of programs goes back to Turing [1949]. We thus will not attempt to give a complete timeline of the topic; Ben-Amram and Genaim [2014] present a longer survey.

In proof assistants (such as e.g. the B method or PVS), it is customary to require from the user such a function for proving the termination of loops or recursive calls. Linear ranking functions are one of the simplest kind to generate automatically; yet not every interesting numerical program can be proved to terminate with such a function. Polynomial ranking functions are more powerful, but considerably more difficult to handle.

In recent years, two approaches have attracted particular attention. *Lexicographic linear ranking functions* Alias et al. [2010], Cook et al. [2013], Bradley et al. [2005b,a], Ben-Amram and Genaim [2014] are both much more powerful than linear ranking functions (e.g. one can easily prove the termination of the Ackermann-Péter function) and not much harder to obtain. A natural extension of this framework is, instead of requiring a single, monolithic ranking function at all program points, to make it dependent on the program point. Other extensions include *piecewise* linear ranking functions [Urban, 2013] and ordinal-value ranking functions [Urban and Miné, 2014], which subsume lexicographic orders.

Another approach is based on Ramsey's theorem Codish and Genaim [2003], the idea being that a program is non-terminating if and only if there exists some feasible non-terminating loop in the control

flow. As noted in Cook et al. [2013], while some of the reasoning is local (one loop at a time), the method operates over the transitive closure of program transitions, which is not easy to approximate finely.

The problem of synthesizing a nonnegative linear function over a convex polyhedron satisfying certain properties was studied in the context of scheduling. Feautrier [1992a] proposed applying Farkas' lemma; others suggested an approach based on generators (vertices), as ours. A comparison of the two dual approaches (constraints vs generators) showed that constraints scaled better for scheduling problems Balev et al. [1998]. In contrast, Ben-Amram and Genaim [2014] used the vertices of the integer hull of the transition relation. We improve on this approach by lazily enumerating the generators, without explicitly computing this convex hull.

2 Preliminary Definitions

In this section, we will define the concepts used in the rest of the article, following the notations of Ben-Amram and Genaim [2014]. 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

Schrijver [1998] presents this topic in detail.

Definition 1 (Polyhedra)**.** *A rational convex polyhedron $\mathcal{P} \in \mathbb{Q}^n$ is the set of solutions of a set of inequalities $A\boldsymbol{x} \leq \boldsymbol{b}$ where $A \in \mathbb{Q}^{m\times n}$ is a rational matrix of n columns and m rows, $\boldsymbol{x} \in \mathbb{Q}^n$ and $\boldsymbol{b} \in \mathbb{Q}^m$ are columns vectors of n and m rational values respectively. We denote by Constraints($\mathcal{P}$) the set of constraints of $\mathcal{P}$.*

Definition 2 (Integer Hull)**.** *For a given polyhedron $\mathcal{P} \in \mathbb{Q}^n$ the set of integer points of $\mathcal{P}$ is denoted by $I(\mathcal{P})$.*

Definition 3 (Generator representation)**.** *The* vertices *and* rays *(if unbounded) of a closed convex polyhedron form a system of* generators*:*

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

In the following, the expressions "convex polyhedron" and "polyhedron" refer to rational closed convex polyhedra as defined above.

2.2 Transitions and Invariants

We consider programs over a state space $\mathcal{W} \times \mathbb{Z}^n$, where $\mathcal{W}$ is the finite set of control states, defined by an initial state and a transition relation τ.

$\mathcal{W}$ needs not be the set of syntactic control points in the program (e.g. one per instruction, or one per basic block): it is sufficient that they form a *cut-set* of the syntactic control states, that is, a set of points such that removing them cuts all cycles in the program. This ensures that, for any $k, k' \in \mathcal{W}$, the transition relation $(k, \boldsymbol{x}) \rightarrow_\tau (k, \boldsymbol{x}')$ can be expressed in the same theory as the individual transitions with a formula of linear size, using propositional variables to encode execution paths, as commonly practiced in predicate abstraction and other analysis methods using satisfiability modulo theory Monniaux and Gonnord [2011]. In block-structured programs the set of loop headers is a cut-set, and a minimal cut-set may be computed in linear time Shamir [1979].

Let us thus 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 equations, excluding negation and strict inequalities. τ is then equivalent to a union U of closed convex polyhedra; intuitively each disjunct of $\{(\boldsymbol{x}, \boldsymbol{x}') \mid (k, \boldsymbol{x}, k', \boldsymbol{x}') \in \tau\}$ corresponds to a path from k to k' in the program.

Definition 4 (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 $(k, \boldsymbol{x})$. We will note invariants as $\mathcal{I}_k$ (or $\mathcal{I}$ when the control point is implicit).*

Definition 5 (Constraints)**.** *Let us consider an invariant $\mathcal{I}$ as a nonempty closed convex polyhedron whose integer hull is given by a set of inequalities $\mathcal{I} = \{\boldsymbol{x} \mid \bigwedge_{i=1}^{m} \boldsymbol{a}_i.\boldsymbol{x} \geq b_i\}$. Constraints($\mathcal{I}$) denotes $\{\boldsymbol{a}_i, i \in [1, m]\}$*

We assume that some external tool provides us with invariants: either affine invariants provided by e.g. polyhedral analysis Cousot and Halbwachs [1978] or its refinements as in the Aspic[3] Gonnord and Halbwachs [2006], Feautrier and Gonnord [2010] or the Pagai Monniaux and Gonnord [2011], Henry et al. [2012b, 2014] tools. In section 8 we shall relax this requirement.

Φ denotes a set of quadruples $(k, \boldsymbol{x}, k', \boldsymbol{x}')$, where k, k' are control states and $\boldsymbol{x}, \boldsymbol{x}'$ vectors of numeric variables, such that for any possible transition $(k, \boldsymbol{x}) \rightarrow_\tau (k', \boldsymbol{x}')$ of the program, $(k, \boldsymbol{x}, k', \boldsymbol{x}') \in \Phi$. Such a Φ may be obtained from invariants as follows: $(k, \boldsymbol{x}, k', \boldsymbol{x}') \in \Phi \iff \mathcal{I}_k(\boldsymbol{x}) \wedge (k', \boldsymbol{x}) \rightarrow_\tau (k, \boldsymbol{x}')$.

We only require that Φ may be expressed in a decidable theory in which it is possible to perform linear optimization; e.g. linear integer arithmetic, combined or not with uninterpreted functions; this is a form of *large block encoding*. Again, this includes the case considered by Ben-Amram and Genaim [2014]: for k, k' fixed, the projection to $\boldsymbol{x}, \boldsymbol{x}'$ can be expressed as a union of polyhedra.

There is a crucial difference between our approach and that of Ben-Amram and Genaim [2014, §2.3–2.4] with respect to complexity. They consider an explicit list of polyhedra, which may contain 2^n polyhedra if the program from location k to location k' consists in n successive if-then-else statement: in essence, they take Φ in *disjunctive normal form*. In contrast, in this article we seek to enumerate vertices of the convex hull of Φ lazily, never computing its disjunctive normal form.

In addition to allowing disjunctions $\vee$ inside the formula (not just at the outer level), we allow existential quantifiers $\exists$. Such a formula is equivalent to one of same size and in prenex form where all the variables not in $\boldsymbol{x}, \boldsymbol{x}'$ are existentially quantified. Again, this is important for complexity: even for a conjunction of linear constraints, eliminating a block of existential quantifiers (that is, projecting the polyhedron) may result in exponential blow-up.

2.3 Ranking Functions

Definition 6 (Linear ranking function)**.** *A (strict)* lexicographic linear ranking function *of dimension m is a function $\rho : \mathcal{W} \times \mathbb{Z}^n \to \mathbb{Z}^m$, such that*

1. *for any state $k \in \mathcal{W}$, $\boldsymbol{x} \mapsto \rho(k, \boldsymbol{x})$ is affine;*
2. *for any $(k, \boldsymbol{x}, k', \boldsymbol{x}') \in \Phi$, $\rho(k', \boldsymbol{x}') \prec \rho(k, \boldsymbol{x})$ 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 $x_i < y_i$;*
3. *for any state $(k, \boldsymbol{x})$ in the invariant $\mathcal{I}$, all coordinates of $\rho(k, \boldsymbol{x})$ are nonnegative.*

It is said to be monodimensional*, or a* linear ranking function*, if $m = 1$, and* multidimensional *otherwise.*

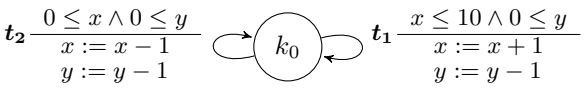
We note ranking functions: $\rho(k, \boldsymbol{x}) = \boldsymbol{\lambda}^k \cdot \boldsymbol{x} + \lambda_0^k$ or $\rho(\boldsymbol{x}) = \boldsymbol{\lambda} \cdot \boldsymbol{x} + \lambda_0$ when the control point is implicit.

Definition 7 (Quasi lexicographic linear ranking function)**.** *A* quasi lexicographic linear ranking function *replaces Condition 2 by $\rho(k', \boldsymbol{x}') \preceq \rho(k, \boldsymbol{x})$ where $\preceq$ is the lexicographic ordering.*

The existence of a strict (lexicographic) linear ranking function entails the termination of the program.

In the following, we deal with a unique control point k with $\mathcal{I}$ its invariant and τ its transition relation[4]. We will consider the situation with multiple control points section 6.

Example 1 (Monodimensional linear ranking function)**.** *Consider the following transition relation on $\mathbb{Q}^2$, where transitions are specified by ($\frac{guard}{action}$):*



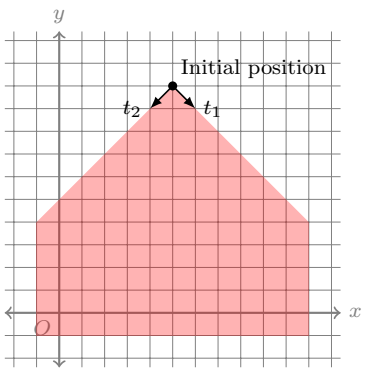


Figure 1: Invariant polyhedron for Example 1

*Under initial assumptions $x = 5, y = 10$, an invariant generator (*Aspic*) gives the following inductive invariant for $\mathcal{I}$ (Figure 1):*

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

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

$$\begin{array}{lccccc} \boldsymbol{a} = & \binom{1}{0} & \binom{-1}{0} & \binom{0}{1} & \binom{1}{-1} & \binom{1}{1} \\ \boldsymbol{b} = & 1 & 11 & 1 & 5 & 15 \end{array}$$

A possible monodimensional strict linear ranking function for this automaton is $\rho(x, y) = y + 1$: this expression is always nonnegative on $\mathcal{I}$, and both t_1 and t_2 make this expression strictly decrease.

2.4 Cones

We first recall the definition of convex cones and orthogonality [Gärtner and Matoušek, 2012, §4.2].

Definition 8. *A convex cone of $\mathbb{Q}^n$ is a subset Δ of $\mathbb{Q}^n$ such that : i) For all $\boldsymbol{x} \in \Delta$, $\alpha > 0$, $\alpha\boldsymbol{x}$ is also in Δ. ii) For all $\boldsymbol{x}, \boldsymbol{y} \in \Delta$, $\boldsymbol{x} + \boldsymbol{y}$ is also in Δ.*

Definition 9. *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.*

Proposition 1. *The quasi ranking functions form a closed convex cone.*

3 Quasi Monodimensional Ranking Functions of Maximal Termination Power

We shall now see that, for a program with one control state, finding a quasi monodimensional ranking function amounts to finding a vector in the intersection of the cone generated by the constraints of an inductive invariant (to ensure positiveness) and the orthogonal of the cone $\{\boldsymbol{x} - \boldsymbol{x}' \mid (\boldsymbol{x}, \boldsymbol{x}') \in \tau\}$ where τ is the transition relation. This suggests a naive (but wrong) algorithm, which incrementally constructs the desired cone as the solution set of a constraint system, generated through a counterexample-guided approach.

3.1 Quasi Ranking Functions in Terms of Cones

We assume here that $\Phi = \{(\boldsymbol{x}, \boldsymbol{x}') \mid \boldsymbol{x} \in \mathcal{I} \wedge (\boldsymbol{x}, \boldsymbol{x}') \in \tau\}$ where $\mathcal{I}$ is a (nonempty) invariant defined by $\mathcal{I} = \{\boldsymbol{x} \mid \bigwedge_{i=1}^{m} \boldsymbol{a}_i.\boldsymbol{x} \geq b_i\}$. A quasi ranking function is considered to be a linear affine function that is nonincreasing when a τ step is taken, and stays nonnegative on $\mathcal{I}$. The rest of the section recalls some results obtained in the field of polyhedral scheduling Feautrier [1992a,b].

[3] http://laure.gonnord.org/aspic/

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

Proposition 2. *Under these assumptions, $\rho = \boldsymbol{\lambda} \cdot \boldsymbol{x} + \lambda_0$ is a quasi ranking function if and only if the following conditions hold:*

$$\forall \boldsymbol{x}, \boldsymbol{x'}\ \boldsymbol{x} \in \mathcal{I} \wedge (\boldsymbol{x}, \boldsymbol{x'}) \in \tau \implies \boldsymbol{\lambda} \cdot (\boldsymbol{x} - \boldsymbol{x'}) \geq 0 \tag{1}$$

$$\exists \gamma_1 \geq 0, \ldots, \gamma_m \geq 0,\ \boldsymbol{\lambda} = \sum_{i=1}^{m} \gamma_i \boldsymbol{a_i} \tag{2}$$

Proof. First, Equation 1 is obtained by rewriting the decreasing condition using linearity. Secondly, Farkas' Lemma Schrijver [1998] applied to the nonempty polyhedron $\mathcal{I}$, where vector inequality $\rho(k, \boldsymbol{x}) \geq 0$ holds, yields:

$$\exists \gamma_1 \geq 0, \ldots, \gamma_m \geq 0,\ \boldsymbol{\lambda} = \sum_{i=1}^{m} \gamma_i \boldsymbol{a}_i \wedge \lambda_0 \leq \sum_{i=1}^{m} \gamma_i b_i$$

Once the condition $\boldsymbol{\lambda} = \sum_{i=1}^{m} \gamma_i \boldsymbol{a}_i$ is met then an appropriate choice of λ_0 can always be made, thus Equation 2 is verified. □

Equation 2 means that $\boldsymbol{\lambda}$ belongs to the cone generated by the $\boldsymbol{a}_i$ constraints, denoted as $Cone_{constraints}(\mathcal{I})$ from now on. Equation 1 is also equivalent to stating that $\boldsymbol{\lambda}$ belongs to some (other) cone, as we will see in the following. 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.[5] Equation 1 is then equivalent to stating that $\boldsymbol{\lambda}$ belongs to the orthogonal of the convex cone generated by $\mathcal{P}_{\mathcal{I},\tau}$.

Let us reexamine the condition that $\boldsymbol{\lambda} \cdot \boldsymbol{u} \geq 0$ 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 Definition 3: the condition $\boldsymbol{\lambda} \cdot \boldsymbol{u} \geq 0$ for all $\boldsymbol{u} \in \mathcal{P}^H_{\mathcal{I},\tau}$ is then replaced by $\forall i\ \boldsymbol{\lambda} \cdot \boldsymbol{v}_i \geq 0 \wedge \forall i\ \boldsymbol{\lambda} \cdot \boldsymbol{r}_i \geq 0$ For the sake of simplicity, we now group these three conditions into one: there is a finite set $V = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m\}$ such that Equation 1 is equivalent to $\forall 1 \leq i \leq m,\ \boldsymbol{\lambda} \cdot \boldsymbol{v}_i \geq 0$, otherwise said $\boldsymbol{\lambda}$ belongs to the polyhedral convex cone with faces defined by the $\boldsymbol{v}_i$, denoted by $Cone(V_{\mathcal{I},\tau})$.

We have expressed both conditions of Definition 6 as membership of $\boldsymbol{\lambda}$ in polyhedral cones, respectively given by constraints and generators:

Proposition 3. *Let $V_{\mathcal{I},\tau}$ a set of generators of $\mathcal{P}^H_{\mathcal{I},\tau}$. Then $\rho(\boldsymbol{x}) = \boldsymbol{\lambda} \cdot \boldsymbol{x} + \lambda_0$ is a quasi ranking function iff $\boldsymbol{\lambda} \in Cone_{constraints}(\mathcal{I}) \cap Cone(V_{I,\tau})$.*

Thanks to this proposition, we can restrict the search of a ranking function to vectors that are linear combinations of $Constraints(\mathcal{I})$ and of $V_{\mathcal{I},\tau}$, which we will do in Definition 11.

3.2 Maximal "Termination Power" and Strict Ranking Functions

We are however interested in more than *quasi* ranking functions (otherwise, $\boldsymbol{\lambda} = \boldsymbol{0}$ would do!). In the simplest cases, we would like $\forall 1 \leq i \leq m,\ \boldsymbol{\lambda} \cdot \boldsymbol{v}_i > 0$: by appropriate scaling of $\boldsymbol{\lambda}$, we can ensure that $\forall 1 \leq i \leq m,\ \boldsymbol{\lambda} \cdot \boldsymbol{v}_i \geq 1$ and thus $\boldsymbol{\lambda}$ is a ranking function, proving termination. Remark that this condition $\forall i\ \boldsymbol{\lambda} \cdot \boldsymbol{v}_i > 0$ is otherwise expressed by "$\boldsymbol{\lambda}$ 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 10. *$\boldsymbol{\lambda}$ is said to have* maximal termination power *if $\boldsymbol{\lambda} \cdot \boldsymbol{v}_i > 0$ for as many $\boldsymbol{v}_i$ as possible. In the framework of Alias et al. [2010], this would be equivalent to maximizing the number of transitions where ρ decreases strictly.*

The notations and results of this section are also adapted from Feautrier [1992a,b].

For $\boldsymbol{\lambda} \in V^{\perp}$, we define the set $\pi_{\mathcal{V}}(\boldsymbol{\lambda}) = \{\boldsymbol{v} \in V \mid \boldsymbol{\lambda} \cdot \boldsymbol{v} > 0\}$; we would like $\rho = \boldsymbol{\lambda} \cdot \boldsymbol{x} + \lambda_0$ of maximal $|\pi_{\mathcal{V}}(\boldsymbol{\lambda})|$.

[5] Ancourt et al. [2010] use it to compute loop invariants.

[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 $\{(1,x) \mid x \in \mathbb{Q}\}$ is $\{(0,0)\} \cup (0,1] \times \mathbb{Q}$, which is not closed.

Proposition 4. *Given $\mathcal{V} = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_N\}$ a set of vectors and $\boldsymbol{\lambda}$ a vector, $\pi_{\mathcal{V}}(\boldsymbol{\lambda})$ is maximal with respect to cardinality if and only if it is the maximum with respect to inclusion, i.e.*

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

Proof. It is obvious that for all $\boldsymbol{\lambda}'$, if $\pi_{\mathcal{V}}(\boldsymbol{\lambda}') \subseteq \pi_{\mathcal{V}}(\boldsymbol{\lambda})$ then $|\pi_{\mathcal{V}}(\boldsymbol{\lambda}')| \leq |\pi_{\mathcal{V}}(\boldsymbol{\lambda})|$. Let us now prove the other implication. Let $\boldsymbol{\lambda}$ be a vector such that $\forall \boldsymbol{\lambda}', |\pi_{\mathcal{V}}(\boldsymbol{\lambda}')| \leq |\pi_{\mathcal{V}}(\boldsymbol{\lambda})|$ and $\boldsymbol{\lambda}'$ a vector. Let us suppose that $\pi_{\mathcal{V}}(\boldsymbol{\lambda}') \not\subseteq \pi_{\mathcal{V}}(\boldsymbol{\lambda})$, then $\pi_{\mathcal{V}}(\boldsymbol{\lambda}') \backslash \pi_{\mathcal{V}}(\boldsymbol{\lambda})$ is not empty. Since $\pi_{\mathcal{V}}(\boldsymbol{\lambda}) \cup \pi_{\mathcal{V}}(\boldsymbol{\lambda}') \subseteq \pi_{\mathcal{V}}(\boldsymbol{\lambda} + \boldsymbol{\lambda}')$, $|\pi_{\mathcal{V}}(\boldsymbol{\lambda} + \boldsymbol{\lambda}')| \geq |\pi_{\mathcal{V}}(\boldsymbol{\lambda}) \cup \pi_{\mathcal{V}}(\boldsymbol{\lambda}')| = |\pi_{\mathcal{V}}(\boldsymbol{\lambda})| + |\pi_{\mathcal{V}}(\boldsymbol{\lambda}') \backslash \pi_{\mathcal{V}}(\boldsymbol{\lambda})| > |\pi_{\mathcal{V}}(\boldsymbol{\lambda})|$, which is absurd since $|\pi_{\mathcal{V}}(\boldsymbol{\lambda})|$ is assumed to be maximal. Thus, $\forall \boldsymbol{\lambda}', \pi_{\mathcal{V}}(\boldsymbol{\lambda}') \subseteq \pi_{\mathcal{V}}(\boldsymbol{\lambda})$. □

From now, the problem of finding a ranking function of maximal termination power is thus reduced to maximising an affine objective function on a set of affine constraints. We thus define a family of Linear Programming instances:

Definition 11. *Given $V = \{\boldsymbol{v}_j | 1 \leq j \leq N\}$ a set of generators of (the convex hull of) $\mathcal{P}_{\mathcal{I},\tau}$ and $Constraints(\mathcal{I}) = \{\boldsymbol{a}_i | 1 \leq i \leq m\}$ a set of vectors (constraints of $\mathcal{I}$), we denote by $LP(V, Constraints(\mathcal{I}))$ the following linear programming instance where γ_i and δ_i are the unknowns:*

$$\begin{cases} \text{Maximize } \sum_i \delta_i \text{ s.t.} & \\ \gamma_1, \ldots, \gamma_m \geq 0 & \\ 0 \leq \delta_j \leq 1 & \text{for all } 1 \leq j \leq N \\ \sum_{i=1}^{m} \gamma_i (\boldsymbol{v}_j . \boldsymbol{a}_i) \geq \delta_j & \text{for all } 1 \leq j \leq N \end{cases}$$

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

Proposition 5. *Let $V = \{\boldsymbol{v}_j | 1 \leq j \leq N\}$ be a set of generators of the convex hull of $P_{\mathcal{I},\tau}$ and $Constraints(\mathcal{I}) = \{\boldsymbol{a}_i | 1 \leq i \leq m\}$ the constraints of $\mathcal{I}$. Then :*

- *$LP(V, Constraints(\mathcal{I}))$ is always feasible.*
- *$LP(V, Constraints(\mathcal{I}))$ gives γ_is such that $\rho(\boldsymbol{x}) = \boldsymbol{\lambda} \cdot \boldsymbol{x} + \lambda_0$ with $\boldsymbol{\lambda} = \sum_{i=1}^{m} \gamma_i \boldsymbol{a}_i$ and $\lambda_0 = \sum_{i=1}^{m} \gamma_i b_i$ is a quasi ranking function of maximal $\pi_{\mathcal{V}}$.*

Proof.

- Let us start by noticing that $(\boldsymbol{\gamma}, \boldsymbol{\delta}) = (\mathbf{0}, \mathbf{0})$ is a solution of the inequalities, and $\sum_i \delta_i \leq N$, thus the set $\{\sum_i \delta_i | (\boldsymbol{\gamma}, \boldsymbol{\delta}) \text{ is a solution of LP}\}$ has a least upper bound.

  Moreover, in the optimum of $LP(V, Constraints(\mathcal{I}))$, each δ_j is either 0 or 1 : if $0 < \delta_j < 1$, then by scaling up $\boldsymbol{\lambda}$ by a factor $1/\delta_j$ we obtain a solution ρ with $\delta'_j = 1$. Therefore, the least upper bound is a maximum. It ensues that $LP(V, Constraints(\mathcal{I}))$ is always feasible.

- All solutions of the inequalities are quasi ranking functions, thanks to Proposition 3.

- $\delta_i = 1$ iff $\boldsymbol{v}_i \in \pi_{\mathcal{V}}(\boldsymbol{\lambda})$, thus $|\pi_{\mathcal{V}}(\boldsymbol{\lambda})| = \sum_i \delta_i$.

Note that $\delta_j = 0$ means that all solutions to the problem satisfy $\boldsymbol{\lambda} \cdot \boldsymbol{v}_j = 0$, a flatness condition that we shall discuss later. □

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 Ben-Amram and Genaim [2014], compute their generators $(\boldsymbol{x}, \boldsymbol{x}')$ and collect all resulting $\boldsymbol{x} - \boldsymbol{x}'$; computing the DNF has exponential cost even if not all disjuncts are needed. The new approach described in this article will only compute extremal points as needed, as opposed to eagerly expanding the DNF.

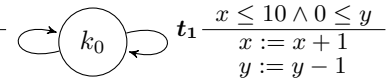
3.3 A Wrong but Intuitive Algorithm

Let us propose a simple but somewhat wrong incremental algorithm, which will help understand the correct algorithm presented later. $\mathcal{C}$ is a set of vectors and $\rho(\boldsymbol{x}) = \boldsymbol{\lambda} \cdot \boldsymbol{x} + \lambda_0$ a candidate ranking function.

1. Initialize $\mathcal{C} := \emptyset$, $\rho(\boldsymbol{x}) := \mathbf{0}$, thus $\boldsymbol{\lambda} = \mathbf{0}$.

2. Find a transition with an $\boldsymbol{u} = \boldsymbol{x} - \boldsymbol{x'}$ in $\mathcal{P}^H_{\mathcal{I},\tau}$ which contradicts that ρ is a *strict* ranking function. More precisely:
   (a) Ask an SMT-solver if $\exists \boldsymbol{u}$ s.t. $\mathcal{I} \wedge \tau \wedge \boldsymbol{\lambda} \cdot \boldsymbol{u} \leq 0$.
   (b) If $Unsat$, ρ is a *strict* ranking function. Return it.
   (c) If Sat, we get $\boldsymbol{u}$ from the SMT model.
3. Add $\boldsymbol{u}$ to $\mathcal{C}$.
4. Call $LP(\mathcal{C}, Constraints(\mathcal{I}))$ (see Definition 11) to compute a new ranking function that maximises the number of $\boldsymbol{v} \in \mathcal{C}$ such that $\rho(\boldsymbol{v}) > 0$ and go back to step 2.

Example 2 (Example 1, cont.)**.** *We already computed the generators $\boldsymbol{a}_i$ for $\mathcal{I}$.*



$$\tau = \left\{ (x, y, x', y') \;\middle|\; \begin{array}{c} x \leq 10 \wedge 0 \leq y \wedge x' = x + 1 \wedge y' = y - 1 \\ \vee \\ 0 \leq x \wedge 0 \leq y \wedge x' = x - 1 \wedge y' = y - 1 \end{array} \right\}$$

Recall that we are looking for $\rho(\boldsymbol{x}) = \boldsymbol{\lambda} \cdot \boldsymbol{x} + \lambda_0$ with $\boldsymbol{\lambda}$ a positive linear combination of the $\boldsymbol{a}_i$s.
We will follow the algorithm described subsection 3.3 step by step, repeated steps will be noted with '.

1. *Beginning with $\mathcal{C} = \{\}$ and $\rho(\boldsymbol{x}) = 0$, that is: $\boldsymbol{\lambda} = \boldsymbol{0}$ and $\lambda_0 = 0$. The unknown vector $\boldsymbol{u}$ is always $\binom{x-x'}{y-y'}$.*

First iteration.

2. *Sat $(\mathcal{I} \wedge \tau \wedge \boldsymbol{0} \cdot \boldsymbol{u} \leq 0)$?*
Yes and we have the model $\boldsymbol{u} = \binom{-1}{1}$

3. $\mathcal{C} \leftarrow \left\{ \binom{-1}{1} \right\}$

4. *Call $LP(\mathcal{C}, Constraints(\mathcal{I})) =$*

$$\left\{ \begin{array}{l} \textit{Maximize } \delta_1 \textit{ s.t.} \\ \gamma_1, \gamma_2, \gamma_3, \gamma_4, \gamma_5 \geq 0 \\ 0 \leq \delta_1 \leq 1 \\ -\gamma_1 + \gamma_2 + \gamma_3 - 2\gamma_4 \geq \delta_1 \end{array} \right.$$

(The last constraint comes from $\boldsymbol{u} \cdot \boldsymbol{a}_i$ for each i.)
The solver answers $\gamma_2 = 1$, $\gamma_1 = \gamma_3 = \gamma_4 = \gamma_5 = 0$.
We deduce $\boldsymbol{\lambda} = \boldsymbol{a}_2 = \binom{-1}{0}$ and $\lambda_0 = b_2 = 11$.

Second iteration.

2'. *Sat $\left(\mathcal{I} \wedge \tau \wedge \binom{-1}{0} \cdot \boldsymbol{u} \leq 0\right)$?*
Yes and we have the model $\boldsymbol{u} = \binom{1}{1}$

3'. $\mathcal{C} \leftarrow \left\{ \binom{-1}{1}; \binom{1}{1} \right\}$

4'. *Call $LP(\mathcal{C}, Constraints(\mathcal{I})) =$*

$$\left\{ \begin{array}{l} \textit{Maximize } \delta_1 + \delta_2 \textit{ s.t.} \\ \gamma_1, \gamma_2, \gamma_3, \gamma_4, \gamma_5 \geq 0 \\ 0 \leq \delta_1, \delta_2 \leq 1 \\ -\gamma_1 + \gamma_2 + \gamma_3 - 2\gamma_4 \geq \delta_1 \\ \gamma_1 - \gamma_2 + \gamma_3 - 2\gamma_5 \geq \delta_2 \end{array} \right.$$

The solver answers $\gamma_3 = 1$, $\gamma_1 = \gamma_2 = \gamma_4 = \gamma_5 = 0$. We deduce $\boldsymbol{\lambda} = \boldsymbol{a}_3 = \binom{0}{1}$ and $\lambda_0 = b_3 = 1$.

Third iteration.

2". *Sat$(\mathcal{I} \wedge \tau \wedge y - y' \leq 0)$? No, we stop.*

Return. *We have $\boldsymbol{\lambda} = \binom{0}{1}$ and $\lambda_0 = 1$. We obtain $\rho(x, y) = y + 1$, a strict ranking function for $(\tau, \mathcal{I})$.*
Let us point out the fact that the two models $\boldsymbol{u}$ we obtained correspond to the two transitions $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$.

3.4 Termination Problems

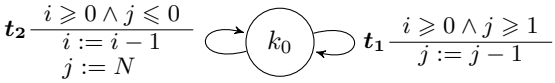
Unfortunately, this simple algorithm suffers from two problems which may prevent its termination (see Example 3).

The set of counterexamples is infinite 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 $\boldsymbol{u}$ 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 $\boldsymbol{\lambda}\cdot\boldsymbol{u}$, as in "optimization modulo theory" Nieuwenhuis and Oliveras [2006], Sebastiani and Tomasi [2012], 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 ($\boldsymbol{\lambda}\cdot\boldsymbol{v} > 0$ for all $\boldsymbol{v} \in \mathcal{V}$). Indeed, termination is ensured by the algorithm never choosing twice the same $\boldsymbol{u} = \boldsymbol{x} - \boldsymbol{x'}$, which is the case if there exists a (quasi) ranking function such that $\boldsymbol{\lambda}\cdot\boldsymbol{u} > 0$. But what if the SMT-solver picks $\boldsymbol{u}$ such that *all* quasi ranking functions $\boldsymbol{\lambda}$ satisfy $\boldsymbol{\lambda}\cdot\boldsymbol{u} = 0$? In this case, the algorithm may not terminate, always picking the same $\boldsymbol{u}$.

Example 3. *The algorithm does not terminate on this automaton:*



Indeed, on the transition $\boldsymbol{t_2}$, $j - j' \leq -N$, *and there is no constraint on* N. *Thus, if* $\boldsymbol{x} = (i, j, N)$ *denoted the vector of variables,* $\boldsymbol{r} = (0, -1, 0)$ *is a ray of* $\mathcal{P}^H_{\mathcal{I},\tau}$. *As long as* $\boldsymbol{\lambda}\cdot\boldsymbol{r} < 0$, *the SMT solver can return an infinite number of models as* $\mathcal{I} \wedge \tau \wedge \boldsymbol{\lambda}\cdot(\boldsymbol{x} - \boldsymbol{x'}) \leq 0$ *is always satisfiable.* □

4 Corrected Mono-dimensional Algorithm

Let us now address the termination issues raised in subsection 3.4. We will first assume, as said then, that we use an optimizing SMT solver to minimize $\boldsymbol{\lambda}\cdot\boldsymbol{u}$ for $\boldsymbol{u} = \boldsymbol{x} - \boldsymbol{x'}$ 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 $\{\boldsymbol{u} \mid \boldsymbol{\lambda}\cdot\boldsymbol{u} = 0 \;\forall$ quasi ranking function $\rho(\boldsymbol{x}) = \boldsymbol{\lambda}\cdot\boldsymbol{x} + \lambda_0\}$ 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 $\{\boldsymbol{B}_1, \ldots, \boldsymbol{B}_p\} \subseteq \mathbb{Q}^n$ (initially empty), such that we search for solutions $\boldsymbol{\lambda}$ such that $\boldsymbol{\lambda}\cdot\boldsymbol{B}_i = 0$ for all $1 \leq i \leq p$. Every time a new model $\boldsymbol{u}$ is found, it is added to $\mathcal{C}$, and a new optimal solution $(\boldsymbol{\lambda}, \lambda_0, \boldsymbol{\delta})$ is computed, we check whether $\delta_{\boldsymbol{u}}$ (ie. δ_i for i the index of $\boldsymbol{u}$ in $\mathcal{C}$) is 0 or 1. If it is 0, which means that *all* quasi ranking functions will satisfy $\boldsymbol{\lambda}\cdot\boldsymbol{u} = 0$, thus $\boldsymbol{u}$ is added to $\mathcal{B}$. We then add to the SMT-query a subformula $\mathit{AvoidSpace}(\boldsymbol{u}, \mathcal{B})$ that forces $\boldsymbol{u}$ not to be a linear combination of elements of vectors of $\mathcal{B}$ ($\boldsymbol{u} \notin \mathrm{Span}(\mathcal{B})$).

This constraint can be implemented by completing $\mathcal{B}$ into a basis $(\mathcal{B}, \mathcal{B}')$ of $\mathbb{Q}^n$, then:

$$\mathit{AvoidSpace}(\boldsymbol{u}, \mathcal{B}) \Leftrightarrow \begin{array}{l} \exists(\alpha_i)_{\boldsymbol{v}_i\in\mathcal{B}} \exists(\beta_i)_{\boldsymbol{v}_i\in\mathcal{B}'} \text{ s.t.} \\ \quad \boldsymbol{u} = \sum_{\boldsymbol{v}_i\in\mathcal{B}} \alpha_i \boldsymbol{v}_i + \sum_{\boldsymbol{v}_i\in\mathcal{B}'} \beta_i \boldsymbol{v}_i \\ \quad \wedge \bigvee_i \beta_i \neq 0 \end{array}$$

4.2 SMT Formulas with Unbounded Domain

In light of Example 3, the issue could be corrected by adding the constraints $\boldsymbol{\lambda}\cdot\boldsymbol{r} \geq 0$ 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 $\mathcal{C}$ the ray generators $\boldsymbol{r}$ incrementally, when the SMT-solver returns a model of the form $\boldsymbol{u} = \cdots + \beta\boldsymbol{r}$ such that $\boldsymbol{\lambda}\cdot\boldsymbol{u}$ unbounded, we compute $\boldsymbol{r}$ and add it to $\mathcal{C}$.

Proposition 5 is still true, since $\sum_{i=1}^m \gamma_i(\boldsymbol{r}.\boldsymbol{a}_i) \geq 0$ for all $\boldsymbol{r}$ a ray of $\mathcal{P}_H$ is a necessary condition for $\boldsymbol{\lambda}$ to be a quasi ranking function.

4.3 Final Algorithm and Proof

These remarks and solutions finally lead to Algorithm 1.

Algorithm 1 MONODIM

Input: $\mathcal{I}$ and τ
  $\mathcal{C} \leftarrow \emptyset$, $\mathcal{B} \leftarrow \emptyset$
  finished $\leftarrow$ false
  $\boldsymbol{\lambda} \leftarrow \mathbf{0}$, $\lambda_0 \leftarrow 0$
  while $\neg$*finished* $\wedge$ $Sat(\mathcal{I} \wedge \tau \wedge AvoidSpace(\boldsymbol{u}, \mathcal{B}))$ with minimization for $\boldsymbol{\lambda} \cdot \boldsymbol{u} (\leq 0)$ **do**
    $(\boldsymbol{u}, unbound) \leftarrow$ a model for $\boldsymbol{u}$ in the above SMT test
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{\boldsymbol{u}\}$
    if *unbound* **then**
      Let $\boldsymbol{r}$ a ray generator of $\mathcal{P}_H$ such that $\boldsymbol{u} = \cdots + \alpha \boldsymbol{r}$
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{\boldsymbol{r}\}$
    $(\boldsymbol{\gamma}, \boldsymbol{\delta}) \leftarrow LP(\mathcal{C}, Constraints(\mathcal{I}))$
    if $\boldsymbol{\gamma} = \mathbf{0}$ **then**
      finished $\leftarrow$ true
    else
      $\boldsymbol{\lambda} \leftarrow \sum_{i=1}^{m} \gamma_i \boldsymbol{a}_i$, $\lambda_0 \leftarrow \sum_{i=1}^{m} \gamma_i b_i$
      if $\delta_{\boldsymbol{u}} = 0$ **then**
        $\mathcal{B} \leftarrow \mathcal{B} \cup \{\boldsymbol{u}\}$
  return $(\boldsymbol{\lambda}, \lambda_0, (\bigwedge_i \delta_i = 1) \wedge \neg Sat(\mathcal{I} \wedge \tau \wedge \boldsymbol{u} = \mathbf{0}))$

Adding "$(\mathcal{I} \wedge \tau \wedge \boldsymbol{u} = \mathbf{0})$ unsatisfiable" is necessary in the case where there is a vector $\boldsymbol{x}$ such that $(\boldsymbol{x}, \boldsymbol{x}') \in \tau$. Indeed, the constraint $AvoidSpace(\boldsymbol{u}, \mathcal{B})$ 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 $(\boldsymbol{\lambda}, \lambda_0, \text{true})$ instead of $(\boldsymbol{\lambda}, \lambda_0, \text{false})$.

Proposition 6. *Algorithm 1 always terminates and returns a quasi ranking function of maximal termination power (Definition 10)*

Lemma 1. *Algorithm 1 always terminates.*

Proof. The number of vectors in $\mathcal{B}$ can only increase or stay the same, and it is bounded by n, thus $\mathcal{B}$ becomes stationary. Once it is stationary, so is the SMT formula (save for the optimization direction $\boldsymbol{\lambda} \cdot \boldsymbol{u} \leq 0$).

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, $\bigwedge_i \boldsymbol{\lambda} \cdot \boldsymbol{c}_i \geq 0$) such that $\delta_{i_0} = 1$ at the preceding iteration, and we are adding a new constraint $\mathcal{C}'$ ($\boldsymbol{\lambda} \cdot \boldsymbol{c}' \geq 0$). As $LP(\mathcal{C} \cup \{\mathcal{C}'\}, Constraints(\mathcal{I}))$ yields $\delta_{i_0} = 0$ for some i_0, this means, by Farkas' lemma Schrijver [1998], that $-\boldsymbol{c}_{i_0}$ can be expressed as a combination of the other constraints in $\mathcal{C}$ and of $\mathcal{C}'$, otherwise said there exist nonnegative γ_i, γ' such that $-\boldsymbol{c}_{i_0} = \sum_{i \neq i_0} \gamma_i \boldsymbol{c}_i + \gamma' \boldsymbol{c}'$. If $\gamma' = 0$, then $\mathcal{C}$ (without $\mathcal{C}'$) already implies $\boldsymbol{\lambda} \cdot \boldsymbol{c}_{i_0} \leq 0$ and thus δ_{i_0} is already null at the preceding iteration; contradiction. Therefore, there exist nonnegative μ_i such that $-\boldsymbol{c}' = \sum_i \mu_i \boldsymbol{c}_i$, otherwise said any solution of $\mathcal{C}$ satisfies $\boldsymbol{\lambda} \cdot \boldsymbol{c}' \leq 0$; 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 $\boldsymbol{u}$ (or $\boldsymbol{r}$) in $\mathcal{C}$ either belong to the vector space spanned by $\mathcal{B}$ (case $\delta_u = 0$), or satisfy $\delta_u = 1$. All $\boldsymbol{\lambda}$ obtained thus verify $\boldsymbol{\lambda} \cdot \boldsymbol{u} > 0$ for all $\boldsymbol{u}$ in $\mathcal{C}$ but not in the span of $\mathcal{B}$. A $\boldsymbol{u}$ or $\boldsymbol{r}$ that was already given as solution by the SMT solver cannot be given again.

Since we have assumed that the solutions of the form $\sum_i \alpha_i \boldsymbol{v}_i$ returned by the SMT solver are taken from the finite set of generators of $\mathcal{P}_H$, then the algorithm terminates in at most as many iterations as the number of generators. □

The algorithm returns $(\boldsymbol{\lambda}, \lambda_0, b)$ where $\rho(\boldsymbol{x}) = \boldsymbol{\lambda} \cdot \boldsymbol{x} + \lambda_0$ is a quasi ranking function and b is a Boolean stating whether it is strict. Note that it is possible that $\boldsymbol{\lambda}$ is null, meaning that there is no linear ranking function that makes at least one transition $(\boldsymbol{x}, \boldsymbol{x}')$ decrease strictly. The following lemmas state that $\boldsymbol{\lambda}$ is of maximal termination power; recall that $\pi(\boldsymbol{\lambda})$ is the set of vertices $(\boldsymbol{x}, \boldsymbol{x}')$ of $I \wedge \tau$ such that $\boldsymbol{\lambda} \cdot \boldsymbol{u} > 0$.

Lemma 2. *The algorithm returns $\boldsymbol{\lambda}$ with maximal $\pi(\boldsymbol{\lambda})$.*

Proof. If there is $(\boldsymbol{x}, \boldsymbol{x}')$ such that $\boldsymbol{\lambda} \cdot \boldsymbol{u} = 0$, either $\boldsymbol{u}$ ultimately belongs to $\mathcal{B}$ in which case there is no $\boldsymbol{\lambda}$ such that $\boldsymbol{\lambda} \cdot \boldsymbol{u} > 0$, either it does not in which case $\boldsymbol{\lambda} \cdot \boldsymbol{u} > 0$. □

Lemma 3. *Let $\boldsymbol{\lambda}$ be the quasi ranking function produced by the algorithm and $\boldsymbol{\lambda}'$ be another quasi ranking function. Then, for all $(\boldsymbol{x}, \boldsymbol{x}') \in \mathcal{I} \wedge \tau$, if $\boldsymbol{\lambda}' \cdot \boldsymbol{u} > 0$ then $\boldsymbol{\lambda} \cdot \boldsymbol{u} > 0$.*

Proof. $\boldsymbol{u}$ can be expressed as a barycenter of a subset of vertices of $\mathcal{P}_H$: $\boldsymbol{u} = \sum_{\boldsymbol{v} \in V'} \alpha_v \boldsymbol{v}$ with $\sum_{\boldsymbol{v} \in V'} \alpha_v = 1$ and for all $\boldsymbol{v} \in V'$, $\alpha_v > 0$, where $V' \subseteq V$. Thus $\boldsymbol{\lambda} \cdot \boldsymbol{u} = \sum_{\boldsymbol{v}} \alpha_v \boldsymbol{\lambda} \cdot \boldsymbol{v}$ (similarly for $\boldsymbol{\lambda}'$). Thus if $\boldsymbol{\lambda} \cdot \boldsymbol{u} = 0$, then for all $\boldsymbol{v} \in V'$, $\boldsymbol{\lambda} \cdot \boldsymbol{v} = 0$. Since $\boldsymbol{\lambda}$ has maximal $\pi_{\mathcal{V}}(\boldsymbol{\lambda})$ by the preceding lemma, it follows that $\boldsymbol{\lambda}' \cdot \boldsymbol{v} = 0$ for any $\boldsymbol{v} \in V'$. But then $\boldsymbol{\lambda} \cdot \boldsymbol{u} = 0$. We thus have proved the contrapose of the result. □

5 Multidimensional Algorithm

Let us note $\prec$ (resp. $\preceq$) the strict lexicographic ordering (resp. quasi lexicographic ordering) over vectors of integers. Our approach is similar to the one of Alias et al. [2010]: for each dimension (d), generate a quasi ranking function of maximal termination power $\rho_d(\boldsymbol{x}) = \boldsymbol{\lambda}_d \cdot \boldsymbol{x} + \lambda_{0,d}$, after restricting the invariant on successive states to transitions that leave the previous components constant. The algorithm is described in Algorithm 2.

Algorithm 2 MULTIDIM

Input: $\mathcal{I}$ and τ
  $d \leftarrow 1$, *failed* $\leftarrow$ false
  repeat
    $(\boldsymbol{\lambda}, \lambda_0, \mathit{strict}) \leftarrow$ MONODIM $\left(\mathcal{I}, \tau \wedge \bigwedge_{d' < d} \boldsymbol{\lambda}_{d'} \cdot \boldsymbol{u} = 0 \right)$
    if $\neg$*strict* **then**
      if $\boldsymbol{\lambda}$ is in the span of ρ **then**
        failed $\leftarrow$ true
      else
        $\rho_d \leftarrow \boldsymbol{\lambda} + \lambda_0$
        $d \leftarrow d + 1$
  until *strict* $\vee$ *failed*
  return if *failed* then "None" else ρ

Lemma 4. *At every iteration of Algorithm 2, ρ is a linearly independent family.*

Proof. By induction over the number of iterations (the size of ρ). If $\boldsymbol{\lambda}$ obtained from MONODIM is a linear combination of ρ, then $\boldsymbol{\lambda} \cdot \boldsymbol{u} = 0$ for all transitions specified in the call to MONODIM; but then MONODIM should have returned *failed*. □

Corollary 1. *Algorithm 2 always terminates in at most n iterations.*

Proof. ρ is a linearly independent family in $\mathbb{Q}^n$, its size growing at each iteration. □

The multidimensional ranking function produced by Algorithm 2 is at least as powerful for proving termination as any other:

Lemma 5. *Let ρ be the quasi multidimensional ranking function produced by Algorithm 2* at any iteration *m and ρ' be another quasi multidimensional ranking function of dimension at most m. 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. By induction over m. The base case $m = 0$ is obvious. Now for $m > 0$. Let $\rho_{<m}$ and $\rho'_{<m}$ be the respective projections of ρ and ρ' to their first $m - 1$ coordinates. Let $(\boldsymbol{x}, \boldsymbol{x}') \in I \wedge \tau$ such that $\rho'(\boldsymbol{x}) \succ \rho'(\boldsymbol{x}')$. If $\rho'_{<m}(\boldsymbol{x}) \succ \rho'_{<m}(\boldsymbol{x}')$, then the induction hypothesis concludes.

Now assume $\rho'_{<m}(\boldsymbol{x}) = \rho'_{<m}(\boldsymbol{x}')$. Let ρ_m and ρ'_m be the respective projections of ρ and ρ' on their m-th coordinate; then $\rho'_m(\boldsymbol{x}) > \rho'_m(\boldsymbol{x}')$. Recall that ρ'_m and ρ_m are quasi monodimensional ranking functions over $\{\boldsymbol{x} \mid \rho'_{<m}(\boldsymbol{x}) = \rho'_{<m}(\boldsymbol{x}')\}$, and ρ_m is the one returned by MONODIM. By Lemma 3, $\rho_m(\boldsymbol{x}) > \rho_m(\boldsymbol{x}')$, which concludes. □

Theorem 1. *Algorithm 2 returns a multidimensional strict ranking function if and only if one exists relative to the invariant given; furthermore, the function returned is of minimum dimension.*

Proof. (If) Let ρ' be a multidimensional strict ranking function and ρ is the quasi multidimensional ranking function returned by Algorithm 2 (or, if *None*, the last value of ρ). Let $(\boldsymbol{x}, \boldsymbol{x'}) \in \tau \wedge \mathcal{I}$; $\rho'(\boldsymbol{x}) \succ \rho'(\boldsymbol{x'})$ because ρ' is a strict ranking function; then by Lemma 5, $\rho(\boldsymbol{x}) \succ \rho(\boldsymbol{x'})$; thus ρ is also a strict ranking function.

(Only if) From the termination condition, if Algorithm 2 does not return *None*, then it returns a strict ranking function.

(Minimality) Let ρ' be a multidimensional strict ranking function and ρ the one returned by Algorithm 2. Assume also that the dimension m of ρ' is less than that of ρ. By Lemma 5 applied to the m-th iteration, then $\rho_{<m}$ is a strict ranking function; but then the algorithm should have stopped at that iteration. □

6 Multiple Control Points

Let us now consider a program with n variables, a set of control points $\mathcal{W}$, an invariant $\mathcal{I}$ and a transition invariant Φ (the general case). We do not distinguish between a control point k and its integer index in $\mathcal{W}$.

Once a cut-set W of program control points has been identified, the program can be rewritten with only W as control points, with "large block transitions" between the control points in W. To each control point k in W, one associates a lexicographic linear function f_k. We want that, for any path $k_1 \to k_2$ with initial values $\boldsymbol{x}$ and final values $\boldsymbol{x'}$, , $f_{k1}(x) < f_{k2}(x_2)$. Then, searching for such rankings amounts to searching for a unique vector of size $|w| \times n$, which is $f = (f_{k1}, f_{k2}, \ldots)$. All notations are adapted adequately.

Considering k in $\mathcal{W}$, we remind the following notations:

1. We note the ranking function in k: $\rho(k, \boldsymbol{x}) = \boldsymbol{\lambda}^k \cdot \boldsymbol{x} + \lambda_0^k$.
2. We note the invariant in k: $\mathcal{I}_k = \{\boldsymbol{x}, \bigwedge_{i=1}^{m_k} \boldsymbol{a_i^k}.\boldsymbol{x} \geq b^k\}$.

Definition 12. *$e^k(\boldsymbol{x})$ is a vector of size $|\mathcal{W}| \times n$ null everywhere except in the coordinates $k \times n$ to $(k+1) \times n - 1$, where it is $\boldsymbol{x}$.*

Definition 13. *We note $\boldsymbol{\lambda}$ the vector of size $|\mathcal{W}| \times n$ formed by the concatenation of the $\boldsymbol{\lambda}^k$.*

$$e^k(\boldsymbol{x}) = \begin{pmatrix} \mathbf{0} & \cdots & \mathbf{0} & \boldsymbol{x} & \mathbf{0} & \cdots & \mathbf{0} \end{pmatrix}^\top$$
$$\boldsymbol{\lambda} = \begin{pmatrix} \boldsymbol{\lambda}^1 & \cdots & \boldsymbol{\lambda}^{k-1} & \boldsymbol{\lambda}^k & \boldsymbol{\lambda}^{k+1} & \cdots & \boldsymbol{\lambda}^{|\mathcal{W}|} \end{pmatrix}^\top$$

We have $\boldsymbol{\lambda} \cdot e^k(\boldsymbol{x}) = \boldsymbol{\lambda}^k \cdot \boldsymbol{x}$.

Definition 14. *We note $Constraints(\mathcal{I}) = \cup_k \{e^k(\boldsymbol{a}_i^k) | \boldsymbol{a}_i^k \in Constraints(\mathcal{I}_k)\}$ the set of vectors encoding the program invariants.*

We will now modify the algorithm presented subsection 3.3 to work on multiple control points. The main modification is in the encoding of the SMT problem:

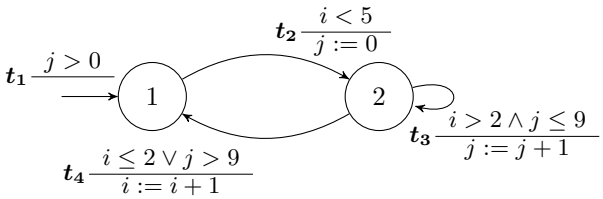
1. Two new additional variables, k and k', denotes the starting and ending control points.
2. We now have $\boldsymbol{u} = e^k(\boldsymbol{x}) - e^{k'}(\boldsymbol{x'})$.
3. We still minimize $\boldsymbol{\lambda} \cdot \boldsymbol{u} = \rho(k, \boldsymbol{x}) - \rho(k', \boldsymbol{x'})$.

Algorithm 1 is modified into Algorithm 3. The multidimensional Algorithm 2 is left unchanged, except that it calls Algorithm 3 instead of Algorithm 1.

Example 4 (Multi control points)**.** *The following example comes from a program with two nested loops Alias et al. [2010]. We have $\mathcal{W} = \{1, 2\}$ and 2 variables i, j.*

Algorithm 3 MONODIM-MULTI for multiple control points

Input: $(\mathcal{I}_k)_{k\in\mathcal{W}}$ and τ
  $\mathcal{C} \leftarrow \emptyset$, $\mathcal{B} \leftarrow \emptyset$
  $finished \leftarrow false$
  $\boldsymbol{\lambda}^k \leftarrow \mathbf{0}$, $\lambda_0^k \leftarrow 0$ for all $k \in \mathcal{W}$
  while $\neg finished \wedge Sat(\mathcal{I} \wedge \tau \wedge AvoidSpace(\boldsymbol{u}, \mathcal{B}) \wedge \boldsymbol{u} = e^k(\boldsymbol{x}) - e^{k'}(\boldsymbol{x}'))$ with minimization for $\boldsymbol{\lambda} \cdot \boldsymbol{u} (\leq 0)$ **do**
    $(\boldsymbol{u}, unbound) \leftarrow$ a model for $\boldsymbol{u}$ in the above SMT test
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{\boldsymbol{u}\}$
    if *unbound* **then**
      Let $\boldsymbol{r}$ a ray generator of $\mathcal{P}_H$ such that $\boldsymbol{u} = \cdots + \alpha \boldsymbol{r}$
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{\boldsymbol{r}\}$
    $(\boldsymbol{\gamma}, \boldsymbol{\delta}) \leftarrow LP(\mathcal{C}, Constraints(\mathcal{I}))$
    if $\boldsymbol{\gamma} = \mathbf{0}$ **then**
      $finished \leftarrow true$
    else
      $\boldsymbol{\lambda}^k \leftarrow \sum_i \gamma_i^k \boldsymbol{a}_i^k$, $\lambda_0^k \leftarrow \sum_i \gamma_i^k b_i^k$ for all k
      if $\delta_{\boldsymbol{u}} = 0$ **then**
        $\mathcal{B} \leftarrow \mathcal{B} \cup \{\boldsymbol{u}\}$
  return $(\boldsymbol{\lambda}^1, \ldots, \boldsymbol{\lambda}^{|\mathcal{W}|}, \lambda_0^1, \ldots, \lambda_0^{|\mathcal{W}|}, (\bigwedge_i \delta_i = 1) \wedge \neg Sat(\mathcal{I} \wedge \tau \wedge \boldsymbol{u} = \mathbf{0}))$



We have the following invariant:

$$\mathcal{I} : (k = 1 \implies i \leq 5) \wedge (k = 2 \implies 0 \leq j \leq 10 \wedge i \leq 4)$$

1. *Beginning with* $\mathcal{C} = \{\}$ *and* $\rho(\boldsymbol{x}) = 0$, *that is:* $\boldsymbol{\lambda}^1 = \mathbf{0}$ *and* $\boldsymbol{\lambda}^2 = \mathbf{0}$. *We have* $\boldsymbol{x} = \binom{i}{j}$ *and* $\boldsymbol{u} = e^k(\boldsymbol{x}) - e^{k'}(\boldsymbol{x}')$. *In the SMT-query,* τ *is now written as follows:*

$$\left(k = 1 \wedge k' = 2 \implies i < 5 \wedge j' = 0 \wedge \boldsymbol{u} = (i, j, i', j')^\top\right) \wedge \ldots$$

First iteration.

2. $Sat(\mathcal{I} \wedge \tau \wedge AvoidSpace(\boldsymbol{u}, \mathcal{B}) \wedge \mathbf{0} \cdot \boldsymbol{u} \leq 0)$ *?*
Yes, with $k = 2$, $k' = 1$, $\boldsymbol{x} = \binom{1}{10}$ *and* $\boldsymbol{x}' = \binom{-2}{10}$ *(this corresponds to transition* $\mathbf{t_4}$*)*

3. $\mathcal{C} \leftarrow \left\{\begin{pmatrix}1 & 10 & -2 & -10\end{pmatrix}^\top\right\}$

4. *Call* $LP(\mathcal{C}, Constraints(\mathcal{I}))$.
It gives us $\boldsymbol{\lambda}^1 = \binom{0}{0}$ *and* $\boldsymbol{\lambda}^2 = \binom{1/2}{0}$.

Second iteration.

2'. $Sat(\mathcal{I} \wedge \tau \wedge AvoidSpace(\boldsymbol{u}, \mathcal{B}) \wedge \boldsymbol{\lambda} \cdot \boldsymbol{u} \leq 0)$ *?*
Yes, with $k = 2$, $k' = 2$, $\boldsymbol{x} = \binom{3}{0}$ *and* $\boldsymbol{x}' = \binom{3}{1}$.

3'. $\mathcal{C} \leftarrow \mathcal{C} \cup \left\{\begin{pmatrix}0 & 0 & 0 & -1\end{pmatrix}^\top\right\}$

4'. *Call* $LP(\mathcal{C}, Constraints(\mathcal{I}))$.
It gives us $\boldsymbol{\lambda}^1 = \binom{0}{0}$ *and* $\boldsymbol{\lambda}^2 = \binom{-11/2}{-1}$.

Third iteration.

2". $Sat(\mathcal{I} \wedge \tau \wedge y - y' \leq 0)$ *? No, we stop.*

Return. *We obtain* $\rho^1(\boldsymbol{x}) = 0, \rho^2(i, j) = -11/2i - j + 32$, *a strict ranking function for* $(\tau, \mathcal{I})$.

7 Complexity

Ben-Amram and Genaim [2014, Sec. 3], which consider integer programs with the combined transition

relation and invariant Φ specified in disjunctive normal form, provide an exponential-time algorithm and prove that the problem of deciding the existence of a linear ranking function is coNP-complete. We now shall show that this decision problem is still coNP-complete if the Φ specified as input is given in general form (let us say, with a linear integer arithmetic formula with prefix existential quantifiers and disjunctions allowed) as opposed to disjunctive normal form.

Proposition 7. *Deciding the existence of a linear ranking function given Φ (with $\exists$-prefix and $\vee$) is coNP-complete.*

Proof. coNP-hardness directly follows from our class of input including that studied by Ben-Amram and Genaim [2014]; let us now show membership in coNP.

Our algorithms (section 4, section 6) stop when either they have a linear ranking function, either they reach an unsolvable system of constraints. If the number of program variables is n and the number of control points is $|P|$, then these constraints are over $u = (n+1) \cdot |P|$ unknowns. By a combination of Farkas' lemma and Carathéodory's theorem [Schrijver, 1998, §7.7], if this system is unsatisfiable, there exists an unsatisfiable subset of $u+1$ of these constraints.

Each of this constraints is given by a generator (vertex or ray) of the integer hull of one of the disjuncts of the disjunctive normal form of Φ. The bit-size of such a generator is polynomial in the size of the constraint representation of this disjunct [Schrijver, 1998, §17.1], and this constraint representation is an extract of Φ. Therefore, a witness of the unsatisfiability of the constraint system (i.e. of the absence of a linear affine ranking function) may be given by $u+1$ elements of polynomial size, and thus is itself polynomial. □

Proposition 8. *Our algorithms have exponential complexity at most.*

Proof. Our algorithms enumerate at most once each generator of the integer hull of each disjunct of the disjunctive normal form of Φ, which (as in the previous proof) have polynomial size; thus there is an exponential number of them. Each of them may be obtained as the maximum solution of integer linear programming problems for all disjuncts, which can be done in exponential time. Finally, at each iteration the constraint system is solved by linear programming. □

Thus, moving to a more succinct representation of transitions than in Ben-Amram and Genaim [2014] conserves coNP-membership and solving in exponential complexity.

8 Extensions

Coupling of Invariant and Transition Relation We supposed in subsection 3.1 that we have an invariant $\mathcal{I}$ expressed as the constraints of a convex closed polyhedron, and a transition related τ expressed as a formula of real linear arithmetic. In subsection 2.2, following Ben-Amram and Genaim [2014] we instead considered a single relation Φ incorporating both invariant and transition. This format can be handled by also using counterexamples for the nonnegativity constraint — i.e. counterexamples are not only extremal nondecreasing quadruples $(k, \boldsymbol{x}, k', \boldsymbol{x}')$ but also extremal points $(k, \boldsymbol{x})$ where the ranking function candidate is negative.

Note a subtle difference between the two approaches, when instantiated on the multidimensional algorithm from section 5: in our original algorithm, as in Alias et al. [2013], we require that each component ρ_j of the multidimensional ranking function should be nonnegative with respect to an invariant $\mathcal{I}$ ($\forall \boldsymbol{x} \in \mathcal{I}\ \rho_j(\boldsymbol{x}) \geq 0$) while in the modified version, we restrict at each step the invariant to the states over which the preceding components of the multidimensional ranking function do not decrease. This is a more powerful approach: there are programs that cannot be proved to terminate by using the same invariant for nonnegativity in all components of a lexicographic linear ranking function, but can be if the invariant is refined across components [Ben-Amram and Genaim, 2014, Ex. 2.12].

Richer Classes of Formulas For the transition relation τ and for the compound transition relation+invariant Φ, we have so far assumed linear integer arithmetic. In fact, any decidable theory for which it is possible to perform optimization with respect to the integer components will do.

It is for instance possible to handle uninterpreted functions and arrays, since, by Ackermann's expansion, to a quantifier-free formula F with uninterpreted functions one can associate a formula F' without uninterpreted functions, such that F' has the same models with respect to the free variables of

F [Kroening and Strichman, 2008, ch. 4]. Universally quantified properties (e.g. $\forall 0 \leq k < n\ t[k] = 4$) can be over-approximated by quantifier instantiation.

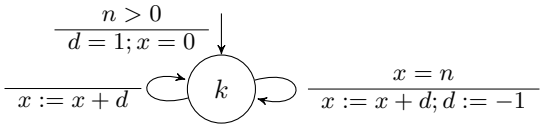
Rational Variables The monodimensional algorithms are suitable for synthesizing what Ben-Amram and Genaim [2014] call *weak ranking functions* on programs using rational variables: in Definition 6, replace $\mathbb{Z}$ by $\mathbb{Q}$.

Proposition 9. *If ρ is a monodimensional weak ranking function, there exists $\alpha > 0$ such that $\alpha\rho$ decreases by at least 1 at each step. Thus, the program terminates.*

Proof. Let $\beta = \inf\{\rho_k \cdot \boldsymbol{x} - \rho_{k'} \cdot \boldsymbol{x}' \mid (k, \boldsymbol{x}, k', \boldsymbol{x}') \in \Phi\}$. Since Φ is a finite union of closed convex polyhedra, this infimum is reached within the set. $\beta \leq 0$ would contradict the weak ranking function hypothesis. Then $\beta > 0$ and one can take $\alpha = 1/\beta$. □

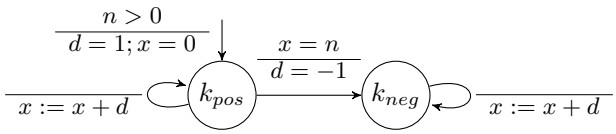
This simple scaling approach does not always work for multidimensional ranking functions, and a somewhat more complicated procedure must be applied to obtain a multidimensional ranking function (strict decreasing steps of at least 1 unit) from a multidimensional weak ranking function [Ben-Amram and Genaim, 2014, §5.3]. We did not investigate the combination of that procedure with our counterexample-based approach, since all multidimensional problems that we tried (except for concocted examples) were solved by scaling.

Disjunctive Invariants In some cases, it is interesting to have multiple ranking functions for the same control point, depending on some kind of partitioning. For instance, in



it is necessary to distinguish two phases of the loop ($d = 1$, $d = -1$) with two different ranking functions ($n - x$ and x).

Certain static analyzers, including PAGAI in certain modes of operation Henry et al. [2012b,a,b], return disjunctive invariants: to a given program point (possibly in a cut-set of program points) they associate a disjunction of abstract elements; for instance, here, a possible invariant is $(d = 1 \wedge 0 \leq x \leq n) \vee (d = -1 \wedge 0 \leq x \leq n)$ It is possible to split the control point according to $d = 1 \vee d = -1$:



Then by taking $\{k_{pos}; k_{neg}\}$ as cut-set, one can construct the ranking function.

9 Implementation and Experimental Results

Implementation We implemented our prototype TERMITE in 3k lines of OCAML [7]. Termite uses LLVM[8] to compile C code into a Single Static Assignment intermediate representation Cytron et al. [1991]. PAGAI Henry et al. [2012b, 2014] is used to compute invariants from the LLVM IR. The transition relation is produced from these invariants and the LLVM IR. Then, the core analyzer implements the multidimensional, multiple control point algorithm described herein. Z3[9] answers (optimizing) SMT and LP queries.

Experiments We have compared TERMITE with LOOPUS[10] Zuleger et al. [2011], APROVE[11] Giesl et al. [2014], Ultimate Büchi Automizer[12], RANK Alias et al. [2013] (with the C2FSM front-end and AS-

[7] http://termite-analyser.github.io/
[8] http://llvm.org/
[9] https://github.com/Z3Prover
[10] http://forsyte.at/software/loopus/
[11] http://aprove.informatik.rwth-aachen.de/
[12] http://ultimate.informatik.uni-freiburg.de/BuchiAutomizer/

Table 1: Comparison. For each benchmark suite, the total number of benchmarks and the number of benchmarks proved to terminate by each tool are given. Timings, in milliseconds, exclude the front-end for TERMITE and LOOPUS and the invariant generator for TERMITE. (l, c) are the average number of lines and columns of the linear programming instances.

Suite	# benchmarks	Termite			Loopus		AProVE		Ultimate	
		# success	time	(l, c)	# success	time	# success	time	# success	time
PolyBench	30	22	117	$(9, 3)$	30	37	0	2100	0	2750
Sorts	6	5	126	$(15, 4)$	3	67	0	12230	0	2980
TermComp	129	119	12	$(2, 1)$	78	15	111	9757	85	5863
WTC	58	46	64	$(5, 2)$	33	48	36	11740	40	4536

PIC[13] Gonnord and Schrammel [2014] accelerating invariant generator), on examples from POLYBENCH[14], WTC (V2)[15], the termination competition[16] and some sorting algorithms (Table 9). TERMITE solves 33% more examples than LOOPUS, in twice the time. APROVE and ULTIMATE are considerably slower.

Due to limitations in its front-end and invariant generator, RANK can only run on 46 files from the WTC test suite, solving 25 of them in total time 76 ms, with average linear programming problem $(\mathit{lines}, \mathit{columns}) = (584, 229)$, much higher than TERMITE's $(5, 2)$. On some examples however, RANK is able to prove termination that TERMITE is unable to prove. This essentially comes from the combination of its front-end C2FSM and its invariant generator ASPIC that sometimes perform clever graph transformations on the control-flow graph and thus are able to synthesize better invariants than PAGAI.

Difficulties A termination analyser typically consists in 1) a front-end 2) an invariant generator 3) a termination analysis. One therefore compares whole toolchains instead of only the termination analyses. One difficulty is that some front-ends are more picky than others; for instance, APROVE's front-end crashes on some LLVM opcodes. We have made some reasonable efforts to have our examples accepted by the various tools when we could identify some obvious reason why they could not be processed by the front-end or invariant generator. We had to leave out other tools from our experiments due to difficulties in running them successfully and lack of documentation. It is also possible that we used some of the listed tools improperly; if so we apologize to their respective authors.

Limitations Contrary to IRANKFINDER [17], TERMITE is unable to prove termination when the desired (multidimensional) ranking function has non positive components: we did not implement the CEGAR loop for nonnegativity suggested in section 8.

Contrary to T2[18] TERMITE is also unable to prove termination when the behavior of a loop is divided into *phases* (as we already discussed in section 6), unless some preprocessor divided the phases into different control points.

We unfortunately were unable to compare these two tools with ours, due to different input formats and unavailability of converters.

10 Conclusion and Future Work

We have proposed a new algorithm for the derivation of lexicographic linear ranking functions. There has been ample literature on this on related topics (see [Alias et al., 2010, §6] and Cook et al. [2011] for a large panel of proposed methods) ; let us therefore summarize salient points of difference.

Termination with Guaranteed Result Several existing approaches do not necessarily terminate. In contrast, our approach always terminates, and always outputs a lexicographic linear ranking function if one exists. Also, it is guaranteed to be of minimal dimension.

[13] http://laure.gonnord.org/pro/aspic/aspic.html
[14] http://www.cs.ucla.edu/~pouchet/software/polybench/
[15] http://compsys-tools.ens-lyon.fr/wtc/index.html
[16] http://termination-portal.org/wiki/Termination_Competition
[17] http://www.loopkiller.com/irankfinder/linrf.php
[18] http://research.microsoft.com/en-us/projects/t2/

Use of Concrete Runs instead of Farkas' Lemma Like previous algorithms Alias et al. [2010], we solve a sequence of problems of finding quasi monodimensional linear ranking functions; and each quasi monodimensional linear ranking function is obtained as the solution of a linear program. Where we differ, however, is how we build these linear programs; our method produces much smaller ones (e.g. by 1–2 orders of magnitude compared to RANK Alias et al. [2013], and thus is more *scalable*.

Most other approaches Bradley et al. [2005a,b], Alias et al. [2013], Podelski and Rybalchenko [2004], Larraz et al. [2013] create new unknowns (coefficients from Farkas' lemma) for each face of each transition polyhedron corresponding to a path between control nodes considered for the ranking function, along with relevant constraints. In contrast, with ours, constraints correspond to concrete $(\boldsymbol{x}, \boldsymbol{x}')$ transitions and the number of unknowns does not depend on the complexity of the transitions. Also, instead of creating constraints upfront, we create them on demand.

Furthermore, we can deal with integer variables both easily and precisely (by specifying them as integers in the SMT-solving call) while other approaches Larraz et al. [2013] need to apply refinements based on Gomory-Chvátal cutting planes. We can similarly deal with existential quantification, uninterpreted functions, arrays, Booleans etc. in the transition relation.

Use of a Cut-set Our approach is able to solve for ranking functions only at a "cut set" of control nodes. In particular, we do not require that there is a lexicographic linear ranking function at points outside of the cut-set. This is useful for instance if a ranking function would need to be defined by case analysis at such points (e.g. according to the value of a Boolean variable, as at point B in the following example); it is difficult to synthesize such ranking functions, but in our case we do not need to do so, because we synthesize the ranking function only a cut-set points (e.g. A).[19]

Listing 1: A linear r.f. decreases on each path, not each step

```
int x = positive();
while(x >= 0) {      /* A */
  bool c = choose();
  if (c) x=x-1;      /* B */
  if (!c) x=x-1;
}
```

In previous approaches Gulwani and Zuleger [2010], one had to expand the control flow between these nodes, yielding (in general) an exponential number of transitions. In contrast, our algorithm "sees" transitions as needed, performing counterexample-guided refinement of the constraint system, which improves scalability.

Comparison with Other Approaches The difference with RANK and IRANKFINDER is not with respect to expressive power (we show that one variant of our algorithm is equivalent to RANK and another to IRANKFINDER) but with respect to complexity and efficiency. Rank and iRankFinder need transition relations in disjunctive normal form: in order to deal, as we do, with "big block" transitions between a cut-set of control locations, they would need to expand them at exponential cost — though one can argue that in many programs this unfolding is not so large.

Our system of constraints is constructed lazily, theirs are constructed eagerly. Heizmann, Hoenicke & Podelski's approach (Ultimate Büchi Automizer) is to cover the set of program traces by ω-regular languages of terminating traces ("modules"). The termination argument for each module can be obtained from a variety of approaches; thus their approach truly is a meta-approach since a variety of sub-provers can be used. We in fact think that our technique could be adapted into a sub-prover for their system.

Cook, See & Zuleger Cook et al. [2013] give contrived examples where lexicographic linear arguments cannot prove termination, but linear Ramsey-based arguments can. They however conclude that in almost all cases where linear Ramsey-based arguments work, lexicographic linear ranking functions are sufficient and that the question is therefore how to find them efficiently. This is the question we address in this article.

[19] Our approach can also be applied to synthesize the same linear ranking function at all nodes in the cut-set. This linear ranking function is therefore allowed to locally increase at nodes outside of the cut-set, as long as this increase is compensated before the next node in the cut-set is reached, as in some other approaches Zankl and Middeldorp [2009].

The algorithm in Loopus Zuleger et al. [2011] is similar to ours in some ways; however, it explores all paths in a given loop, and uses heuristics to syntactically derive ranking functions from the transitions of the loop. In our experiments, Loopus is able to quicky conclude for termination in many cases since simple arguments are very often sufficient to prove that a given loop terminates Rodrigues et al. [2014].

Future Work A natural extension of our counterexample-guided refinement approach would be to use it to generate the supporting invariant: instead of generating constraints for each $(\boldsymbol{x}, \boldsymbol{x}')$ transition encountered where $\boldsymbol{x}$ lies within a precomputed invariant $\mathcal{I}$, one would first try to prove $\boldsymbol{x}$ to be unreachable, using any approach capable of providing an inductive invariant $\mathcal{J}$ such that $\boldsymbol{x} \notin \mathcal{J}$; if successful, one would replace $\mathcal{I}$ by $\mathcal{I} \cap \mathcal{J}$ and restart.

A further refinement of the method would be to look for a large set of initial states that guarantee that $\boldsymbol{x}$ is unreachable; the intersection of such sets would be a sufficient condition for termination to be proved by the ranking function output by the algorithm (*conditional termination*).

Our algorithm for finding linear ranking functions is a particular kind of partial *quantifier elimination*: conditions 2 and 3 in Definition 6, or condition 1 in Proposition 2, are formulas of the form $\forall x\ F$ where F is quantifier free: we wish to obtain a satisfying assignment of their free variables. A full quantifier elimination would compute exactly the cone of solutions, but we stop once we have a point in the relative interior. It remains to be seen how our approach of extremal instantiation would generalize to a more general class of quantified formulas.

Acknowledgments

We would like to show our gratitude to Paul Feautrier for sharing his comments and ideas during the course of this research, and Marc Vincenti who helped us a lot with the benchmarks. We also thank the anonymous reviewers for their insightful comments and feedback, which improved the quality of this paper.

References

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.

C. Alias, A. Darte, P. Feautrier, and L. Gonnord. Rank: a tool to check program termination and computational complexity. In *Constraints in Software Testing Verification and Analysis*, Luxembourg, Mar. 2013. doi: 10.1109/ICSTW.2013.75. URL http://hal.inria.fr/hal-00801571.

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: 10.1016/j.entcs.2010.09.002.

S. Balev, P. Quinton, S. Rajopadhye, and T. Risset. Linear programming models for scheduling systems of affine recurrence equations - a comparative study. In *ACM Symposium on Parallel algorithms and architectures*, pages 250–258. ACM, 1998. doi: 10.1145/277651.277691.

A. M. Ben-Amram and S. Genaim. Ranking functions for linear-constraint loops. *J. ACM*, 61(4): 26:1–26:55, July 2014. ISSN 0004-5411. doi: 10.1145/2629488.

A. R. Bradley, Z. Manna, and H. B. Sipma. The polyranking principle. In *Intl. Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *LNCS*, pages 1349–1361. Springer, July 2005a. doi: 10.1007/11523468_109.

A. R. Bradley, Z. Manna, and H. B. Sipma. Linear ranking with reachability. In K. Etessami and S. K. Rajamani, editors, *Computer aided verification (CAV)*, volume 3576 of *LNCS*, pages 491–504. Springer, July 2005b. doi: 10.1007/11513988_48.

M. Codish and S. Genaim. Proving termination one loop at a time. In F. Mesnard and A. Serebrenik, editors, *13th International Workshop on Logic Programming Environments, Tata Institute of Fundamental Research, Mumbai, India, December 8, 2003*, Technical Report CW371, pages 48–59. Katholieke Universiteit Leuven, 2003. URL http://www.cs.kuleuven.ac.be/publicaties/rapporten/cw/CW371.pdf.

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.

B. Cook, A. See, and F. Zuleger. Ramsey vs. lexicographic termination proving. In *TACAS*, volume 7795 of *LNCS*, pages 47–61. Springer, 2013. doi: 10.1007/978-3-642-36742-7_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. doi: 10.1145/512760.512770.

R. Cytron, J. Ferrante, B. Rosen, M. Wegman, and K. Zadeck. Efficiently computing static single assignment form and the control dependence graph. *TOPLAS*, 13(4):451–490, 1991. doi: 10.1145/115372.115320.

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. 1992a.

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. 1992b.

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.

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

J. Giesl, M. Brockschmidt, F. Emmes, F. Frohn, C. Fuhs, C. Otto, M. Plücker, P. Schneider-Kamp, T. Ströder, S. Swiderski, and R. Thiemann. Proving termination of programs automatically with aprove. In S. Demri, D. Kapur, and C. Weidenbach, editors, *Automated Reasoning (IJCAR)*, volume 8562 of *LNCS*, pages 184–191. Springer, 2014. doi: 10.1007/978-3-319-08587-6_13.

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.

L. Gonnord and P. Schrammel. Abstract acceleration in linear relation analysis. *Sci. Comput. Program.*, 93:125–153, 2014. doi: 10.1016/j.scico.2013.09.016. URL http://dx.doi.org/10.1016/j.scico.2013.09.016.

S. Gulwani and F. Zuleger. The reachability-bound problem. In *ACM symposium on programming language design and implementation (PLDI)*, pages 292–304. ACM, 2010. doi: 10.1145/1806596.1806630.

J. Henry, D. Monniaux, and M. Moy. Succinct representations for abstract interpretation - combined analysis algorithms and experimental evaluation. In *Static Analysis - 19th International Symposium, SAS 2012, Deauville, France, September 11-13, 2012. Proceedings*, pages 283–299, 2012a. doi: 10.1007/978-3-642-33125-1_20.

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

J. Henry, D. Monniaux, and M. Moy. The Pagai static analyser, 2014. URL http://pagai.forge.imag.fr/.

D. Kroening and O. Strichman. *Decision procedures*. Springer, 2008.

D. Larraz, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. Proving termination of imperative programs using max-SMT. In *FMCAD*, 2013.

D. Monniaux and L. Gonnord. Using bounded model checking to focus fixpoint iterations. In *18th International Static Analysis Symposium (SAS'11)*, Venice, Italy, Sept. 2011. doi: 10.1007/978-3-642-23702-7_27.

R. Nieuwenhuis and A. Oliveras. On SAT modulo theories and optimization problems. In *SAT*, volume 4121 of *LNCS*, pages 156–169. Springer, 2006. doi: 10.1007/11814948_18.

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. doi: 10.1007/978-3-540-24622-0_20.

R. E. Rodrigues, P. Alves, F. Pereira, and L. Gonnord. Real-world loops are easy to predict : a case study. In *Workshop on Software Termination*, Vienne, Austria, July 2014. URL https://hal.inria.fr/hal-01006208.

A. Schrijver. *Theory of linear and integer programming*. Wiley, 1998.

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. doi: 10.1007/978-3-642-31365-3_38.

A. Shamir. A linear time algorithm for finding minimum cutsets in reducible graphs. *SIAM J. Comput.*, 8(4):645–655, 1979. doi: 10.1137/0208051.

J. Tristan and X. Leroy. Verified validation of lazy code motion. In M. Hind and A. Diwan, editors, *Programming Language Design and Implementation (PLDI)*, pages 316–326. ACM, 2009. doi: 10.1145/1542476.1542512.

A. M. Turing. Checking a large routine. In *Report of a Conference on High Speed Automatic Calculating Machines*, 1949. URL http://www.turingarchive.org/browse.php/B/8.

C. Urban. The abstract domain of segmented ranking functions. In *Static Analysis (SAS)*, volume 7935 of *LNCS*, pages 43–62. Springer, 2013. doi: 10.1007/978-3-642-38856-9_5.

C. Urban and A. Miné. An abstract domain to infer ordinal-valued ranking functions. In *Programming Languages and Systems (ESOP)*, volume 8410 of *LNCS*, pages 412–431. Springer, 2014.

H. Zankl and A. Middeldorp. Increasing interpretations. *Ann. Math. Artif. Intell.*, 56(1):87–108, 2009. doi: 10.1007/s10472-009-9144-7.

F. Zuleger, S. Gulwani, M. Sinn, and H. Veith. Bound analysis of imperative programs with the size-change abstraction. In *Proceedings of the 18th international conference on Static analysis*, SAS'11, pages 280–297, Berlin, Heidelberg, 2011. Springer-Verlag. URL http://dl.acm.org/citation.cfm?id=2041552.2041574.