HAL
archives-ouvertes.fr

Program Termination and Worst Time Complexity with Multi-Dimensional Affine Ranking Functions

Christophe Alias, Alain Darte, Paul Feautrier, Laure Gonnord, Clément Quinson

▶ **To cite this version:**

Christophe Alias, Alain Darte, Paul Feautrier, Laure Gonnord, Clément Quinson. Program Termination and Worst Time Complexity with Multi-Dimensional Affine Ranking Functions. [Research Report] 2009, pp.31. inria-00434037

HAL Id: inria-00434037
https://hal.inria.fr/inria-00434037

Submitted on 20 Nov 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Program Termination and Worst Time Complexity with Multi-Dimensional Affine Ranking Functions

Christophe Alias — Alain Darte — Paul Feautrier — Laure Gonnord — Clément Quinson

N° 7037

Novembre 2009

---

apport de recherche

ISSN 0249-6399 ISRN INRIA/RR--7037--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
GRENOBLE – RHÔNE-ALPES

Program Termination and Worst Time Complexity with Multi-Dimensional Affine Ranking Functions

Christophe Alias*, Alain Darte †, Paul Feautrier ‡, Laure Gonnord §, Clément Quinson ¶

Thème : Architecture et compilation
Équipe-Projet Compsys

Rapport de recherche n° 7037 — Novembre 2009 — 31 pages

Abstract: A standard method for proving the termination of a flowchart program is to exhibit a ranking function, i.e., a function from the program states to a well-founded set, which strictly decreases at each program step. Our main contribution is to give an efficient algorithm for the automatic generation of multi-dimensional affine nonnegative ranking functions, a restricted class of ranking functions that can be handled with linear programming techniques. Our algorithm is based on the combination of the generation of invariants (a technique from abstract interpretation) and on an adaptation of multi-dimensional affine scheduling (a technique from automatic parallelization). We also prove the completeness of our technique with respect to its input and the class of rankings we consider. Finally, as a byproduct, by computing the cardinal of the range of the ranking function, we obtain an upper bound for the computational complexity of the source program, which does not depend on restrictions on the shape of loops or on program structure. This estimate is a polynomial, which means that we can handle programs with more than linear complexity. The method is tested on a large collection of test cases from the literature. We also point out future improvements to handler larger programs.

Key-words: static analysis, termination proof, multidimensional affine ranking functions, worst-case time complexity estimation

* INRIA Researcher
† CNRS Researcher
‡ Professor at ENS Lyon
§ Research Assistant at Lille University
¶ PhD student at LIP, ENS Lyon

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Terminaison de programmes et complexité au pire avec des fonctions de terminaison multidimensionnelles

Résumé : Une manière standard de prouver la terminaison d'un programme est d'exhiber une fonction de terminaison, c'est-à-dire une fonction qui associe à chacun des états du programme un élément d'un ensemble bien fondé, et qui décroît strictement à chaque étape du programme. La contribution de ce rapport est un algorithme performant pour générer automatiquement des fonctions de terminaison affines multidimensionnelles, une classe restreinte de fonctions qui peuvent être traitées à l'aide de la programmation linéaire. Notre algorithme est basé sur une combinaison de techniques : la génération d'invariants par interprétation abstraite et (une adaptation de la technique de) l'ordonnancement multidimensionnel affine. Nous prouvons la complétude de notre technique vis à vis de ses entrées et de la classe des fonctions de terminaison que nous considérons. Finalement, en calculant le cardinal de la plage de valeurs prises par la fonction de terminaison, nous obtenons, indépendamment de la structure du programme ou du type de boucles, une borne supérieure sur la complexité du programme source. Cette estimation est un polynôme, ce qui signifie que nous pouvons traiter des programmes de complexité sur-linéaire. Cette méthode est appliquée à de nombreux exemples de la littérature. Nous donnons enfin quelques pistes pour des améliorations futures, notamment afin de traiter des programmes de plus grande taille.

Mots-clés : analyse statique, preuve de terminaison, génération de fonctions de terminaison multidimensionnelles affines, estimation de la complexité au pire

1 Introduction and motivation

The problem of proving program correctness has been with us since the early days of Computer Science. In a seminal paper [22], R. W. Floyd proposed what has become one of the standard approaches: affix assertions to each program point and prove that they are consequences of the assertions of its predecessors in the program control graph. The assertions at the entry point of the program are its *preconditions*, the assertions at loop entry points are *invariants*, while the assertions at its exit point must entail correctness, according to some set of requirements. Constructing the required set of assertions is a tedious and error-prone task. In the case of a loop-free program, it is possible to mechanically build all the assertions, either going forward from the preconditions, or backward from the requirements. However, there exist while loop programs whose invariants are recursively enumerable but not recursive [4]. Since the formalisms we dare use for describing sets in compilers – finite and co-finite sets, regular and context-free languages, polyhedra, Presburger arithmetics – all deals with recursive sets, this means that in some cases, the invariants must be approximated, mostly using techniques from abstract interpretation [15].

At the same time, it was soon realized that this method proves only partial correctness, i.e., that the program gives the correct result if and when it terminates. To prove termination, one needs a variant or ranking function (a W-function in Floyd's terminology), i.e., a function from the states of the program to some well-founded set, which strictly decreases at each program step. Here again, it would be nice to have an algorithm for building ranking functions, but this is not possible in all cases since it would give a solution to the undecidable halting problem. However, this does not preclude the existence of partial solutions, which, e.g., handle only programs (or approximated models) of a restricted shape, or look for rankings in a restricted class of functions. Our main goal is to provide such an algorithm, which applies to arbitrary programs but considers only the set of multi-dimensional nonnegative affine functions under lexicographic ordering, as defined formally in Section 2.

The construction of a ranking function can sometimes be linked to the evaluation of the worst case execution time (WCET) of the source program. Obviously, if a program does not terminate, its WCET is infinite. If the program terminates and a ranking function exists, then, under some hypotheses, it can give some information on the WCET. For example, if the ranking function has co-domain $\mathbb{N}$ (one-dimensional function), its value at program start is an upper bound on the number of steps before termination since it decreases at least by one at each program step. (The situation is more complicated in the case of multi-dimensional ranking functions, i.e., if the co-domain is $\mathbb{N}^d$.) In other words, if accurate, the ranking function can give the order of magnitude of the WCET, or *time complexity* [25] of the program but not its exact value. This estimate is static and "high-level" in the sense that it just counts the program steps. Estimating the WCET is much more difficult [41].

The rest of the paper is organized as follows. We first describe *integer interpreted automata*, a standard model for abstracting the control part of arbitrary non-recursive programs. We then define ranking functions and prove a few general results. Section 3 is the main part of the paper; we present a new method for constructing multi-dimensional affine ranking functions and to infer the computational complexity of the source program. We prove the relative completeness of the method, which means that if there exists a ranking function for the set of invariants we have found, then our algorithm finds one. In Section 4, we report on our implementation through a large set of

benchmarks from the literature. We next describe other approaches to the termination problem and conclude, pointing to some unsolved problems and outlining future work.

2 Definitions and elementary properties

2.1 Notations

Hereafter, matrices are written with capital letters (as A) and column vectors with a top arrow (as $\vec{x}$). The components of a vector $\vec{x}$ of dimension d are denoted by $\vec{x}[i]$, $0 \leq i < d$. Thus, the i-th component of $\vec{x}$ is $\vec{x}[i-1]$. Sets are represented with calligraphic letters such as $\mathcal{W}$, $\mathcal{K}$, etc.

2.2 Integer interpreted automata

In the tradition of most previous work on program termination and static program analysis, we do not start from the program itself but from an abstraction: the associated *integer interpreted automaton*. This is similar to the flowcharts which were used a long time ago to express programs (see, e.g., Manna's book [33]) until the advent of structured programming. However, when one looks at real-life programs, many deviations from the strict structured model can occur, including premature loop termination, exceptions, and even the occasional `goto`. Starting from a flowchart allows us to not depend of the details of the syntax and semantics of the source language, which can be dealt with by an appropriate preprocessor.

A program is represented by a *general relational (integer) interpreted automaton* $(\mathcal{K}, n, k_{init}, \mathcal{T})$ defined by:

- a finite set $\mathcal{K}$ of *control points*;
- n integer variables represented by a vector $\vec{x}$ of size n;
- an initial control point $k_{init} \in \mathcal{K}$;
- a finite set $\mathcal{T}$ of 4-tuples (k, g, a, k'), called *transitions*, where $k \in \mathcal{K}$ (resp. $k' \in \mathcal{K}$) is the source (resp. target) control point, $g : \mathbb{Z}^n \mapsto \mathbb{B} = \{true, false\}$ is a guard (function from variable valuations to Booleans), and $a : \mathbb{Z}^n \mapsto \mathcal{P}(\mathbb{Z}^n)$ is an action that assigns to each variable valuation $\vec{x}$ a subset $a(\vec{x})$ of $\mathbb{Z}^n$.

Semantics The set of states is $\mathcal{K} \times \mathbb{Z}^n$. A *trace* from $(k_0, \vec{x}_0)$ to $(k, \vec{x})$ is a sequence $(k_0, \vec{x_0}), (k_1, \vec{x}_1), \ldots, (k_p, \vec{x}_p)$ such that $k_p = k$, $\vec{x}_p = \vec{x}$ and for each i, $0 \leq i < p$, there exists in $\mathcal{T}$ a transition (k_i, g_i, a_i, k_{i+1}) such that $g_i(\vec{x}_i) = true$ and $\vec{x}_{i+1} \in a_i(\vec{x}_i)$. Given an initial valuation $\vec{v}$, a state $(k, \vec{x})$ is *reachable from* $\vec{v}$ iff (if and only if) there exists a trace from $(k_{init}, \vec{v})$ to $(k, \vec{x})$. A state $(k, \vec{x})$ is *reachable* if there exists $\vec{v} \in \mathbb{Z}^n$ such that $(k, \vec{x})$ is reachable from $\vec{v}$. The set of reachable states is denoted by $\mathcal{R}$. Note that there can be two forms of non-determinism in the "execution" of an interpreted automaton. First, if $a(\vec{x})$ is not a singleton, a transition can lead to several distinct variable valuations ($a(\vec{x}) = \mathbb{Z}$ is a way to assign to $\vec{x}$ a random value). Second, if there exist $(k, \vec{x}) \in \mathcal{R}$ and two transitions (k, g_1, a_1, k_1) and (k, g_2, a_2, k_2) such that $g_1(\vec{x})$ and $g_2(\vec{x})$ are both true, then two different states after $(k, \vec{x})$ are possible.

In this paper, we consider *affine interpreted automata*, a subclass of integer interpreted automata with *affine guarded transitions*. In this case, the guard is a conjunction of affine tests and the set of pairs $(\vec{x}, \vec{x}')$ where $\vec{x}' \in a(\vec{x})$ is described by a polyhedron.

Definition 1 (General affine guarded transition) *A transition $t = (k, g, a, k')$ is an* affine guarded transition *when:*

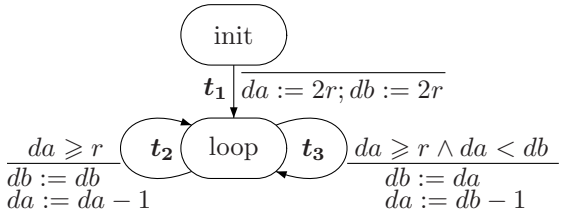


Figure 1: Affine interpreted automaton

- *$g(\vec{x})$ is true iff $G\vec{x} + \vec{g} \geq \vec{0}$ (component-wise) where $\vec{g}$ is an integer vector and G an integer matrix with n columns and as many rows as the size of $\vec{g}$;*
- *$\vec{x}' \in a(\vec{x})$ iff $A\vec{x} + A'\vec{x}' + \vec{a} \geq \vec{0}$ (component-wise) where $\vec{a}$ is an integer vector and A and A' are integer matrices with n columns and as many rows as the size of $\vec{a}$. We write $a = (A, A', \vec{a})$.*

The algorithms and theory we develop in the next sections can handle general affine interpreted automata. However, in terms of implementation, the tool we rely on for the computation of invariants (see Section 3.1) is, so far, limited to the case where actions define functions but with unspecified components: in other words, for each $\vec{x}$, $a(\vec{x})$ is a singleton $\{\vec{x}'\}$ where each component of $\vec{x}'$ is either expressed as an affine function of the components of $\vec{x}$, or unspecified (i.e., it can take any value in $\mathbb{Z}$), which we denote by the symbol "?". This will be sufficient to handle general (extended) affine relations, as we will see in Section 3.1. In this case, we identify $a(\vec{x})$ with the element $\vec{x}' \in (\mathbb{Z} \cup \{?\})^n$ it contains, and we say that $a(\vec{x})$ is an affine function of $\vec{x}$.

An example of such an automaton is given in Figure 1. The control points are labelled for convenience, and transitions are represented with arrows indexed by $\frac{g}{a}$ (g is omitted when $g = true$).

Example 1 The C code below is an abstraction of a real C code computing the greatest common divisor (gcd) of two polynomials.

```
// expr is an expression, A is an array,
// r is a constant positive integer parameter.
  da = 2r; db = 2r;a
  while (da >= r) {
    cond = ( da >= db || A[expr] == 0 );
    if (!cond) {
         tmp = db; db = da; da = tmp - 1;
    }
    else da = da - 1;
  }
```

This simplified code is itself abstracted by the automaton of Figure 1, where `init` is the initial control point and `loop` corresponds to the while loop. The fact that the condition `A[expr]==0` cannot be statically evaluated introduces some non-determinism in the automaton. To enter the `then` part of the test, the condition `da < db` must be true (transition t_3) while the `else` part can always be traversed (transition t_2) as long as the termination test of the while loop is false (additional condition `da >= r` for both transitions). □

Definition 2 (Bounded non-determinism) *An affine interpreted automaton has bounded non-determinism if there exists $B \in \mathbb{N}$, such that, for each state $(k, \vec{x})$, the number of states $(k', \vec{x}')$ that can be obtained after one transition from state $(k, \vec{x})$ is at most B.*

Invariants The guard g in a transition $t = (k, g, a, k')$ gives a necessary condition on variables $\vec{x}$ to traverse the transition t and to apply its corresponding action a. To get the exact valuations $\vec{x}$ of variables for which the action a can be performed, one needs to take into account the initial valuations and the successive conditions that led to the control point k. We denote by $\mathcal{R}_k$ the set of possible valuations $\vec{x}$ of variables when the control is in k:

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

Then, there exists a trace containing the transition (k, g, a, k') iff $\vec{x} \in \mathcal{R}_k$ and $g(\vec{x})$ is true. Note that $\mathcal{R}_k$ does not depend on any initial valuation. More precisely, it is the union, for all initial valuations $\vec{v}$, of the set of vectors $\vec{x}$ such that $(k, \vec{x})$ is reachable from $\vec{v}$.

In practice, it is difficult to determine the set $\mathcal{R}_k$ exactly but it is possible to give over-approximations, thanks to the notion of *invariants*. An invariant on a control point k is a formula ϕ_k that is true for all reachable states $(k, \vec{x})$:

$$\vec{x} \in \mathcal{R}_k \Rightarrow \phi_k(\vec{x}) \text{ is true.}$$

An invariant is *affine* if it is the conjunction of a finite number of affine conditions on program variables. The set $\mathcal{R}_k$ is then over-approximated by the integer points within a polyhedron $\mathcal{P}_k$. Some analyzes exist that can compute such a polyhedral approximation. We briefly present in Section 3.1 the technique and software tool we use in our implementation for deriving such affine invariants.

Example 1 (Cont'd) For the automaton of Figure 1, the vector $\vec{x}$ is equal to (r, da, db). The most precise affine invariants are:

- for the control point `init` : $\mathbb{N}^* \times \mathbb{Z}^2$.
- for `loop` : $\{(r, da, db) \mid 0 \leq r - 1 \leq da \leq 2r, r \leq db \leq 2r\}$. □

2.3 Termination and ranking functions

Invariants can only prove partial correctness of a program. The standard technique for proving termination is to consider ranking functions and well-founded sets, as we now recall.

A well-founded set $\mathcal{W}$ is a set with a (total or partial) order $\preceq$ (we write $a \prec b$ if $a \preceq b$ and $a \neq b$) such that there is no infinite descending chain, i.e., an infinite sequence $(x_i)_{i \in \mathbb{N}}$ such that $x_i \in \mathcal{W}$ and $x_{i+1} \prec x_i$ for all $i \in \mathbb{N}$.

Definition 3 (Ranking function) *A ranking function is a function $\rho : \mathcal{K} \times \mathbb{Z}^n \to \mathcal{W}$, from the automaton states to a well-founded set $(\mathcal{W}, \preceq)$, whose values decrease at each transition $t = (k, g, a, k')$:*

$$\vec{x} \in \mathcal{R}_k \wedge g(\vec{x}) = \mathit{true} \wedge \vec{x}' \in a(\vec{x}) \Rightarrow \rho(k', \vec{x}') \prec \rho(k, \vec{x}) \tag{1}$$

Definition 4 (1D and kD ranking function) *A ranking function ρ is one dimensional if its co-domain is $(\mathbb{N} \leq)$. It is multi-dimensional of dimension k (or k-dimensional) if its co-domain is $(\mathbb{N}^k \leq_{lex})$, where $\leq_{lex}$ is the standard lexicographic order on integer vectors.*

Obviously, the existence of a ranking function implies program termination for any valuation $\vec{v}$ at the initial control point k_{init}. The next lemma shows that this existence is also a necessary condition.

Lemma 1 *An integer interpreted automaton terminates for any initial valuation if and only if it has a ranking function. Furthermore, if it terminates and has bounded non-determinism, there is a one-dimensional ranking function, i.e., with co-domain* $(\mathbb{N}, \leq)$.

Proof If an integer interpreted automaton has a ranking function ρ with co-domain $(\mathcal{W}, \preceq)$, but does not terminate for some initial valuation $\vec{v}$, then there is an infinite trace $(k_n, \vec{x}_n)_{n\in\mathbb{N}}$ from $(k_{init}, \vec{v})$. As ρ is a ranking function, the sequence $\rho(k_n, \vec{x}_n)$ is an infinite descending chain in the well-founded set $(\mathcal{W}, \preceq)$, impossible.

Conversely, if an integer interpreted automaton terminates for any initial valuation, consider the set $\mathcal{W} = \mathcal{K} \times \mathbb{Z}^n$ of all its states and define the relation $\prec$ as follows: if $(k, \vec{x}) \in \mathcal{R}$ and (k, g, a, k') is a transition such that $g(\vec{x})$ is true, then define $(k', \vec{x}') \prec (k, \vec{x})$ for all $\vec{x}' \in a(\vec{x})$. Then, extend the relation $\prec$ to its transitive closure $\prec^+$. Note that $(k_1, \vec{x}) \prec^+ (k_2, \vec{y})$ iff there is a trace from $(k_1, \vec{x})$ to $(k_2, \vec{y})$ (this is true because the guards depend only on the current state and not on the traces that lead to it). Finally, consider the relation $\preceq^+$ defined by $x \preceq^+ y$ iff $x \prec^+ y$ or $x = y$. There is no infinite trace starting from a reachable state $(k, \vec{x})$, thus in particular no circuit, hence $\preceq^+$ is a partial order and it has no infinite descending chain. The identity function on $(\mathcal{W}, \preceq^+)$ is thus a ranking function.

Finally, suppose that the interpreted automaton terminates and has bounded non-determinism (see Definition 2). Then, there is a one-dimensional ranking function, i.e., from $\mathcal{K} \times \mathbb{Z}^n$ into $(\mathbb{N}, \leq)$. Indeed, consider a reachable state $(k, \vec{x})$. The previously-defined relation $\prec$, when restricting to states reachable from $(k, \vec{x})$, i.e., those smaller than $(k, \vec{x})$ for $\prec^+$, corresponds to a DAG rooted at $(k, \vec{x})$. This DAG has no infinite path since the automaton terminates. Furthermore, if its paths were of unbounded length, the DAG would be of infinite size, and, by König's lemma, this would contradict the bounded determinism hypothesis. Hence, we may define $\rho(k, \vec{x})$ as the maximum length of a path starting from state $(k, \vec{x})$. By construction, ρ is a one-dimensional ranking function. ■

Remark 1 We point out that there is a strong similarity between Lemma 1 and the existence of a free schedule for a computable system of uniform recurrence equations (SURE) defined by Karp, Miller, and Winograd [28, 19]. For a SURE, a computation $(k, \vec{x})$, defined for $\vec{x} \in \mathcal{D}_k$, *depends on* a bounded number of other computations $(k_i, \vec{x}_i)$, $1 \leq i < p_k$, where $\vec{x}_i = \vec{x} - \vec{d}_i$ and $\vec{d}_i$ is an integer vector. If $\vec{x}_i \notin \mathcal{D}_i$, $(k_i, \vec{x}_i)$ is available as an input, otherwise it has to be computed first. If this "unrolling" of computations terminates, $(k, \vec{x})$ can be computed. A *schedule* σ assigns a "date" (a nonnegative integer) to each $(k, \vec{x})$ such that $\sigma(k, \vec{x}) \geq \sigma(k_i, \vec{x}_i) + 1$ if $(k, \vec{x})$ depends on $(k_i, \vec{x}_i)$. The *free schedule* is the fastest one if each computation is assumed to take one unit of time. In a SURE and even in a SARE (system of *affine* recurrence equations for which $\vec{x}_i = a_i(\vec{x})$ where a_i is an affine function), each state $(k, \vec{x})$ depends on a bounded number of other states $(k_i, a(\vec{x}))$, $1 \leq i < p_k$. Therefore, as in Lemma 1, the free schedule exists for a computable SURE or SARE. In other words, SUREs and SAREs are modeled by a structure similar to an affine interpreted automaton. The main difference is in the interpretation, in terms of computations. In systems of recurrence equations, computations actually take place in the opposite order of transition traversals. Furthermore, *all* traces from a given state are followed. In an interpreted automaton, only *some* transitions

from a given state are performed ("non-determinism"), depending on the guards, and for each transition, only one $\vec{x}' \in a(\vec{x})$ is considered. But, in terms of properties and algorithms to build them, free schedule and ranking function are very similar.

Remark 2 The (countable) well-founded set $\mathcal{W}$ can be embedded in a countable ordinal (following set theory terminology). Each ordinal corresponds to a total order that can be fairly complicated to express or compute in practice. In general, algorithms for termination (or similarly for scheduling SUREs) look for special sets $\mathcal{W}$ and ranking functions of a certain class. For example, in [29], Lee studies ranking functions for a particular approximation of recursive programs, those with the so-called SCT property (size-change termination). According to this study, for SCT programs, it is sufficient to consider "minimums and maximums over lexicographic tuples", i.e., simple functions into the ordinal $\omega^d = (\mathbb{N}^d, \preceq_d)$, where $\preceq_d$ stands for the lexicographic order for vectors of size d.

Affine ranking functions In this paper, in the context of affine interpreted automata, we restrict ourselves to functions $\rho(k, \vec{x})$, affine for the second parameter (i.e., when k is fixed), and with co-domain $(\mathbb{N}^d, \preceq_d)$. Characterizing the smallest class of ranking functions (as well as "smallest" ordinal) needed for affine interpreted automata is, to our knowledge, an open problem. However, our algorithm is complete in the sense that it always finds an affine ranking if one exists. Furthermore, the dimension d is determined by the algorithm, and not *a priori*, and it is the smallest possible.

Note that Floyd [33] uses a condition weaker than Condition (1): let $\mathcal{H}$ be a subset of $\mathcal{K}$ such that any cycle of the automaton contains at least one control point in $\mathcal{H}$. Then to prove termination, it is enough to exhibit a function into a well-founded set, which decreases on all paths from one point in $\mathcal{H}$ to another. That a ranking satisfying Condition (1) can also serve in this context is obvious. Furthermore, since both techniques are equivalent to termination, the existence of a function ρ decreasing on paths implies the existence of a ranking function ρ', as defined in Section 2.3, i.e., decreasing on transitions. However, both techniques differ when restricting to particular well-founded sets or class of functions. The next lemma gives a more precise view of the difference between these two proof mechanisms for the simplest case of an interpreted automaton where all actions $a(\vec{x})$ define affine functions.

Given the set $\mathcal{H}$, for each transition $t = (k, g, a, k')$, we define $\mathcal{R}_{t,\mathcal{H}}$ as follows: $\vec{x} \in \mathcal{R}_{t,\mathcal{H}}$ iff $\vec{x} \in \mathcal{R}_k$ and there is a trace, starting with the transition t, from $(k, \vec{x})$ to $(h, \vec{y})$ for some $h \in \mathcal{H}$. To prove termination, it is sufficient to check that a ranking function decreases for transition t only for $\vec{x} \in \mathcal{R}_{t,\mathcal{H}}$ because if $\vec{x} \notin \mathcal{R}_{t,\mathcal{H}}$, any trace starting from $(k, \vec{x})$ through the transition t visits a finite number of control points (otherwise it would go through a cycle of the automaton, thus through some $h \in \mathcal{H}$).

Lemma 2 *Consider an integer interpreted automaton such that all actions $a(\vec{x})$ define a function. Suppose there is an affine function ρ into $(\mathbb{N}^d, \preceq_d)$, decreasing on all paths between two control points in $\mathcal{H}$. Then, there is a piecewise affine function ρ' into $(\mathbb{N}^{d+1}, \preceq_{d+1})$, decreasing on all transitions t for all states $(k, \vec{x})$ such that $\vec{x} \in \mathcal{R}_{t,\mathcal{H}}$.*

Proof Since each cycle of the automaton contains a point in $\mathcal{H}$, removing all outgoing edges for vertices in $\mathcal{H}$ leads to a DAG G. Given $k \in K$, let $h(k)$ be the height of k, i.e., the length of the longest path, in G, starting from k (if k has no successor, $h(k) = 0$). The function ρ is defined only for elements in $\mathcal{H}$. The ranking ρ' that we build has an

extra dimension. For $k \in \mathcal{H}$ and $\vec{x} \in \mathcal{R}_k$, we set $\rho'(k,\vec{x}) = \begin{pmatrix} \rho(k,\vec{x}) \\ 0 \end{pmatrix}$. We extend ρ' for all $k \in \mathcal{K} \setminus \mathcal{H}$, by induction on increasing values of $h(k)$. To make the notations simpler, we denote by $\rho(k,\vec{x})$ the first d dimensions of $\rho'(k,\vec{x})$, even for $k \notin \mathcal{H}$.

Assume ρ' is defined for all k' such that $h(k') < n$. Consider k such that $h(k) = n$ and $\vec{x} \in \mathcal{R}_k$. Let $t_i = (k, g_i, a_i, k_i)$, $i \in [1..p]$, be the transitions such that $\vec{x} \in \mathcal{R}_{t_i,\mathcal{H}}$. (If there is no such transition for $\vec{x}$, there is nothing to do.) We let:

$$\rho'(k,\vec{x}) = \begin{pmatrix} \max_{\preceq_d}\{\rho(k_i, a_i(\vec{x})) \mid i \in [1..p]\} \\ n+1 \end{pmatrix}.$$

Thanks to this inductive construction, ρ' is decreasing for each transition $t = (k, g, a, k')$, $k \notin \mathcal{H}$, and $\vec{x} \in \mathcal{R}_{t,\mathcal{H}}$. It remains to consider all transitions $t = (k, g, a, k')$ such that $k \in H$. Let $\vec{x} \in \mathcal{R}_{t,\mathcal{H}}$. By definition of $\mathcal{R}_{t,\mathcal{H}}$ and construction of ρ', there is $h \in H$ such that $\rho(k', a(\vec{x})) = \rho(h, A(\vec{x}))$ where A is the composition of all actions a along the path to h. The property of ρ on paths implies $\rho(k', a(\vec{x})) = \rho(h, A(\vec{x})) \prec_d \rho(k,\vec{x})$, thus $\rho'(k', a(\vec{x})) \prec_{d+1} \rho'(k,\vec{x})$ as desired. With this construction, if ρ is affine, then ρ' is piecewise affine due to the use of the maximum function for defining ρ'. ■

The previous lemma shows that there is an interaction between the class of functions we consider and the way we prove termination, either with a (global) ranking function, decreasing for each transition, or with a function decreasing on paths, as Floyd proposes. Also, constraining the ranking function to decrease even for $\vec{x} \notin \mathcal{R}_{t,\mathcal{H}}$ is an over-approximation. We will come back to this later.

3 Generating affine ranking functions for affine interpreted automata

3.1 Generating invariants

In this paper, we chose to generate invariants through the use of linear relation analysis [27]. We first recall the main characteristics of this method. We first make a restriction on the form of the actions we consider: we restrict to what we call *extended affine functions*.

Definition 5 (Extended affine function) *An extended affine function $a : Z^n \to (\mathbb{Z} \cup \{?\})^n$ assigns to the vector $\vec{x}$ an extended unique vector $\vec{x}'$ where for all i, $0 \le i < n$, $\vec{x}'[i] \in \mathbb{Z} \cup \{?\}$. The i-th component of $\vec{x}'$ is defined either as an affine function of the components of $\vec{x}$ or is non deterministic (i.e., it can take any value in $\mathbb{Z}$).*

As in Definition 1, the guards g of transitions are polyhedral guards of form $G\vec{x} + g \ge \vec{0}$ and actions are extended affine functions. These transitions are called *extended guarded affine transitions*.

Collecting semantics: As $\mathcal{R}_k$ is the set of possible variable valuations when the control is in k, we get, for each $k \in \mathcal{K}$:

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

where $\mathcal{R}_{k'} \cap g$ stands for $\{\vec{x} \in \mathcal{R}_{k'} \mid g(\vec{x}) = \text{true}\}$ and $a(\mathcal{R}_{k'} \cap g)$ stands for $\{a(\vec{x}) \mid \vec{x} \in \mathcal{R}_{k'} \wedge g(\vec{x}) = \text{true}\}$. The initial invariant can be refined if some information is provided at program start as for Example 1 for the condition $r \ge 1$.

Iterative computations The linear relation analysis consists in attaching to each control point k a polyhedron P_k, which is an over-approximation of $\mathcal{R}_k$ (thus, $\vec{x} \in P_k$ is an *invariant*), as follows:

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

where:

- $\top$ denotes the polyhedron representing $\mathbb{R}^n$,
- $\sqcup$ and $\sqcap$ denote respectively the convex union (convex hull) and the intersection of polyhedra,
- $a(P_k) = \{a(\vec{x}) \mid \vec{x} \in P_k\}$ is computed by first applying the affine part of a to P_k, and then eliminating the "undefined variables" (whose value is specified by the symbol ?).

Efficient polyhedral libraries exist for computing these operations, such as Polylib [1], PPL [2], and the polyhedra part of APRON [3]. We are then left with a system of fixpoint equations (with $m = \#\mathcal{K}$):

$$\forall k \in \mathcal{K}, P_k = F_k(P_1, \ldots, P_k, \ldots, P_m)$$

whose least solution exists (by Tarski's theorem) and can be computed by iteration from $P_k = \bot = \emptyset$, $\forall k$. Of course, this computation will raise the problem of convergence. This problem is solved thanks to the *widening operator* ∇. The general definition of a widening operator [14] ensures the convergence of the sequence

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

The functions F_k^{∇} are defined with a subset $\mathcal{K}_{\nabla} \subseteq \mathcal{K}$ of "cutting nodes" such that each loop in the program contains at least one control point in $\mathcal{K}_{\nabla}$ (this is similar to the set $\mathcal{H}$ used in the previous section for Floyd's termination proof method). Then

$$F_k^{\nabla} = \begin{cases} P_k \nabla F_k(P_1, \ldots, P_k, \ldots, P_m) & \text{if } k \in \mathcal{K}_{\nabla} \\ F_k(P_1, \ldots, P_k, \ldots, P_m) & \text{otherwise} \end{cases}$$

Finally, the limit, obtained after a finite number of steps, is an over-approximation of the least solution of the initial system.

In linear relation analysis, the set of *widening nodes* $\mathcal{K}_{\nabla}$ is usually defined as the set of entry nodes of the strongly connected components of the graph. We also use the *standard widening* on polyhedra, as defined in [15] and [27], which roughly consists in keeping for $P_1 \nabla P_2$ only the constraints of P_1 also satisfied by P_2.

Aspic tool To compute invariants, we chose to use the tool Aspic [4], which provides an implementation of *abstract acceleration* [23]. Compared to the standard widening approach, this method computes the exact reachability set for "accelerable" loops, which locally avoids the use of widening and globally increase precision. Aspic takes as input an affine counter automaton and generates an affine invariant for each control point.

[1] `http://icps.u-strasbg.fr/polylib/`
[2] `http://www.cs.unipr.it/ppl/`
[3] `http://apron.cri.ensmp.fr/library/`
[4] `http://laure.gonnord.org/pro/aspic/aspic.html`

Example 1 (Cont'd) With the initial condition $r \geq 1$ and loop exit to `end` with guard $r > da$, Aspic produces the following invariants:

- $P_{init} = \{1 \leq r\}$;
- $P_{loop} = \{da \leq 2r, db \leq 2r, 1 \leq r, r \leq db, r - 1 \leq da\}$;
- $P_{end} = \{da < r, db \leq 2r, 1 \leq r, r \leq db, r - 1 \leq da\}$. □

We extended Aspic with a `-ranking` option, which initializes each variable to an unknown value at the initial control point. The discovered invariants are then parameterized by these initial values.

General affine relations The use of Aspic does not restrict the fragment of programs we are able to deal with since we can encode a general affine transition (see Definition 1) into a combination of two extended affine transitions (see Definition 5).

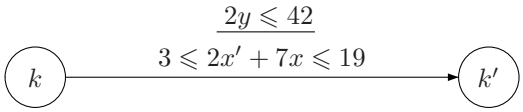
Lemma 3 *A general affine transition (as defined in Definition 1) can be encoded by the sequence of two extended affine transitions.*

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

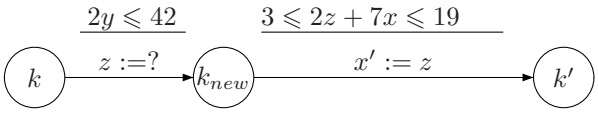
- $t_1 = (k, g, a_1, k_{new})$ with $a_1(\vec{z}) = (?)^n$
- $t_2 = (k_{new}, g_2, a_2, k')$ where $g_2(\vec{x}, \vec{z})$ is true iff $\vec{z} \in a(\vec{x})$ (affine guard) and $a_2(\vec{x}) = \vec{z}$ (projection).

In other words, an affine relation that does not define a function is encoded in the guard of a transition. ■

Let us illustrate this construction. The following transition:



is equivalent to the following couple of transitions:



3.2 Computing affine ranking functions

This section presents an algorithm to construct multi-dimensional affine ranking functions, i.e., ranking functions $\rho : \mathcal{K} \times \mathbb{Z} \to \mathbb{N}^d$, affine for the second parameter. The integer d is the dimension of the ranking. For the sake of clarity, we first explain in Section 3.2.1 the particular case $d = 1$, i.e., how to get a *one-dimensional* affine ranking function. The method is then generalized in Section 3.2.2 to compute an affine ranking function of dimension $d > 1$.

Note: as we use linear programming (but not integer linear programming), we may end up with functions with rational components. Then, we can always multiply such a function by a suitable integer to get a ranking function with integer values. Thus, in the rest of the paper, we will not insist on this subtlety any longer.

3.2.1 One-dimensional affine ranking functions

As explained in Section 3.1, in practice, the exact sets $\mathcal{R}_k$ are not necessarily available. They are over-approximated by invariants $\mathcal{P}_k$, with $\mathcal{R}_k \subseteq \mathcal{P}_k$, which are, in our case, described by polyhedra. The conditions that a ranking function must satisfy are then related to these invariants and not to the exact sets of reachable states.

A one-dimensional ranking function ρ has co-domain $\mathbb{N}$, i.e., it assigns a nonnegative integer to each relevant state:

$$\vec{x} \in \mathcal{P}_k \Rightarrow \rho(k, \vec{x}) \geq 0 \tag{2}$$

Consider Inequality (1), which specifies that the ranking decreases on transitions. Let $\mathcal{Q}_t$ be the polyhedron described by the constraints of a transition $t = (k, g, a, k')$, i.e., $\vec{x} \in \mathcal{P}_k$, $g(\vec{x})$ is true, and $\vec{x}' \in a(\vec{x})$. For an affine interpreted automaton, $\mathcal{Q}_t$ is directly built from matrices A, A', and G, and vectors $\vec{a}$ and $\vec{g}$ (see Definition 1 in Section 2.2). Inequality (1) becomes:

$$\vec{y} = (\vec{x}, \vec{x}') \in \mathcal{Q}_t \Rightarrow \rho(k, \vec{x}) - \rho(k', \vec{x}') \geq 1 \tag{3}$$

More general cases can be handled the same way if the set of pairs $(\vec{x}, \vec{x}')$ is over-approximated by a polyhedron. Now, it remains to translate Inequalities (2) and (3) into a linear system and to get a ranking function by means of a linear solver. Classically, we use the affine form of Farkas lemma [37] to linearize the constraints:

Lemma 4 (Farkas lemma, affine form) *An affine form $\phi : \mathbb{R}^n \to \mathbb{R}$ with $\phi(\vec{x}) = \vec{c}.\vec{x} + c_0$ is nonnegative everywhere in a non-empty polyhedron $\{\vec{x} \mid A\vec{x} + \vec{a} \geq \vec{0}\}$ iff:*

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

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

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

For all $\vec{x}$, $\phi(\vec{x})$ is thus expressed as the dot product of $\vec{x}$ and a nonnegative combination of the normals of the facets of $\mathcal{P}$, plus a nonnegative constant. We can now apply the affine form of Farkas lemma to Inequalities (2) and (3). Write $P_k\vec{x} + \vec{p_k} \geq \vec{0}$ the constraints that define $\mathcal{P}_k$. For Inequality (2), we get:

$$\exists \vec{\lambda_k} \in (\mathbb{R}^+)^n, \lambda_k^0 \in \mathbb{R}^+ \text{ such that } \rho(k, \vec{x}) \equiv \vec{\lambda_k}.(P_k\vec{x} + \vec{p_k}) + \lambda_k^0 \tag{4}$$

Similarly, let us write $\mathcal{Q}_t = \{\vec{y} = (\vec{x}, \vec{x}') \mid Q_t\vec{y} + \vec{q_t} \geq \vec{0}\}$. We call $\Delta_t(\rho, \vec{x}, \vec{x}') = \rho(k, \vec{x}) - \rho(k', \vec{x}')$ the *delay* of transition t. Inequality (3) states that $\Delta_t(\rho, \vec{x}, \vec{x}') \geq 1$, which means:

$$\exists \vec{\mu_t} \in (\mathbb{R}^+)^n, \mu_t^0 \in \mathbb{R}^+ \text{ s.t. } \Delta_t(\rho, \vec{x}, \vec{x}') - 1 \equiv \vec{\mu_t}.(Q_t\vec{y} + \vec{q_t}) + \mu_t^0 \tag{5}$$

A substitution of (4) in (5) and an identification on each dimension of $\vec{y}$ leads to a linear system $\mathcal{S}_t$ with nonnegative unknowns $\vec{\lambda_k}$, λ_k^0 (from $\rho(k, \vec{x})$), $\vec{\lambda_{k'}}$, $\lambda_{k'}^0$ (from $\rho(k', \vec{x}')$), $\vec{\mu_t}$, and μ_t^0. Finally, concatenating all $\mathcal{S}_t$ for all transitions $t \in \mathcal{T}$ gives a linear system $\mathcal{S}$, with nonnegative unknowns $(\vec{\lambda_k}, \lambda_k^0)_{k \in \mathcal{K}}$ and $(\vec{\mu_t}, \mu_t^0)_{t \in \mathcal{T}}$, that characterizes all one-dimensional affine ranking functions for the automaton.

Example 1 (Cont'd) As explained before, Aspic provides the following invariants (the initial values are denoted by r_0, da_0, db_0):

- $P_{init} = \{1 \leq r_0 = r, db = db_0, da = da_0\}$
- $P_{loop} = \{da \leq 2r, db \leq 2r, r \leq db, r - 1 \leq da, 1 \leq r = r_0\}$
- $P_{end} = \{da < r, db \leq 2r, r \leq db, r - 1 \leq da, 1 \leq r = r_0\}$

For readability in the following systems, for $i \geq 1$ and $k \in \mathcal{K}$, we write λ_k^i instead of $\vec{\lambda}_k[i-1]$, same for $\vec{\mu}$. Also $\vec{x}$ stands for the vector $(da, da_0, db, db_0, r, r_0)$. The subsystem (4) is obtained by applying Farkas lemma to P_{init}, P_{loop}, and P_{end} (here, with no simplification):

(4i) $\rho(init, \vec{x}) = \lambda^0_{init} + \lambda^1_{init}(r-1) + \lambda^2_{init}(r_0-1) + \lambda^3_{init}(r_0-r) + \lambda^4_{init}(r-r_0) + \lambda^5_{init}(db_0 - db) + \lambda^6_{init}(db-db_0) + \lambda^7_{init}(da_0-da) + \lambda^8_{init}(da-da_0)$ with $\lambda^i_{init} \geq 0$ for all $i \in [0..8]$;

(4ii) $\rho(loop, \vec{x}) = \lambda^0_{loop} + \lambda^1_{loop}(2r-da) + \lambda^2_{loop}(2r-db) + \lambda^3_{loop}(r-1) + \lambda^4_{loop}(db-r) + \lambda^5_{loop}(da-r-1) + \lambda^6_{loop}(r_0-r) + \lambda^7_{loop}(r-r_0)$ with $\lambda^i_{loop} \geq 0$ for all $i \in [0..7]$;

(4iii) and a similar formula for the control point end.

To get the subsystem (5), we first compute Q_t (here in logical form):

- $Q_{t_1} = \vec{x} \in P_{init} \wedge da' = 2r \wedge db' = 2r \wedge r' = r$;
- $Q_{t_2} = \vec{x} \in P_{loop} \wedge r \leq da \wedge da' = da - 1 \wedge db' = db \wedge r' = r$;
- $Q_{t_2} = \vec{x} \in P_{loop} \wedge r \leq da < db \wedge da' = db - 1 \wedge db' = da \wedge r' = r$.

And, finally, we obtain:

(5i) $\rho(init, \vec{x}) - \rho(loop, \vec{x}') - 1 = \mu^0_{t_1} + \mu^1_{t_1}(r_0-1) + \mu^2_{t_1}(r-r_0) + \mu^3_{t_1}(r_0-r) + \ldots + \mu^7_{t_1}(da'-2r) + \mu^8_{t_1}(2r-da') + \mu^9_{t_1}(db'-2r) + \mu^{10}_{t_1}(2r-db') + \ldots$ with $\mu^i_{t_1} \geq 0$ for all i;

(5ii) and similar expressions for transitions t_2 and t_3.

In (5i), we replace the expressions involving ρ by the values obtained in (4i) and (4ii). We do the same in the other expressions coming from (5). We obtain a conjunction of conditions involving the different $\vec{\lambda}_k$ and $\vec{\mu}_k$. The solver finally produces:

- $\rho(init, \vec{x}) = 2r_0 + 3$;
- $\rho(loop, \vec{x}) = 2 + da + db - 2r$;
- $\rho(end, \vec{x}) = 0$.

which means that the loop terminates in at most $2r_0 + 3$ steps (including at most $2r_0 + 1$ iterations of the while loop). The quantity $i = 2 + da + db - 2r$, which is automatically extracted, can be viewed as a kind of counter for the while loop. □

Solving the system $\mathcal{S}$ gives a termination test that will work providing the invariants are accurate enough. Note however that, since ρ is, so far, affine and one-dimensional, there is no hope, with this method, to be able to determine the termination of an automaton whose *worst time complexity* (WTC), i.e., maximal trace length, is more than linear in the input variables. The extension in the next section can determine the termination of some programs with a non-linear (but still polynomial) time complexity.

3.2.2 Multi-dimensional affine ranking functions

Using multi-dimensional affine ranking functions, i.e., affine ranking functions with codomain $(\mathbb{N}^d, \preceq_d)$ for some integer d, extends the set of programs whose termination can be determined. Indeed, some terminating programs with polynomial time complexity (of degree $\leq d$) can now be handled, while no one-dimensional affine ranking function exists for them. Furthermore, when it exists, a polynomial WTC can be derived from the ranking, with a simpler method than by manipulating directly polynomials of degree d.

For a d-dimensional ranking function ρ, the decreasing constraint expressed by Inequality (1) becomes:

$$(\vec{x}, \vec{x}') \in Q_t \Rightarrow \Delta_t(\rho, \vec{x}, \vec{x}') \succ_d \vec{0} \tag{6}$$

which means that $\Delta_t(\rho, \vec{x}, \vec{x}') \neq \vec{0}$ and its first nonzero component is positive. Unlike for a one-dimensional ranking function where we look for a function ρ such that $\rho(k, \vec{x}) - \rho(k', \vec{x}') \geq 1$, here we consider each dimension $\sigma = \rho[i]$ of ρ with the relaxed constraint:

$$(\vec{x}, \vec{x}') \in Q_t \Rightarrow \sigma(k, \vec{x}) - \sigma(k', \vec{x}') \geq \varepsilon_t \tag{7}$$

with $0 \leq \varepsilon_t \leq 1$. As we did in Section 3.2.1, we obtain a linear system $\mathcal{S}_t(\vec{\lambda}_k, \lambda_k^0, \vec{\lambda}_{k'}, \lambda_{k'}^0, \vec{\mu}_t, \mu_t^0, \varepsilon_t)$ for each transition, this time with the additional unknown ε_t. Then, considering all transitions, we get a global system $\mathcal{S}$. For a solution σ of $\mathcal{S}$, we say that a transition t is *satisfied* if $\varepsilon_t = 1$, otherwise, it is *respected*.

To build a multi-dimensional ranking ρ, we use a greedy algorithm, similar to the scheduling algorithm of [20], that builds the different dimensions of ρ, one after the other, from dimension 0 to dimension $d-1$, respecting unsatisfied transitions until all are satisfied for some dimension. Furthermore, for each dimension, we try to satisfy as many transitions as possible by maximizing the number of ε_t equal to 1. This boils down to solving the following optimization:

$$m = \max \left\{ \sum_{t \in \mathcal{T}} \varepsilon_t \mid \bigwedge_{t=(k,g,a,k') \in \mathcal{T}} \mathcal{S}_t(\vec{\lambda}_k, \lambda_k^0, \vec{\lambda}_{k'}, \lambda_{k'}^0, \vec{\mu}_t, \mu_t^0, \varepsilon_t) \right\} \tag{8}$$

It is easy to see that any solution leading to some $\varepsilon_t > 0$ can be multiplied by a suitable positive constant to a solution with $\varepsilon_t = 1$. Thus, for any *optimal* solution of (8), a transition t satisfies either $\varepsilon_t = 0$ or $\varepsilon_t = 1$. Similarly, one can build an integer optimal solution from any rational optimal solution.

The variables $(\vec{\lambda}_k)_{k \in \mathcal{K}}$ define an affine function that we use as the first dimension of ρ. By construction, $\rho[0]$ satisfies (3) for every transition t such that $\varepsilon_t = 1$. Other transitions are simply respected ($\varepsilon_t = 0$). We process them in the subsequent dimensions. For that, we build a new system $\mathcal{S}$, obtained by concatenating the different $\mathcal{S}_t$, but only for the transitions t with $\varepsilon_t = 0$, and we iterate the process. This way, we define $\rho[1]$, $\rho[2]$, and so on. When none of the remaining transitions can be satisfied, i.e., when the maximum m is 0, the algorithm stops and no multi-dimensional ranking function is found. Otherwise, as the number of transitions is finite, all of them are eventually satisfied, after a finite number of steps, by some dimension of ρ. In this case, the algorithm outputs, for each $k \in \mathcal{K}$, an affine function $\rho(k, .) : \mathbb{Z}^n \to \mathbb{N}^{d_k}$, where d_k is the number of successive systems, involving the control point k, that were solved. We can complete each $\rho(k, .)$ with arbitrary additional dimensions so that they all have co-domain $\mathbb{N}^d$, with $d = \max_k d_k$. This defines an affine ranking function ρ of dimension d.

Example 2 Consider the automaton of Figure 2 with the context $N \geq 0$ in control point k_0. Aspic finds the following invariants:

- $P_{k_0} = \{0 \leq N = N_0, j = j_0, i = i_0\}$;
- $P_{k_1} = \{0 \leq N = N_0, 0 \leq i \leq N\}$;
- $P_{k_2} = \{0 \leq N = N_0, 1 \leq i \leq N, 0 \leq j \leq N\}$.

Solving the system while maximizing $\varepsilon_{t_1} + \varepsilon_{t_2} + \varepsilon_{t_3} + \varepsilon_{t_4}$ leads to $\varepsilon_{t_1} = \varepsilon_{t_2} = \varepsilon_{t_4} = 1$ and $\varepsilon_{t_3} = 0$. The corresponding function (first dimension of the ranking) is $\rho(k_0, \vec{x})[0] =$

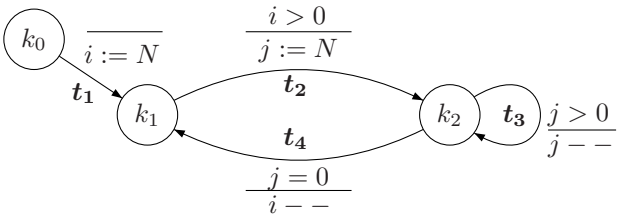


Figure 2: Automaton with a multi-dimensional ranking

$2N + 1$, $\rho(k_1, \vec{x})[0] = 2i$, $\rho(k_2, \vec{x})[0] = 2i - 1$. We keep the transition t_3 for the next dimension, and we get $\varepsilon_{t_3} = 1$ with $\rho(k_2, \vec{x})[1] = j$. The complete ranking function is thus $\rho(k_0, \vec{x}) = 2N + 1$, $\rho(k_1, \vec{x}) = 2i$, $\rho(k_2, \vec{x}) = (2i - 1, j)$. Note that, for k_0 and k_1, the ranking is only one-dimensional. It can be extended arbitrarily, for example as $\rho(k_0, \vec{x}) = (2N + 1, 0)$ and $\rho(k_1, \vec{x}) = (2i, 0)$, if a globally 2D ranking function is desired. □

3.3 Completeness

Since non-terminating programs exist, there is no hope of proving that a ranking function always exists. Moreover, there are terminating affine interpreted automata with no multi-dimensional affine ranking. Thus, all we can prove is that, if a multi-dimensional affine ranking exists, our algorithm finds one. This section gives such a completeness result. Also, as the sets $\mathcal{R}_k$ are over-approximated by invariants $\mathcal{P}_k$, completeness has to be understood with respect to these invariants. We first need a few definitions and lemmas.

We say that a lexicographically positive vector $\vec{x}$ is of *level i* if its i-th component is its first nonzero component: $\vec{x}[j - 1] = 0$ if $j < i$ and $\vec{x}[i - 1] > 0$. Without loss of generality, we assume that all d-dimensional ranking functions we consider are such that d equals ℓ, the maximal level of all $\Delta_t(\rho, \vec{x}, \vec{x}')$. Indeed, all dimensions $\rho[i]$ for $i \geq \ell$ are useless to decrease on transitions. We can also define the *sum* of two multi-dimensional ranking functions of dimensions d_1 and d_2, even if $d_1 \neq d_2$, as stated by the next lemma.

Lemma 5 *Let ρ_1 and ρ_2 be two multi-dimensional ranking functions of dimensions d_1 and d_2. One can define a ranking ρ of dimension $d \leq \min(d_1, d_2)$ such that $\rho[i] = \rho_1[i] + \rho_2[i]$, for all $i < d$.*

Proof Assume, without loss of generality, $d_1 \leq d_2$. We can first extend ρ_1 by $d_2 - d_1$ null dimensions and compute $\rho_1 + \rho_2$. Since delays are linear in the ranking functions and since the sum of two lexicopositive vectors is lexicopositive, $\rho_1 + \rho_2$ is a ranking function. Note that if $\vec{u}_1$ and $\vec{u}_2$ are both lexicopositive with levels l_1 and l_2, then the level of $\vec{u}_1 + \vec{u}_2$ is $\min(l_1, l_2)$. Thus, the maximal level d of all $\Delta_t(\rho, \vec{x}, \vec{x}')$ is at most $\min(d_1, d_2)$. It follows that all dimensions of $\rho_1 + \rho_2$ beyond $d \leq d_1$ are useless and can be removed. ■

A consequence of Lemma 5 related to our greedy algorithm is that, if a transition t_1 (resp. t_2) is satisfied by the first dimension of a ranking ρ_1 (resp. ρ_2), then the first dimension of $\rho_1 + \rho_2$ satisfies both transitions. Thus, there is no trade-off to make when deciding which transitions to satisfy at the dimension being considered.

Lemma 6 *If ρ is a d-dimensional ranking function and T a lower triangular $d \times d$ matrix, with positive diagonal, then $T\rho$, defined by $(T\rho)(\vec{x}) = T(\rho(\vec{x}))$, is a ranking function.*

Proof We have $\Delta_t(T\rho, \vec{x}, \vec{x}') = T\Delta_t(\rho, \vec{x}, \vec{x}')$. Let ℓ be the level of $\Delta_t(\rho, \vec{x}, \vec{x}')$. For $i < \ell - 1$, $\Delta_t(T\rho, \vec{x}, \vec{x}')[i] = \Delta_t(\rho, \vec{x}, \vec{x}')[i] = 0$. For $i = \ell - 1$, $\Delta_t(T\rho, \vec{x}, \vec{x}')[i] = T_{i,i} \times \Delta_t(\rho, \vec{x}, \vec{x}')[i] > 0$. Thus $\Delta_t(T\rho, \vec{x}, \vec{x}')$ is lexicographically positive and its level is ℓ. The remaining components are irrelevant. ■

Lemma 6 will be used to show that if a transition is only *partially* satisfied (i.e., $\varepsilon_t = 0$ but there are some $\vec{x}$ and $\vec{x}'$ such that $\Delta_t(\rho[0], \vec{x}, \vec{x}') \geq 1$), then we can still keep considering these satisfied pairs $(\vec{x}, \vec{x}')$ for further dimensions. This will not prevent the algorithm to find an affine ranking if one exists.

We now need a technical lemma, which generalizes the affine form of Farkas lemma to lexicographic order. Under its hypotheses, Lemma 7 states that we can always replace an affine function of dimension d by a linear combination of its dimensions (thus a function of dimension 1) and get the same "satisfied" vectors.

Lemma 7 *Let $\mathcal{P} = \{\vec{x} \mid A\vec{x} + \vec{a} \geq \vec{0}\}$ be a polyhedron and δ a d-dimensional affine function such that, $\forall \vec{x} \in \mathcal{P}$, $\delta(\vec{x}) \succeq_d \vec{0}$ and not all $\delta(\vec{x})$, $\vec{x} \in \mathcal{P}$, have level $< d$. Then, there is $\vec{\alpha}$ such that, $\forall \vec{x} \in \mathcal{P}$, $\vec{\alpha}.\delta(\vec{x}) \geq 0$ and $\vec{\alpha}.\delta(\vec{x}) \neq 0$ if $\delta(\vec{x}) \neq \vec{0}$. More precisely, there is a matrix Λ whose rows are lexicographically nonnegative and such that any $\vec{\alpha}$ is suitable, provided that $\vec{\alpha}[d-1] > 0$, $\Lambda\vec{\alpha} \geq \vec{0}$ and $(\Lambda\vec{\alpha})[i] = 0$ only if the corresponding row of Λ is $\vec{0}$.*

Proof Let $\mathcal{P}_i$ be the set of $\vec{x} \in \mathcal{P}$ such that $\delta(\vec{x})$ has level $> i$. By definition, $\mathcal{P}_0 = \mathcal{P}$ and $\mathcal{P}_{i+1} = \mathcal{P}_i \cap \{\vec{x} \mid \delta[i](\vec{x}) = 0\}$. Since not all $\delta(\vec{x})$ have level $< d$, $\mathcal{P}_i \neq \emptyset$ for all $i < d$.

For all $\vec{x} \in \mathcal{P}_0$, $\delta[0](\vec{x}) \geq 0$. Since $\mathcal{P}_0 \neq \emptyset$, by Farkas lemma, $\delta[0](\vec{x}) \equiv \vec{\lambda}_0.(A\vec{x} + \vec{a}) + r_0$ with $\vec{\lambda}_0 \geq \vec{0}$ and $r_0 \geq 0$. If $d > 1$, $\delta[0](\vec{x}_0) = 0$ thus $r_0 = 0$. Furthermore, if $\vec{x} \in \mathcal{P}_0$, $\delta[0](\vec{x}) = 0$ iff $(A\vec{x} + \vec{a})[i] = 0$ for each i such that $\vec{\lambda}_0[i] > 0$:

$$\mathcal{P}_1 = \{\vec{x} \mid A\vec{x} + \vec{a} \geq \vec{0}; (A\vec{x} + \vec{a})[i] = 0 \text{ if } \vec{\lambda}_0[i] > 0\}$$

where some inequalities that define $\mathcal{P}_0$ are now equalities. The same study can now be done for $\mathcal{P}_1$ if $d > 2$. The only difference is that, because of the equalities (called saturated inequalities), Farkas lemma leads to $\delta[1](\vec{x}) \equiv \vec{\lambda}_1.(A\vec{x} + \vec{a})$ with $\vec{\lambda}_1[i] \geq 0$ if $\vec{\lambda}_0[i] = 0$ (standard inequality), otherwise $\vec{\lambda}_1[i]$ can be of any sign if $\vec{\lambda}_0[0] > 0$ (saturated inequality). Then, for $\vec{x} \in \mathcal{P}_1$, $\delta[1](\vec{x}) = 0$ iff $(A\vec{x} + \vec{a})[i] = 0$ for each i such that $\vec{\lambda}_0[i] = 0$ and $\vec{\lambda}_1[i] > 0$. Finally, by induction, we get the following properties, for all $0 < j < d$:

- $\mathcal{P}_j = \mathcal{P}_{j-1} \cap \{\vec{x} \mid (A\vec{x} + \vec{a})[i] = 0 \text{ if } \vec{\lambda}_{j-1}[i] > 0\}$, i.e., $\mathcal{P}_j = \mathcal{P} \cap \{\vec{x} \mid (A\vec{x} + \vec{a})[i] = 0 \text{ if } \exists k<j \text{ s. t. } \vec{\lambda}_k[i] > 0\}$;
- $\delta[j](\vec{x}) \equiv \vec{\lambda}_j.(A\vec{x} + \vec{a}) + r_j$;
- $r_j = 0$ if $j < d - 1$, $r_{d-1} \geq 0$;
- $\vec{\lambda}_j[i] \geq 0$ if $\vec{\lambda}_k[i] = 0$ for all $k < j$.

The last property can be interpreted as follows. If Λ is the matrix whose columns are the vectors $\vec{\lambda}$, then each row of Λ is lexicographically nonnegative. Thus, there exists $\vec{\alpha}$ such that $\Lambda\vec{\alpha} \geq 0$ (component-wise) and $(\Lambda\vec{\alpha})[i] = 0$ only if the corresponding row

of Λ is equal to $\vec{0}$. For that, we just need to define the components of $\vec{\alpha}$ successively, from the last one to the first one. Indeed, a row i of Λ of level $j+1$, thus with $\Lambda_{i,j} > 0$ and $\Lambda_{i,k} = 0$ for $k < j$, corresponds to a constraint $\sum_{k \leq j} \Lambda_{i,k} \vec{\alpha}[k] > 0$, i.e., it gives a lower bound on $\vec{\alpha}[j]$, once all $\vec{\alpha}[k]$, for $k > j$, have been fixed:

$$\vec{\alpha}[j] > -\frac{\sum_{k>j} \vec{\lambda}_k[i]\vec{\alpha}[k]}{\vec{\lambda}_j[i]}$$

We can also choose $\vec{\alpha}$ such that $\vec{\alpha}[d-1] > 0$.

Now, for such $\vec{\alpha}$, $\vec{\alpha}.\delta(\vec{x}) = (\Lambda\vec{\alpha}).(A\vec{x}+\vec{d}) + \vec{\alpha}[d-1]r_{d-1}$, thus $\vec{\alpha}.\delta(\vec{x}) \geq 0$ for all $\vec{x} \in \mathcal{P}$. It remains to show that $\vec{\alpha}.\delta(\vec{x}) \neq 0$ if $\delta(\vec{x}) \neq \vec{0}$. Let ℓ be the level of $\delta(\vec{x})$. If $\ell < d$, $\vec{x} \in \mathcal{P}_{\ell-1}$ and $\vec{x} \notin \mathcal{P}_\ell$. Thus $\delta[\ell-1](\vec{x}) > 0$ and there is i such that $\vec{\lambda}_{\ell-1}[i] > 0$ and $(A\vec{x}+\vec{d})[i] > 0$. Thus, $\Lambda_{i,\ell-1} > 0$, then $\vec{\alpha}[i] > 0$, and finally $\vec{\alpha}.\delta(\vec{x}) > 0$. If $\ell = d$, the situation is the same except that it is possible that $\vec{\lambda}_{\ell-1}.(A\vec{x}+\vec{d}) = 0$. But, in this case, we must have $r_{d-1} > 0$ and again $\vec{\alpha}.\delta(\vec{x}) > 0$. ■

Note that our algorithm, described in Section 3.2.2, looks for a function that always fully satisfies at least one transition, otherwise it stops. Lemma 8 shows that this strategy is not a restriction, at least for the first dimension.

Lemma 8 *If an affine interpreted automaton has an affine ranking function ρ, then there is an affine ranking function ρ', of same dimension, whose first dimension fully satisfies at least one transition, i.e., $\exists\, t \in \mathcal{T}$ such that $\forall (\vec{x}, \vec{x}') \in Q_t$, $\Delta(\rho'[0], \vec{x}, \vec{x}') \geq 1$.*

Proof The trick is to use Lemma 7 to replace the first dimension of ρ by a linear combination of the first $\ell_{\min}$ dimensions that, all together, satisfy all constraints of at least one transition. $\ell_{\min}$ is defined as follows. For each transition t, each $\Delta_t(\rho, \vec{x}, \vec{x}')$ is a lexicographically positive vector. Let ℓ_t be the maximal level of all $\Delta_t(\rho, \vec{x}, \vec{x}')$ for $(\vec{x}, \vec{x}') \in Q_t$. Then $\ell_{\min}$ is the minimum of all ℓ_t.

For each transition $t = (k, g, a, k')$, let δ_t be the affine function of dimension $\ell_{\min}$ defined, for all $\vec{y} = (\vec{x}, \vec{x}')$, by the first $\ell_{\min}$ components of $\rho(k, \vec{x}) - \rho(k', \vec{x}')$: δ_t satisfies the hypotheses of Lemma 7 for Q_t. Let Λ be the matrix formed by all rows of the matrices Λ_t associated to the different δ_t. Its rows are all lexicographically nonnegative, thus again, we can select $\vec{\alpha}$ such that, for all $t \in \mathcal{T}$ and for all $\vec{y} \in Q_t$, $\vec{\alpha}.\delta_t(\vec{y}) \geq 0$ and $\vec{\alpha}.\delta_t(\vec{y}) \neq 0$ if $\delta_t(\vec{y}) \neq \vec{0}$.

We now come back to ρ. We extend $\vec{\alpha}$, with 0s, to get a vector with same dimension as ρ. Then $\vec{\alpha}.\rho$ is a linear combination of the first $\ell_{\min}$ dimensions of ρ. By construction, if $t \in \mathcal{T}$ and $(\vec{x}, \vec{x}') \in Q_t$, $\Delta(\vec{\alpha}.\rho, \vec{x}, \vec{x}') \geq 0$ and $\Delta(\vec{\alpha}.\rho, \vec{x}, \vec{x}') > 0$ if the level of $\Delta(\rho, \vec{x}, \vec{x}')$ is $\leq d$. Thus, replacing $\rho[0]$ by $\vec{\alpha}.\rho$ defines a ranking ρ' whose first dimension fully satisfied all transitions t such that $\ell_t = \ell_{\min}$.

Note: so far, we did not enforce $(\vec{\alpha}.\rho)(k, \vec{x}) \geq 0$ for all $\vec{x} \in \mathcal{P}_k$. We can handle this additional constraint thanks to Lemma 7 again. We can also simply impose directly $\vec{\alpha} \geq \vec{0}$ (component-wise). ■

We can now use all lemmas to prove the algorithm completeness.

Theorem 1 *If an affine interpreted automaton, with associated invariants, has an affine ranking function, then the algorithm of Section 3.2.2 finds one and its dimension is minimal.*

Proof Consider an affine ranking function ρ of dimension d. According to Lemma 8, there is an affine ranking function ρ' of dimension d that fully satisfies at least one transition. Thus, the algorithm of Section 3.2.2 succeeds to generate a one-dimensional

function σ, which maximizes the number of fully satisfied transitions. In particular, σ fully satisfies all transitions fully satisfied by ρ', otherwise, as showed for Lemma 5, the function $\rho'[0] + \sigma$ fully satisfies more transitions than σ, a contradiction. Thus, if we replace $\rho'[0]$ by σ, we get a new ranking ρ'' of dimension d.

Now, there is a subtlety. For computing the next dimension of the ranking function with the algorithm of Section 3.2.2, we could consider only the pairs $(\vec{x}, \vec{x}') \in Q_t$ such that $\Delta_t(\sigma[0], \vec{x}, \vec{x}') = 0$. Nevertheless, to make the algorithm simpler, we keep the whole Q_t as soon as the transition t is not fully satisfied. We need to show that this strategy does not weaken the algorithm. We use again Lemma 7, considering this time only the transitions t not fully satisfied by $\rho''[0] = \sigma$. Again, for these transitions, the hypotheses of Lemma 7 are fulfilled if we consider the two first dimensions of $\Delta_t(\rho'', \vec{x}, \vec{x}')$. As we did for Lemma 8, there is a (one-dimensional) linear combination δ of $\rho''[0]$ and $\rho''[1]$ such that $\Delta(\delta, \vec{x}, \vec{x}') \geq 0$ and $\Delta(\delta, \vec{x}, \vec{x}') > 0$ if $\Delta(\rho'', \vec{x}, \vec{x}')$ has level ≤ 2. Finally, if we replace the second dimension of ρ'' by δ, Lemma 6 shows that we get a new ranking function ρ''' of dimension d. Furthermore, by construction of δ, the last $d - 1$ dimensions of ρ''' satisfy all transitions not fully satisfied by its first dimension, i.e., by σ, the first dimension generated by the algorithm of Section 3.2.2.

We can now iterate the process similarly until all transitions are satisfied, which shows that the algorithm of Section 3.2.2 generates a ranking of dimension at most d. This is true for any d, dimension of an affine ranking function, thus the ranking function generated by the algorithm of Section 3.2.2 has minimal dimension. ■

To summarize the proof, we start from an affine ranking of dimension d. We show that there is an affine ranking of dimension d that fully satisfies at least one transition. This proves that our algorithm does not stop and generates a one-dimensional function σ. Then, we show that there is an affine function of dimension d whose first dimension is σ. Finally, we show that there is an affine function of dimension d, whose first dimension is σ, and such that the $d - 1$ last dimensions satisfy all transitions not fully satisfied by σ. Iterating the process, this shows our algorithm terminates and generates an affine ranking of dimension $\leq d$, for any possible dimension d.

Remark 3 The proof of Theorem 1 is similar to the proof given in [40] of the optimality of the multi-dimensional scheduling algorithm of Feautrier [20]. The main difference is that we handle the general case of transitions where $a(\vec{x})$ is a set not just a singleton, i.e., we do not assume that the actions are functions. Also, as far as we could check, the initialization of the induction in Theorem 1, i.e., the fact that the algorithm fully satisfies at least one transition if a multi-dimensional affine function exists, seems to be missing in [40]. In our proof, this property is given by Lemma 8.

3.4 Scalability

The size of the system (8) is roughly proportional to the number of transitions in the automaton. This may be too much for the underlying linear solver. However, the situation can be improved along the lines of [21]. Notice first that each transition has its own set of Farkas multipliers, the $\vec{\mu}_t$ of (5). These multipliers can be eliminated one transition at a time, thus leaving only the $\vec{\lambda}_k$ as unknowns. Furthermore, the unknowns in the constraint system for a transition pertain only to the source and sink of the transition. Hence, the grand constraint system (8) is a block representation of the connexion graph of the automaton: each system (5) generates a block of rows in which only the columns corresponding to the unknowns for the source and sink of transition t

are nonzero. These unknowns can be successively eliminated using various heuristics (e.g., eliminate first the unknowns for a state of minimum degree), leaving only a system of constraints on the ε_t, which is then solved for the maximum σ. The different $\vec{\lambda}$ are then recovered by a process of back substitution, and there is no need to compute the $\vec{\mu}_t$.

In these algorithms, elimination (or projection) turned out to be best achieved by a combination of Gaussian elimination, controlled Fourier-Motzkin elimination, and Chernikova algorithm.

3.5 Worst-case time complexity

As shown in the survey by Wilhelm et al. [41], the computation of a worst-case execution time (WCET) is a highly complex affair, as it has to take into account the program, its data, and the processor on which it is run. Handling all these complexities is beyond the scope of our paper. Our aim is to evaluate an *abstract* WCET, as would be observed on a processor with a perfectly additive timing model, executing one automaton transition in unit time. We call this quantity the *worst time complexity* of the program (WTC). Such an estimate can be useful, for example as a template with unknown coefficients, to be fitted to actual measurements by a process of regression. It is also standard in high-level synthesis to need an upper-bound on the number of loop iterations (DO loops as well as WHILE loops), to enable scheduling optimization at higher level. We thus define the WTC as an upper bound on the number of transitions executed, given an initial value of the counter variables.

With this definition, one could over-approximate WTC by the total number of reachable states, i.e., WTC $\leq \sum_k \#\mathcal{R}_k$ or even more conservative WTC $\leq \sum_k \#\mathcal{P}_k$ as $\mathcal{R}_k$ is itself over-approximated by $\mathcal{P}_k$. In addition to the fact that this is a very rough over-approximation, this technique can even lead to an infinite WTC, even for a terminating automaton, if some $\mathcal{P}_k$ is unbounded. Rather, we use the ranking function itself to select the useful bounded constraints. Indeed, consider a trace $(k_0, \vec{x}_0)$, $(k_1, \vec{x}_1)$, …, $(k_p, \vec{x}_p)$ in the execution of the automaton. Since by definition of a ranking function, $\rho(k_{i+1}, \vec{x}_{i+1}) \prec \rho(k_i, \vec{x}_i)$, and since $\prec$ is a strict order, it follows by transitivity that all $\rho(k_i, \vec{x}_i)$ are distinct in $\mathcal{W}$. Hence, the length of the trace is bounded by the cardinal of the co-domain of ρ:

$$\text{WTC} \leq \#\bigcup_k \rho(k, \mathcal{P}_k) \leq \sum_k \#\rho(k, \mathcal{P}_k) \tag{9}$$

The first inequality is more accurate but harder to compute as it involves a union of sets. Counting integer points means considering $\mathbb{Z}$-polyhedra, i.e., intersections of an integer lattice (e.g., $\mathbb{Z}^n$) and a polyhedron (e.g., $\mathcal{P}_k$). As $\vec{x} \mapsto \rho(k, \vec{x})$ is affine, the number of integer points in $\rho(k, \mathcal{P}_k)$ can be over-approximated as the cardinal of some $\mathbb{Z}$-polyhedron (e.g., using Smith form) or computed exactly as the number of points in the image of a $\mathbb{Z}$-polyhedron by an affine function. Such problems have been deeply studied in the context of high performance computing, using various techniques related to Ehrhart polynomials [10, 11, 38, 39]. To deal with unions, it is also possible to use tools that generate loops from polyhedra, such as Cloog [5], which implements a modified version of [36]. Then, counting points directly in loops is also an option as done in [31].

[5] `http://www.cloog.org`

We point out that, in the present version, the WTC is computed according to the rightmost expression in (9). Implementing a more precise version, with unions, is left for future work.

Example 2 (Cont'd) In this very simple, fully-deterministic, example, the two inequalities of (9) lead to the same upper bound WTC $\leq 1 + (N_0 + 1) + N_0(N_0 + 1) = 1 + (N_0 + 1)^2$. □

4 Experimental results

We have built a tool suite that converts a C program into an integer interpreted automaton, constructs its invariants, tests its termination and, if successful, computes an upper bound for its WTC.

Implementation The first tool, `c2fsm` is just an exercise in parsing and control graph construction. The main difficulties come from the complexity of C syntax and semantics, and mainly from the convention that an assignment can occur at any depth in an expression. Our guidelines have been to consider only assignments to integer variables, and to give a variable the bottom value unless it is assigned an affine form in other integer variables. The fact that C has no Booleans is both a simplification and a hindrance, as it forbids the use of some of the techniques that were developed for synchronous languages compilation [5]. The result of this analysis is presented as a system of states and transitions in the input format of Aspic which is responsible for the computation of invariants. The reader is referred to [23] or [17] for a full description of Aspic.

The ranking algorithm and the worst time complexity are implemented in a tool called Rank. The minor premise of Farkas lemma (the fact that the polyhedron is non-empty) and the system 8 are solved with the PIP tool (Parametric Integer Programming), now wrapped in the Piplib library [6]. The Ehrhart polynomials part of the Polylib library is then used for computing time complexities.

Preliminary results This tool chain has been tested on a large set of benchmarks from the literature. Most of the examples were collected in [9] from many other papers dealing with termination analysis. They can be found at the following web address:

`http://www.dcs.qmul.ac.uk/~aziem/esop/`.

Furthermore, the source code for all the examples (including ours) are given in the appendix, following the order of the results. The results are summarized in Table 1. The test cases we developed for checking our algorithm are marked by ♣. first two columns give information about the test cases. The third column gives statistics about the (generated) interpreted automaton, in the following way: number of relevant variables, number of control points, number of transitions. The fourth column gives either the dimension of the ranking function if the algorithm succeeds, or DK ("don't know") if it fails to prove termination. The timing measurements (last column) have been done on a 2 GHz Pentium with 1 GByte of memory running Debian 2.6. These measures include the invariants computation time from the Aspic file, the computation of the ranking function, and the evaluation of the WTC.

[6] `piplib.org`

Name	Ref	Variables	Control points	Transitions	dim ρ	Time (s)
easy1	[9]	3	4	5	1	0.05
easy2	[9]	3	3	3	1	0.05
ackermann	[3]	2	7	7	1	0.07
terminate	[12]	3	1	1	1	0.05
gcd	[6]	2	5	1	1	0.07
rsd	♣	3	3	4	1	0.06
nd-loop	♣	2	5	5	1	0.05
wcet2	♣	2	3	5	1	0.11
relation1	♣	2	4	4	1	0.06
ndecr	♣	2	4	4	1	0.05
cousot11	[16]	3	4	5	1	0.07
cousot16	[16]	2	3	4	1	0.05
random2d	[9]	5	10	21	1	2.70
random1d	[9]	3	4	6	2	0.08
wise	♣	2	6	10	2	0.20
wcet0	♣	3	6	8	2	0.05
wcet1	♣	3	6	8	2	0.20
decr	♣	2	4	5	2	0.06
exmini	♣	4	3	6	2	0.08
aaron2	[9]	3	6	10	2	0.13
while2	1	3	3	4	2	0.06
cousot9	[16]	3	4	5	2	0.08
perfect	[8]	4	7	10	2	0.23
loops	[34]	3	4	5	2	0.07
insertsort	♣	3	6	7	2	0.07
bubblesort	[9]	4	10	17	3	0.38
ax	♣	4	3	6	3	0.07
nestedLoop	[26]	6	12	17	3	18
determinant	[8]	4	6	7	4	0.15
maccarthy91	[13]	4	6	7	DK	0.15

Table 1: Experimental results

Comments In most cases, we were able to prove termination, even for nondeterministic examples like `random2d` in Table 1. Nested loops are correctly handled, and we find multi-dimensional rankings for them. The case of recursion is often handled by making assumptions about (the values of) the variables after a recursive call (for instance we assume that the result of `ackermann` is always positive). We were also able to prove the termination of some sortings algorithms like `bubblesort` and `insertsort`. The ranking we discover for `bubblesort` may seem of the wrong dimension, but the additional dimensions have constant values and the order of magnitude of the WTC is still $O(size^2)$ as expected. For the `nestedLoop` example, which has a very complicated iteration domain, most of the time (16s) is spent in computing the WTC.

However, we are for the moment unable to prove the termination of `mergesort` due to scalability reasons. Note also that we did not try yet to eliminate redundant variables or constraints in the different algorithms. We therefore manipulate polyhedra of higher dimensions than needed. In addition, since the termination of concurrent programs sometimes depends on a fairness hypothesis, we were unable to solve some of the examples of [34]. We found the precision of our algorithm to be strongly dependent on the quality of the invariants, and also the quality of the affine approximation of some (non affine) affectations in the C programs. Moreover, in many cases we had to add some preconditions on the values of initial variables. It seems that most of these preconditions were not mentioned in the literature. However, some of them can be precomputed with a forward-backward analysis, like in [16].

5 Related work

Using ranking functions to prove correctness was first proposed in [22]. Early approaches were semi-automatic: one had to guess ranking functions, and then prove their correctness using some form of Hoare logic. Attempts to automate this process followed [12, 13, 7]. It was then realized that one-dimensional rankings were not powerful enough, and propositions to build multi-dimensional [6] or polynomial rankings [16] followed. We believe that our method (which was suggested by our previous work on scheduling [20]), is a satisfactory solution to this problem.

However, the applicability of our method to large programs is still to be ascertained. In [32], Manna suggested to select a subset of the states of the automaton (cutpoints) in such a way that each cycle in the graph crosses at least one cut point. Termination follows if one can find a ranking function, defined only on the cut points, and which decreases on each path from a cut point to another cut point. It is easy to extend such a function to all states (see Lemma 2) but the resulting global ranking will use the max operator, and may therefore be just piecewise affine rather than affine. It is clear that this method is a way of reducing the program automaton by removing states and merging transition by relation composition. This can be done before the computation of invariants, or after application of Farkas lemma, as suggested in Section 3.4. Assessing the relative merits of these two approaches is left for future work.

A large body of research followed the introduction of the size change termination (SCT) principle in [30]. The difference in the two approaches are mainly in semantics: the automaton represents a call graph instead of a control graph, and the variables may be summary information about data structures, like the length of a list or the size of a tree. More importantly, the relations between input and output variables of a transition are restricted to one of the two forms $x' < y$ and $x' \leq y$. An attempt to lift this

restriction can be found in [1]. In [2], it is shown that termination of an SCT system can be proved using an exponential number of very simple local ranking functions, or with a global ranking function involving an exponential number of subterms. This is in contrast to a very simple consequence of Theorem 1, that the dimension of our ranking function is no larger than the number of transitions (and even no larger than the number of variables with some stronger hypotheses [19]). The explanation is probably that the two sets of programs for which our algorithm succeeds and for which the SCT formalism succeeds are almost disjoint.

Another trend of research has been started in [35] and pursued in [9]. Here, one uses several (local) ranking relations, all of them well founded, the intuition being that each relation proves termination of a part of the program. A consistency condition is necessary: the transitive closure of the transition relation of the program must be included in the union of all local ranking relations. The problem is how to find the local rankings, and how to prove the consistency condition. It may be that we can help at least for the first problem: apply our algorithm to cleverly chosen subsets of the automaton states, as for example strongly connected components or loops.

6 Conclusion

6.1 Contributions

The main contribution of this paper is the design of an algorithm for the construction of multi-dimensional ranking functions, which, in contrast to to the combinatorial algorithm of [6], is greedy but nevertheless complete (with respect to the invariants found and the class of ranking functions considered) and optimal in the dimension of the ranking function. The algorithm makes no assumption whatever on the shape of the source program, and can handle, with proper preprocessing (i.e., after the program is approximated to fit into the affine interpreted automaton model), multiple loops of arbitrary nesting patterns, premature termination and goto's, nondeterministic choices and values, exceptions, and Boolean guards of arbitrary structure. Note also that, in case of failure, our algorithm can also exhibit an execution trace which may not terminate, but all the details are not worked up yet.

The computation of the worst time complexity (WTC) is delegated to a very comprehensive stand-alone algorithm. This means that no arbitrary restrictions about the shape of loops and tests are necessary. We can directly rely on existing methods and tools for counting integer points within $\mathbb{Z}$-polyhedra and images of $\mathbb{Z}$-polyhedra by affine functions. More generally, our work establishes a strong link with computation models, theoretical results, and tools, developed by the automatic parallelization and high-performance computing community, and which seem to be not so used (or partly re-discovered) in the context of program termination. We believe that this connection can lead to further fruitful developments to face other problems faced by both communities.

6.2 Future work

There is nevertheless room for many improvements. The preprocessor we use for converting a program into an interpreted automaton is somewhat brute force: any construct that is not affine in integer variables is replaced by the bottom value, which is absorbing ($\bot \oplus x = \bot$ for most operators), and which prints as `true` in a guard and as a ques-

tion mark in an action. This can be improved by noticing that some operations, like modulo and integer division, can be linearized by the introduction of fresh variables, or that a bottom value may be constrained: for instance, a square is always non-negative. Also, variables with a finite domain, like Booleans and enums, can be used to refine the states. This may cost a large increase in the size of the automaton but has the direct benefit of extending the class of ranking functions considered, as these do not need to be affine anymore for such "unrolled" variables. Making sure that domains of integer variables are "fat" (to use the terminology of [18]) increases the chance that an affine ranking exists and improves the quality of the WTC produced.

There is always room for improving an invariant constructor like Aspic. One may for instance improve the acceleration algorithms and loops treatment, or use additional abstract interpretation frameworks, like the congruences and lattices of [24]. Also, to prove program termination, it may be interesting to try to combine the construction of the invariants and of the ranking functions to increase the accuracy of the method, thanks to a better control on how widening is performed when computing invariants.

Last but not least, the power of the ranking algorithm can be increased in many ways. For instance, experiments have shown that imposing that ranking functions are nonnegative everywhere (see Inequality 2) is too strong a constraint in many cases. It is enough to impose it at a set of cut points. If the automaton graph becomes acyclic when these cut points are removed, then termination is still guaranteed, notwithstanding the relaxed nonnegativity constraint.

Research on the SCT paradigm has shown that ranking functions of a more complex shape, like piecewise affine functions, are necessary in some cases. In our framework, this means splitting the invariant of some state(s) by an affine constraint. How to choose the states to split and the splitting predicate is left for future research.

There remains the question of interprocedural termination. If there is no recursive call, one may resort to inlining, but this will raise again the question of scalability. One may want to combine the SCT approach with the present one: if there is no infinite path in the call graph, and no infinite path in each function control graph, then termination is guaranteed. However, in many cases, the caller and the callee may interact in complex ways, especially in the presence of side effects, and this will greatly complicate a termination test.

A point we have not investigated is the termination of distributed programs. Our algorithm fails when termination depends on a fairness hypothesis.

References

[1] Hugh Anderson and Siau-Cheng Khoo. Affine-based size-change termination. In Atsushi Ohori, editor, *1st Asian Symposium on Programming Languages and Systems (APLAS'03)*, volume 2895 of *Lecture Notes in Computer Science*, pages 122–140, Beijing, 2003. Springer Verlag.

[2] Amir M. Ben-Amram. A complexity tradeoff in ranking-function termination proofs. *Acta Informatica*, 46(1):57–72, 2009.

[3] Amir M. Ben-Amram and Chin Soon Lee. Program termination analysis in polynomial time. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 29(1):5, 2007.

[4] Andreas Blass and Yuri Gurevich. Inadequacy of computable loop invariants. *ACM Transactions on Computational Logic (TOCL)*, 2(1):1–11, 2001.

[5] A. Bouajjani, J.-C. Fernandez, N. Halbwachs, P. Raymond, and C. Ratel. Minimal state graph generation. *Science of Computer Programming*, 18:247–269, 1992.

[6] Aaron A. Bradley, Zohar Manna, and Henny B. Sipma. The polyranking principle. In *32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *Lecture Notes in Computer Science*, pages 1349–1361. Springer Verlag, July 2005.

[7] Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. Linear ranking with reachability. In Kousha Etessami and Sriram K. Rajamani, editors, *17th International Conference on Computer Aided Verification (CAV)*, volume 3576 of *Lecture Notes in Computer Science*, pages 491–504. Springer Verlag, July 2005.

[8] Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. Termination analysis of integer linear loops. In *16th International Conference on Concurrency Theory (CONCUR)*, volume 3653 of *Lecture Notes in Computer Science*, pages 488–502. Springer Verlag, August 2005.

[9] Aziem Chawdhary, Byron Cook, Sumit Gulwani, Mooly Sagiv, and Hongseok Yang. Ranking abstractions. In *17th European Symposium on Programming (ESOP'08)*, volume 4960 of *Lecture Notes in Computer Science*, pages 81–92, Budapest, April 2008. Springer Verlag.

[10] Philippe Clauss. Counting solutions to linear and nonlinear constraints through Ehrhart polynomials: Applications to analyze and transform scientific programs. In *International Conference on Supercomputing (ICS'96)*, pages 278–285. ACM, 1996.

[11] Philippe Clauss. Handling memory cache policy with integer points counting. In *Parallel Processing, 3rd International Euro-Par Conference*, volume 1300 of *Lecture Notes in Computer Science*, pages 285–293, Passau, August 1997. Springer Verlag.

[12] Michael Colón and Henny Sipma. Synthesis of linear ranking functions. In *7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01)*, volume 2031 of *Lecture Notes in Computer Science*, pages 67–81. Springer Verlag, 2001.

[13] Michael A. Colón and Henny B. Sipma. Practical methods for proving program termination. In *14th International Conference on Computer Aided Verification (CAV)*, volume 2404 of *Lecture Notes in Computer Science*, pages 442–454. Springer Verlag, January 2002.

[14] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th ACM Symposium on Principles of Programming Languages (POPL'77)*, pages 238–252, Los Angeles, January 1977.

[15] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *5th ACM Symposium on Principles of Programming Languages (POPL'78)*, pages 84–96, Tucson, January 1978.

[16] Patrick Cousot. Proving program invariance and termination by parametric abstraction, Lagrangian relaxation, and semidefinite programming. In *6th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'05)*, volume 3385 of *Lecture Notes in Computer Science*, pages 1–24, Paris, January 2005. Springer Verlag.

[17] Laure Danthony-Gonnord. *Accélération abstraite pour l'amélioration de la précision en Analyse des Relations Linéaires*. PhD thesis, Université Joseph Fourier, Grenoble, October 2007.

[18] Alain Darte, Leonid Khachiyan, and Yves Robert. Linear scheduling is nearly optimal. *Parallel Processing Letters*, 1(2):73–81, 1991.

[19] Alain Darte, Yves Robert, and Frédéric Vivien. *Scheduling and Automatic Parallelization*. Birkhauser, 2000. ISBN 0-8176-4149-1.

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

[21] Paul Feautrier. Scalable and structured scheduling. *International Journal of Parallel Programming*, 34(5):459–487, May 2006.

[22] Robert W. Floyd. Assigning meaning to programs. In J. T. Schwartz, editor, *Symposium on Applied Mathematics*, volume 19, pages 19–32. A.M.S., 1967.

[23] L. Gonnord and N. Halbwachs. Combining widening and acceleration in linear relation analysis. In *13th International Static Analysis Symposium (SAS'06)*, volume 4134 of *Lecture Notes in Computer Science*, pages 144–160, Seoul, August 2006. Springer Verlag.

[24] Philippe Granger. Static analysis of linear congruence equalities among variables of a program. In *International Joint Conference on Theory and Practice of Software Development (TAPSOFT'91)*, volume 493 of *Lecture Notes in Computer Science*, pages 169–192, Brighton, 1991. Springer Verlag.

[25] Sumit Gulvani, Krishna K. Mehra, and Trishul Chilimbi. SPEED: Precise and efficient static estimation of program computational complexity. In *36th ACM Symposium on Principles of Programming Languages (POPL'09)*, pages 127–139, Savannah, January 2009.

[26] Sumit Gulwani, Sagar Jain, and Eric Koskinen. Control-flow refinement and progress invariants for bound analysis. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'09)*, pages 375–385, Dublin, 2009. ACM.

[27] N. Halbwachs. Détermination automatique de relations linéaires vérifiées par les variables d'un programme. PhD thesis, Université de Grenoble, March 1979.

[28] R. M. Karp, R. E. Miller, and S. Winograd. The organization of computations for uniform recurrence equations. *Journal of the ACM*, 14(3):563–590, July 1967.

[29] C. S. Lee. Ranking functions for size-change termination. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 31(3):1–42, 2009.

[30] Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. The size-change principle for program termination. *ACM SIGPLAN Notices*, 36(3):81–92, 2001.

[31] P. Lokuciejewski, D. Cordes, H. Falk, and P. Marwedel. A fast and precise static loop analysis based on abstract interpretation, program slicing, and polytope models. In *International Symposium on Code Generation and Optimization (CGO'09)*, pages 136–146, Seattle, March 2009. IEEE Computer Society Press.

[32] Zohar Manna. Termination of programs represented as interpreted graphs. In *Spring Joint Computer Conference (AFIPS'70)*, pages 83–89. ACM, May 1970.

[33] Zohar Manna. *Mathematical Theory of Computing*. MacGraw-Hill, 1974.

[34] Andreas Podelski and Andrey Rybalchenko. A complete method for the synthesis of linear ranking functions. In Bernhard Steffen and Giorgio Levi, editors, *Verification, Model Checking, and Abstract Interpretation (VMCAI'03)*, volume 2937 of *Lecture Notes in Computer Science*, pages 239–251. Springer Verlag, 2004.

[35] Andreas Podelski and Andrey Rybalchenko. Transition invariants. In Harald Ganzinger, editor, *IEEE Symposium on Logic in Computer Science (LICS'04)*, pages 32–41. IEEE Computer Society, July 2004.

[36] Fabien Quilleré, Sanjay V. Rajopadhye, and Doran Wilde. Generation of efficient nested loops from polyhedra. *International Journal of Parallel Programming*, 28(5):469–498, 2000.

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

[38] Sven Verdoolaege, Kristof Beyls, Maurice Bruynooghe, and Francky Catthoor. Experiences with enumeration of integer projections of parametric polytopes. In *14th International Conference on Compiler Construction (CC'05)*, volume 3443 of *Lecture Notes in Computer Science*, pages 91–105, Edinburgh, 2005. Springer Verlag.

[39] Sven Verdoolaege, Rachid Seghir, Kristof Beyls, Vincent Loechner, and Maurice Bruynooghe. Counting integer points in parametric polytopes using Barvinok's rational functions. *Algorithmica*, 48(1):37–66, 2007.

[40] Frédéric Vivien. On the optimality of Feautrier's scheduling algorithm. *Concurrency and Computation: Practice and Experience*, 15(11-12):1047–1068, September 2003. Euro-Par'02 special issue.

[41] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The determination of worst-case execution times—overview of the methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):1–53, 2008.

A Source code of kernels

This section provides the source code of the kernels used for the experimental results. We also recall the execution times given in table 1. The timings were obtained on a 2GHz Pentium with 1GByte of memory running on a Debian 2.6.

easy1 (0.05s)

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

easy2 (0.05s)

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

ackermann (0.07s)

```
//Automaton entered by hand
//Assuming m>=0 && n>=0
A:
  if (m<=0) //A(0,n) = n+1
    return;
  if (n<=0) { //A(m,0) = A(m-1,1)
    m = m - 1; n = 1; goto A;
  }
  //A(m,n) = A(m-1,A(m,n-1))
  m = m - 1;
  n = random(); //n = A(m,n-1);
  goto A;
```

terminate (0.05s)

```
int i, j, k, ell;
while (i <= 100 && j <= k)
{
  ell = i;
  i = j;
  j = ell + 1;
  k--;
}
```

gcd (0.07s)

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

rsd (0.06s)

```
int r,da,db,temp;
if (r>=0){
  da = 2*r;
  db = 2*r;
  while (da >= r) {
    if (brandom()){
      da --;
    }
    else{
      temp = da;
      da = db - 1;
      db = temp;
    }
  }
}
```

nd-loop (0.05s)

```
int x,y;
x=0;
do {
  y=x;
  x=random();
  if ((x-y>2) || (x-y<1))
    break;
}
while (x<10);
```

wcet2 (0.11s)

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

relation1 (0.06s)

```
int x,y;
do {
  y=x;
  x=random();
  if ((x-y>2) || (x-y<1)) break;
}
while (x<10);
```

ndecr (0.05s)

```
int i,n;
i = n-1;
while (i>1)
{
  i--;
}
```

cousot11 (0.07s)

```
int x,y,i;
while (x<y)
{
  if (i >= 0)
    x = x + i + 1;
  else
    y = y + i;
}
```

cousot16 (0.05s)

```
int i,j;
i = 2;
j = 0;
while (random())
{
  if(j >= 0 && i >= 2*j+2)
    if (brandom()) i = i + 4;
    else {
      i = i + 2;
      j = j + 1;
    }
}
```

random2d (2.70s)
```
int x,y,i,r,N;
x = 0;
y = 0;
i = 0;
while (i<N) {
  i=i+1;
  r=random();
  if (r>=0 && r<=3) {
    if (r==0) x=x+1;
    else if (r==1) x=x-1;
    else if (r==2) y=y+1;
    else if (r==3) y=y-1;
  }
}
```

random1d (0.08s)
```
int a,x,max;
if (max > 0) {
  a = 0;
  x = 1;
  while (x<=max) {
    if (brandom())
      a = a + 1;
    else
      a = a - 1;
    x = x + 1;
  }
}
```

wise (0.20s)
```
int x, y;
if (x<0 || y<0) return;
while (x-y>2 || y-x>2)
{
  if (x < y)
    ++x;
  else
    ++y;
}
```

wcet0 (0.05s)
```
int i,j,n;
j = 0;
i = n;
if (n>=1)
{
  do {
    if (brandom()) {
      j++;
      if (j>=n) j = 0;
    }
    else {
      j--;
      if (j<=-n) j = 0;
    }
    i--;
  }
  while (i>0);
}
```

wcet1 (0.20s)
```
int i,j,n;
j = 0;
i = n;
if (n>=1)
{
  do {
    if (brandom()){
      j++;
      if (j>=n) j = 0;
    }
    else {
      j--;
      if (j<=0) j = 0;
    }
    i--;
  }
  while (i>0);
}
```

decr (0.06s)
```
int i,n;
if (n>=1)
{
  i = n-1;
  while (i>=0)
    {
      if (i==0 && random())
        break;
      i--;
    }
}
```

exmini (0.08s)
```
int i,j,k,tmp;
while (i<=100 && j<=k){
  tmp = i;
  i = j;
  j = tmp + 1;
  k = k - 1;
}
```

aaron2 (0.13s)
```
int tx, x, y;
if (tx >= 0) {
  while (x >= y) {
    if (tx < 0) return 0;
    if (brandom())
      x = x - 1 - tx;
    else
      y = y + 1 + tx;
  }
}
```

while2 (0.06s)
```
int i,j,N;
i = N;
if (i > 0) {
  j = N;
  while (j > 0) j--;
}
```

cousot9 (0.08s)
```
int i,j,N;
i = N;
while (i>0) {
  if (j>0) j--;
  else
    {
      j = N;
      i--;
    }
}
```

perfect (0.23s)
```
int y1, y2, y3;
if (x <= 1) return 0;
y1 = x;
y2 = x;
y3 = x;
for(y1=x-1; y1>0; y1=y1-1)
{
  while (y2 >= y1)
    y2 = y2 - y1;
  if (y2 == 0)
    y3 = y3 - y1;
  y2 = x;
}
//return (y3 == 0);
```

loops (0.07s)
```
int n; /* n > 0 */
int x, y;
x = n;
if (x >= 0)
{
  while (x >= 0){
    y = 1;
    if (y < x)
      while (y < x)
        y = 2*y;
    x = x - 1;
  }
}
```

insertsort (0.07s)

```
int a[];
int len;
int i, j, value;
for (i=1; i<len; i++)
{
  value = a[i];
  for (j=i-1;
       j>=0 && a[j]>value;
       j--)
  {
    a[j + 1] = a[j];
  }
  a[j+1] = value;
}
```

bubblesort (0.38s)

```
int size,error,b,j,t;
error=0;
b=size;
if (size>0) {
  while (b>=1) {
    if (size<=0) return 0;
    j=1;
    t=0;
    while (j<=b-1) {
      if (j<1||j>size)
        return;
      if (j+1<1||j+1>size)
        return ;
      if (brandom())
        t = j;
      j = j + 1;
    }
    if (t<1 || t>=b)
      return;
    b=t;
  }
}
```

ax (0.07s)

```
int i,j,n;
if (n>=1)
{
  i = 0;
  do {
    j = 0;
    while (j<n-1) j++;
    i++;
  }
  while (j>=n-1 && i<n-1);
}
```

nestedLoop (18s)

```
int i, j, k;
if (0<=n && 0<=m && 0<=N)
{
  i = 0;
  while (i<n){
    j = 0;
    while (j<m){
      j += 1;
      k = i;
      while (k<N)
        k += 1;
      i = k;
    }
    ++i;
  }
}
```

determinant (0.15s)

```
//Automaton entered by hand
//from [33], p. 199
int i,j,k,n,y,z;
int X[10][10];
y = random();
//y = X[1][1];
k = 1;
B1:
if(k == n)
{
  z = y;
  return;
}
else
{
  i = k+1;
 B2:
  if(i == n+1)
    {
      k = k + 1;
      y = random();
      //y = y*X[k][k];
      goto B1;
    }
  else
    {
      j = n;
      while(j != k)
      {
//X[i][j] = X[i][j] -
//  X[k][j]*X[i][k]/X[k][k];
        j = j - 1;
      }
      i = i + 1;
      goto B2;
    }
}
```

maccarthy91 (DK,0.15s)

```
int  y1,y2,z;
y1 = x;
y2 = 1;
if (y1>100) z = y1 - 10;
else
{
  while (y1 <= 100)
  {
    y1 = y1 + 11;
    y2 = y2 + 1;
  }
  while (y2 > 1)
  {
    y1 = y1 - 10;
    y2 = y2 - 1;
    if (y1 > 100 && y2 == 1)
      z = y1 - 10;
    else
    {
      if (y1 > 100)
      {
        y1 = y1 - 10;
        y2 = y2 - 1;
      }
      y1 = y1 + 11;
      y2 = y2 + 1;
    }
  }
}
```

Contents

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399