Ranking Functions for Automatic Program Termination Analysis

Roberto BAGNARA, David MERCHAT, Fred MESNARD,
Andrea PESCETTI, Alessandro ZACCAGNINI,
Enea ZAFFANELLA

University of Parma, Italy
University of La Réunion, France

The Problem

- Does a given program (part) terminate for all possible inputs?
- Answering this question is essential to turn assertions of partial correctness into assertions of total correctness.
- The property of termination of a program fragment is not less important than properties concerning the absence of run-time errors.
  - For instance, critical reactive systems (such as fly-by-wire avionics systems) must maintain a continuous interaction with the environment: failure to terminate of some program components can stop the interaction the same way as if an unexpected, unrecoverable run-time error occurred.
  - Another example: all versions of Windows are plagued by device drivers that hang: Microsoft has set up the “TERMINATOR” research project to address this situation.

THE PROBLEM (CONT.)

- Does a given program (part) terminate for all possible inputs?
- The problem is notoriously undecidable (*Halting Problem*).
- But for a subset of programs we can prove termination by synthesizing ranking functions:
  - Functions that “decrease” after each iteration...
  - ...and are “bounded” from below.
- If a ranking function exists, the program terminates
- ...and vice versa.
- Advantage: the program termination problem is approximated by a mathematical problem that may be algorithmically tractable.
- Typically, a class of functions (e.g., linear functions) is selected and the synthesis of ranking functions is formulated as a search problem in that class (search space).
- Notice that ranking functions may exist outside the selected class!

REMINDER: WELL-FOUNDED RELATIONS

A well-founded relation is a binary relation '$\prec$' over a set S such that there exists no infinite "descending chain", i.e., a sequence of elements of S such that

$$a_0 \succ a_1 \succ \cdots \succ a_i \succ \cdots$$

- a_{i+1} is a predecessor of a_i, which, in turn, is a successor of a_{i+1}.
- $(S, \prec)$ is said to be a well-founded set.
- Equivalently, well-founded relations can be defined as:

  Every non-empty subset of S has an element with no predecessors in S, i.e., for each $U \subseteq S$ such that $U \neq \varnothing$, there exists $v \in U$ such that $u \not\prec v$ for each $u \in U \setminus \{v\}$.

Well-founded Relations: Examples and Non-examples

- $(\mathbb{N}, <)$ is well-founded.

  There exists no infinite descending chain on $\mathbb{N}$.

- $(\mathbb{Z}, <)$ is not well-founded.

  An infinite descending chain is $-1, -2, -3, \ldots$

- $(\mathbb{Z}, \prec)$ where

  $$a \prec b \iff |a| < |b|$$

  is well-founded.

- $(\mathbb{R}, <)$ is not well-founded.

Well-founded Relations: More Examples

- $(\mathbb{R}_+, <)$, where $\mathbb{R}_+ = \{\, x \in \mathbb{R} \mid x \geq 0 \,\}$, is not well founded.
- $(\mathbb{R}_+, <_\epsilon)$ where $\epsilon > 0$ and

  $$a <_\epsilon b \iff a + \epsilon \leq b$$

  is well-founded.

Simple Loops

Consider an individual loop of the form

$$\{ I \} \text{ **while** } B \text{ **do** } C$$

where

- ➜ I is a loop invariant that a previous analysis phase has determined to hold just before any evaluation of B;
- ➜ B is a (side-effect-free) boolean guard expressing the condition on the state upon which iteration continues;
- ➜ C is a command that, *in the context of that loop*, is known (perhaps thanks to a previous analysis) to always terminate.

Capturing the Effects of Invariant, Guard and Updates

Consider again the simple loop

$$\{ I \}\ \textbf{while}\ B\ \textbf{do}\ C$$

- Let $\boldsymbol{x} = (x_1, \ldots, x_n)$ be the tuple of variables occurring in I and/or B and/or C.
- The command C may update some of them, or perhaps all of them.
- We focus on one single execution of C within the loop:
  - let us overload $\boldsymbol{x} = (x_1, \ldots, x_n)$ to represent also the values of the program variables before the execution of C;
  - let the new variables $\boldsymbol{x}' = (x'_1, \ldots, x'_n)$ represent the values of the (unprimed) variables after the execution of C;
  - of course, $\boldsymbol{x}'$ depends on $\boldsymbol{x}$, I, B and C;
- We suppose we are given a formula $c[\boldsymbol{x}, \boldsymbol{x}']$ in some first-order language that correctly approximates the above dependencies for each iteration of the loop.

Ranking Functions

- Consider the simple loop of the previous slides, and assume its variables take values in $\mathbb{D}$.
- Consider also a well-founded set $(\mathbb{S}, \prec)$.
- A ranking function for the loop is a function $f \colon \mathbb{D}^n \to \mathbb{S}$ such that

$$\forall \boldsymbol{x}, \boldsymbol{x}' \in \mathbb{D}^n : c[\boldsymbol{x}, \boldsymbol{x}'] \implies f(\boldsymbol{x}') \prec f(\boldsymbol{x}).$$

- The mere existence of a ranking function ensures termination:
  - its existence implies that any iteration of the loop corresponds to a strict decrease in the well-founded ordering $(\mathbb{S}, \prec)$;
  - as this strict decrease cannot go on forever, the loop cannot go on forever.
- In other words, for the purpose of termination analysis, existence of a ranking function is all we need.
- However, exhibition of one or more ranking functions is interesting for other reasons (e.g., witnessing termination).

EXAMPLE: SIMPLE WHILE LOOP

➜ Consider the following program fragment, where variables are $\mathbb{Z}$-valued:

```
x := input integer;
y := 0;
while x >= 2 do begin
  x := x div 2;
  y := y + 1
end
```

➜ Standard analysis techniques can obtain the following simple loop:

```
{ y >= 0 }
while x >= 2 do begin
  x := x div 2;
  y := y + 1
end
```

➜ Here $I = (y \geq 0)$, $B = (x \geq 2)$ and C is `x := x div 2; y := y + 1`.

EXAMPLE: CAPTURING THE EFFECTS...

➜ Consider again the simple loop

```
{ y >= 0 }
while x >= 2 do begin
  x := x div 2;
  y := y + 1
end
```

➜ There are program analysis techniques that are able to derive, in a completely automatic way, that

$$c[x, y, x', y'] = (x \geq 2 \wedge y \geq 0 \wedge 0 \leq x - 2x' \leq 1 \wedge y' = y + 1)$$

correctly describes the values of `x` and `y` before and after each execution of `x := x div 2; y := y + 1` within the loop.

➜ In fact $x \geq 2$ (given by B), $y \geq 0$ (given by I), $0 \leq x - 2x' \leq 1$ ($x \geq 2$ is divided by 2 in C), and $y' = y + 1$ (y is incremented by 1 in C).

Example: Ranking Functions (Manual Synthesis)

- Recall the condition

$$c[x, y, x', y'] = (x \geq 2 \land y \geq 0 \land 0 \leq x - 2x' \leq 1 \land y' = y + 1)$$

- Consider the well-founded set $(\mathbb{R}_+, <_1)$ where, for each $a, b \in \mathbb{R}_+$,

$$a <_1 b \iff a + 1 \leq b.$$

- Let also $f\colon \mathbb{Z}^2 \to \mathbb{R}$ be defined for each $n, m \in \mathbb{Z}$ by $f(n, m) = n$.
- It is easy to show that

$$\forall x, y, x', y' \in \mathbb{Z} : c[x, y, x', y'] \implies f(x', y') <_1 f(x, y).$$

- In fact:
  1. $\forall x, y, x', y' \in \mathbb{Z} : c[x, y, x', y'] \implies f(x', y'), f(x, y) \in \mathbb{R}_+$;
  2. $\forall x, y, x', y' \in \mathbb{Z} : c[x, y, x', y'] \implies f(x', y') + 1 \leq f(x, y)$, since c implies $x' \leq x/2$ and $x \geq 2$, and $x/2 \leq x - 1$ for $x \geq 2$.

Restricting To Linear Constraints and Functions

- In the example:
  - the condition $c[x, y, x', y']$, interpreted over the reals, was a conjunction of linear constraints;
  - the function f, when viewed as $f \colon \mathbb{R}^2 \to \mathbb{R}$, was a linear function.
- In general, the commitment to linearity implies:
  - we lose something: there may be ranking functions that are nonlinear or that can only be proved to be ranking functions by using nonlinear arguments;
  - but we can use analysis techniques based on convex polyhedra to automatically derive the conditions $c[\boldsymbol{x}, \boldsymbol{x}']$;
  - moreover, whatever is the method we use to derive $c[\boldsymbol{x}, \boldsymbol{x}']$, we can use algorithms such as the simplex to prove that ranking functions exist.
- We can also use other concepts from linear algebra, such as eigenvalues and eigenvectors, to reason on termination in a slightly different way.

Reminder: Linear Programming

- Given a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, two vectors $\boldsymbol{b} \in \mathbb{R}^{m}$ and $\boldsymbol{c} \in \mathbb{R}^{n}$, and an n-columns vector $\boldsymbol{x}$ of unknowns, the following is a Linear Programming (LP) problem:

$$\begin{array}{rl} \text{minimize} & \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \end{array}$$

- The feasible region is the polyhedron $F \subseteq \mathbb{R}^{n}$ whose points are all the solutions of the linear constraints $\boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b}$.
- The objective function is the linear expression $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}$.
- The LP problem can be either:
  - unfeasible, if the feasible region is empty;
  - unbounded, if the feasible region is not empty but there is no finite lower bound to the value of the objective function;
  - optimizable, otherwise; in this case, by linearity, the optimum value for the objective function is met at a vertex of the polyhedron F.

Reminder: The Simplex Algorithm

- ➜ The simplex algorithm solves an LP problem in two phases:
  - In the first phase, the algorithm computes a feasible solution by looking for a vertex of the feasible region $F \subseteq \mathbb{R}^n$, if any.
  - If the LP problem is feasible, the second phase optimizes the objective function by moving from vertex to vertex.
- ➜ Since a polyhedron may have a number of vertices which is exponential in the number n of dimensions and m of constraints, the simplex algorithm has a worst case exponential complexity.
- ➜ There are polynomial-time algorithms for the solutions of LP problems, but these heavily rely on non-linear computations that are expensive (and necessarily subject to rounding errors).
- ➜ In contrast, the simplex algorithm is simpler, very efficient in practice (and can be coded using exact arithmetic).

The Duality Theorem for Linear Programming

- Duality Theorem (a classical result of linear programming theory): every linear programming problem can be converted into an equivalent dual problem. If $\boldsymbol{y}$ is an m-columns vector of unknowns,

$$\begin{array}{ll} \text{minimize} & \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \end{array} \quad \longleftrightarrow \quad \begin{array}{ll} \text{maximize} & \boldsymbol{y}^{\mathrm{T}}\boldsymbol{b} \\ \text{subject to} & \boldsymbol{y}^{\mathrm{T}}\boldsymbol{A} = \boldsymbol{c}^{\mathrm{T}} \\ & \boldsymbol{y} \geq \mathbf{0} \end{array}$$

- The dual of the dual is again the primal.
- If both problems have bounded feasible solutions, then both of them have optimal solutions and these solutions have the same value for the corresponding objective functions.
- Several variants of this theorem, corresponding to various types of LP problems, exist.

The Method of Mesnard-Serebrenik: Introduction

➜ The method is applicable when $c[\boldsymbol{x}, \boldsymbol{x}']$ is a conjunction of linear inequalities.

➜ It allows the automatic synthesis of affine (and linear) ranking functions, that is, of the form

$$f(\boldsymbol{x}) = \mu_0 + \boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{x} = \mu_0 + \sum_{i=1}^{n} \mu_i x_i,$$

where $x_1, \ldots\, x_n$ are rational-valued and $\mu_i \in \mathbb{Z}$ for $i = 0, \ldots, n$.

➜ The method is complete in the sense that it decides the following problem:

do $\mu_0, \mu_1, \ldots, \mu_n \in \mathbb{Z}$ and $\epsilon > 0$ exist such that

1. $\forall \boldsymbol{x}, \boldsymbol{x}' \in \mathbb{Q}^n : c[\boldsymbol{x}, \boldsymbol{x}'] \implies f(\boldsymbol{x}') <_\epsilon f(\boldsymbol{x})$,
2. $\forall \boldsymbol{x}, \boldsymbol{x}' \in \mathbb{Q}^n : c[\boldsymbol{x}, \boldsymbol{x}'] \implies f(\boldsymbol{x}'), f(\boldsymbol{x}) \in \mathbb{Q}_+$?

➜ The method is an extension of a method due to Sohn and Van Gelder (1991).

The Method of Mesnard-Serebrenik: Overview

- Suppose for a moment that the problem is not to find values $\mu_0, \ldots, \mu_n$ such that f is a ranking function.
- Suppose that someone gives to us $n+1$ numbers $\mu_0, \ldots, \mu_n$, and that we have to check they are such that f is a ranking function.
- We can do so by solving two linear problems, exploiting the fact that $c[\boldsymbol{x}, \boldsymbol{x}']$ can be expressed as $(\boldsymbol{x}, \boldsymbol{x}')^{\mathrm{T}} \boldsymbol{A}_c \geq \boldsymbol{b}_c^{\mathrm{T}}$, for suitable $\boldsymbol{A}_c$ and $\boldsymbol{b}_c$.
- The first problem expresses that f is ϵ-decreasing for $\epsilon = 1$:

$$\begin{array}{rl} \text{minimize} & (\boldsymbol{x}, \boldsymbol{x}')^{\mathrm{T}} (\boldsymbol{\mu}, -\boldsymbol{\mu}) \\ \text{subject to} & (\boldsymbol{x}, \boldsymbol{x}')^{\mathrm{T}} \boldsymbol{A}_c \geq \boldsymbol{b}_c^{\mathrm{T}} \end{array}$$

  and further verify that the minimum is at least 1.
- If that is the case,

$$f(\boldsymbol{x}) - f(\boldsymbol{x}') = \mu_0 + \boldsymbol{\mu}^{\mathrm{T}} \boldsymbol{x} - \mu_0 - \boldsymbol{\mu}^{\mathrm{T}} \boldsymbol{x}' = (\boldsymbol{x}, \boldsymbol{x}')^{\mathrm{T}} (\boldsymbol{\mu}, -\boldsymbol{\mu}) \geq 1,$$

  so that $f(\boldsymbol{x}') <_1 f(\boldsymbol{x})$.

THE METHOD OF MESNARD-SEREBRENIK: OVERVIEW (CONT.)

- The other linear problem expresses that, subject to $c[\boldsymbol{x}, \boldsymbol{x}']$, f is bounded from below by 0, that is,

$$f(\boldsymbol{x}') = \mu_0 + \boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{x}' \geq 0.$$

- Note that the choice of the numbers 1, for ϵ, and 0, for f's lower bound, is not restrictive.

- So far so good... but we have cheated. No one will give us $\mu_0, \ldots, \mu_n$: we have to find them somehow!

THE METHOD OF MESNARD-SEREBRENIK: OVERVIEW (CONT.)

- Sohn and Van Gelder made the crucial observation that the duality theorem would save the day.
- Consider the problem that corresponds to ensuring ϵ-decrease:

$$\begin{array}{ll} \text{minimize} & (\boldsymbol{x}, \boldsymbol{x}')^{\mathrm{T}}(\boldsymbol{\mu}, -\boldsymbol{\mu}) \\ \text{subject to} & (\boldsymbol{x}, \boldsymbol{x}')^{\mathrm{T}}\boldsymbol{A}_c \geq \boldsymbol{b}_c^{\mathrm{T}} \end{array}$$

- Its dual, obtained with the suitable variant of the Duality Theorem, is

$$\begin{array}{ll} \text{maximize} & \boldsymbol{b}_c^{\mathrm{T}}\boldsymbol{y} \\ \text{subject to} & \boldsymbol{A}_c\boldsymbol{y} = (\boldsymbol{\mu}, -\boldsymbol{\mu}) \\ & \boldsymbol{y} \geq \boldsymbol{0} \end{array}$$

- Here, the unknown parameters $\boldsymbol{\mu}$ occur linearly, whereas in the primal they are multiplied by $\boldsymbol{x}$.

THE METHOD OF MESNARD-SEREBRENIK: OVERVIEW (CONT.)

- Because $\boldsymbol{\mu}$ occurs linearly in

$$\begin{array}{ll} \text{maximize} & \boldsymbol{b}_c^{\mathrm{T}}\boldsymbol{y} \\ \text{subject to} & \boldsymbol{A}_c\boldsymbol{y} = (\boldsymbol{\mu}, -\boldsymbol{\mu}) \\ & \boldsymbol{y} \geq \boldsymbol{0} \end{array}$$

  we can treat the μ's as variables, instead of given constants.
- The boundedness condition, that is $f(\boldsymbol{x}') = \mu_0 + \boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{x}' \geq 0$ subject to $c[\boldsymbol{x}, \boldsymbol{x}']$, is treated in a similar way: a suitable LP problem is set and its dual is considered.

The Method of Mesnard-Serebrenik: Overview (cont.)

➜ Letting $\tilde{\boldsymbol{\mu}} \stackrel{\text{def}}{=} (\boldsymbol{\mu}, \mu_0)$, $\tilde{\boldsymbol{A}}_c \stackrel{\text{def}}{=} \left(\begin{smallmatrix} \boldsymbol{A}_c & 0 & 0 \\ \mathbf{0} & 1 & -1 \end{smallmatrix}\right)$ and $\tilde{\boldsymbol{b}}_c \stackrel{\text{def}}{=} (\boldsymbol{b}_c, 1, -1)$, the method boils down to considering the following LP problems:

$$\begin{array}{ll} \text{maximize} & \boldsymbol{b}_c^{\mathrm{T}} \boldsymbol{y} \\ \text{subject to} & \boldsymbol{A}_c \boldsymbol{y} = (\boldsymbol{\mu}, -\boldsymbol{\mu}) \\ & \boldsymbol{y} \geq \mathbf{0} \\ & \boldsymbol{b}_c^{\mathrm{T}} \boldsymbol{y} \geq 1 \end{array} \qquad \begin{array}{ll} \text{maximize} & \tilde{\boldsymbol{b}}_c^{\mathrm{T}} \boldsymbol{z} \\ \text{subject to} & \tilde{\boldsymbol{A}}_c \boldsymbol{z} = (\mathbf{0}, \tilde{\boldsymbol{\mu}}) \\ & \boldsymbol{z} \geq \mathbf{0} \\ & \tilde{\boldsymbol{b}}_c^{\mathrm{T}} \boldsymbol{z} \geq 0 \end{array}$$

➜ Notice that the requirements on the optima (≥ 1 for the strict decrease and ≥ 0 for boundedness) have been incorporated into the constraints.

➜ Hence the objective functions are now superfluous (they could be replaced by 0) and only satisfiability of the constraints is meaningful.

➜ The two sets of constraints can be merged and a single invocation to the simplex can be used to decide satisfiability.

➜ And satisfiability implies the termination of the original loop.

The Method of Mesnard-Serebrenik: Overview (cont.)

maximize	$\boldsymbol{b}_c^{\mathrm{T}}\boldsymbol{y}$	maximize	$\tilde{\boldsymbol{b}}_c^{\mathrm{T}}\boldsymbol{z}$
subject to	$\boldsymbol{A}_c\boldsymbol{y} = (\boldsymbol{\mu}, -\boldsymbol{\mu})$	subject to	$\tilde{\boldsymbol{A}}_c\boldsymbol{z} = (\boldsymbol{0}, \tilde{\boldsymbol{\mu}})$
	$\boldsymbol{y} \geq \boldsymbol{0}$		$\boldsymbol{z} \geq \boldsymbol{0}$
	$\boldsymbol{b}_c^{\mathrm{T}}\boldsymbol{y} \geq 1$		$\tilde{\boldsymbol{b}}_c^{\mathrm{T}}\boldsymbol{z} \geq 0$

- Alternatively, we can project the first set of constraints on $\boldsymbol{\mu}$:
  - we will obtain the conditions that $\boldsymbol{\mu}$ must satisfy to induce a function f that is 1-decreasing under $c[\boldsymbol{x}, \boldsymbol{x}']$.
- Then, we can project the second set of constraints on $\tilde{\boldsymbol{\mu}}$:
  - we will obtain the conditions that $\tilde{\boldsymbol{\mu}}$ must satisfy to induce a function f that is bounded (actually, nonnegative) under $c[\boldsymbol{x}, \boldsymbol{x}']$.
- Taking the conjunction of these conditions we will obtain a description of all functions f that are *normalized* (i.e., 1-decreasing and nonnegative) ranking functions under $c[\boldsymbol{x}, \boldsymbol{x}']$.

THE METHOD OF MESNARD-SEREBRENIK: OVERVIEW (CONT.)

➜ Notice that obtaining the space of all normalized ranking functions is more expensive (due to the projection operations) than simply testing (using one invocation to the simplex) that at least one ranking function exists.

THE METHOD OF PODELSKI-RYBALCHENKO: INTRODUCTION

- Another complete method for the automatic synthesis of affine (and linear) ranking functions, i.e., of the form

$$f(x_1, \ldots, x_n) = \mu_0 + \sum_{i=1}^{n} \mu_i x_i,$$

  where $x_1, \ldots\ x_n$ are rational-valued and $\mu_i \in \mathbb{Z}$ for $i = 0, \ldots, n$.
- It can be applied, like the method of Mesnard and Serebrenik, when $c[\boldsymbol{x}, \boldsymbol{x}']$ can be expressed as a conjunction of linear inequalities.
- Indeed, it can be proved to be equivalent to the method of Mesnard and Serebrenik: if one of the two methods can prove termination of a given approximated loop $c[\boldsymbol{x}, \boldsymbol{x}']$, or prove that no affine ranking function exists for that loop, then the other method can do the same.

The Method of Podelski-Rybalchenko: Overview

- Condition $c[\boldsymbol{x}, \boldsymbol{x}']$ is rewritten as

$$\left(\boldsymbol{A}\boldsymbol{A}'\right)\left(\begin{smallmatrix}\boldsymbol{x}\\ \boldsymbol{x}'\end{smallmatrix}\right) \leq \boldsymbol{b}$$

- Termination is proved to be equivalent to the existence of two nonnegative rational vectors $\boldsymbol{\lambda}_1$ and $\boldsymbol{\lambda}_2$ satisfying the following four conditions:

$$\begin{aligned} \boldsymbol{\lambda}_1\boldsymbol{A}' &= \boldsymbol{0}, & (\boldsymbol{\lambda}_1 - \boldsymbol{\lambda}_2)\boldsymbol{A} &= \boldsymbol{0}, \\ \boldsymbol{\lambda}_2(\boldsymbol{A} + \boldsymbol{A}') &= \boldsymbol{0}, & \boldsymbol{\lambda}_2\boldsymbol{b} &< 0. \end{aligned}$$

- If two such vectors exist, a ranking function is then defined by

$$f(\boldsymbol{x}) \stackrel{\text{def}}{=} \boldsymbol{\lambda}_2\boldsymbol{A}'\boldsymbol{x}.$$

The Method of Podelski-Rybalchenko: Example

For example, the condition abstracting our example loop,

$$c[x, y, x', y'] = (x \geq 2 \wedge y \geq 0 \wedge 0 \leq x - 2x' \leq 1 \wedge y' = y + 1)$$

can be expressed by

$$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ -1 & 0 & 2 & 0 \\ 1 & 0 & -2 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ x' \\ y' \end{pmatrix} \leq \begin{pmatrix} -2 \\ 0 \\ 0 \\ 1 \\ -1 \\ 1 \end{pmatrix}.$$

The Method of Podelski-Rybalchenko: Example (cont.)

- We look for two nonnegative rational vectors $\boldsymbol{\lambda}_i = (\lambda_{i1}, \lambda_{i2}, \lambda_{i3}, \lambda_{i4}, \lambda_{i5}, \lambda_{i6})$, for $i = 1, 2$, satisfying:

$$\boldsymbol{\lambda}_1 \boldsymbol{A}' = \mathbf{0} \iff \begin{cases} 0 = \lambda_{13} - \lambda_{14} \\ 0 = \lambda_{15} - \lambda_{16} \end{cases}$$

$$(\boldsymbol{\lambda}_1 - \boldsymbol{\lambda}_2)\boldsymbol{A} = \mathbf{0} \iff \begin{cases} 0 = -(\lambda_{11} - \lambda_{21}) - (\lambda_{13} - \lambda_{23}) + (\lambda_{14} - \lambda_{24}) \\ 0 = -(\lambda_{12} - \lambda_{22}) + (\lambda_{15} - \lambda_{25}) - (\lambda_{16} - \lambda_{26}) \end{cases}$$

$$\boldsymbol{\lambda}_2(\boldsymbol{A} + \boldsymbol{A}') = \mathbf{0} \iff \begin{cases} 0 = -\lambda_{21} + \lambda_{23} - \lambda_{24} \\ 0 = -\lambda_{22} \end{cases}$$

$$\boldsymbol{\lambda}_2 \boldsymbol{b} < 0 \iff \begin{cases} -2\lambda_{21} + \lambda_{24} - \lambda_{25} + \lambda_{26} < 0 \end{cases}$$

- A solution is $\boldsymbol{\lambda}_1 = (1, 0, 0, 0, 0, 0)$, $\boldsymbol{\lambda}_2 = (\frac{1}{2}, 0, \frac{1}{2}, 0, 0, 0)$;
- This induces the ranking function $f(x, y) = x$.

Reminder: Eigenvalues and Eigenvectors

➜ Given a square matrix $\mathbf{A}$, we say that a non-zero vector $\boldsymbol{v}$ of the same size is an eigenvector for $\mathbf{A}$ relative to the eigenvalue $\lambda \in \mathbb{C}$ if

$$\mathbf{A}\boldsymbol{v} = \lambda \boldsymbol{v}.$$

➜ Eigenvalues satisfy the characteristic equation $\det(\mathbf{A} - \lambda \mathbf{I}) = 0$, where $\mathbf{I}$ is the identity matrix.

➜ Eigenvectors can be computed from eigenvalues.

➜ Intuitively, eigenvectors with real eigenvalues are directions where $\mathbf{A}$ behaves as the multiplication by a fixed constant; eigenvectors with complex eigenvalues correspond to planes that rotate under the action of $\mathbf{A}$.

➜ In a word, eigenvectors are “directions” where $\mathbf{A}$ behaves in the simplest possible manner.

The Method of Tiwari: Intuition

- Consider the simple loop

$$\{\,\}\ \textbf{while}\ \boldsymbol{b}^{\mathrm{T}}\boldsymbol{x} > 0\ \textbf{do}\ \boldsymbol{x} := \boldsymbol{A}\boldsymbol{x}$$

- If $\lambda \in \mathbb{R}_+ \setminus \{0\}$ is an eigenvalue of $\boldsymbol{A}$ with eigenvector $\boldsymbol{v}$, then $\boldsymbol{A}\boldsymbol{v} = \lambda\boldsymbol{v}$.
- Iterating, we have that, for each $n \in \mathbb{N}$,

$$\boldsymbol{A}^n\boldsymbol{v} = \lambda^n\boldsymbol{v}$$

  and

$$\boldsymbol{b}^{\mathrm{T}}(\boldsymbol{A}^n\boldsymbol{v}) = \boldsymbol{b}^{\mathrm{T}}(\lambda^n\boldsymbol{v}) = \lambda^n(\boldsymbol{b}^{\mathrm{T}}\boldsymbol{v}).$$

- Since $\lambda > 0$, this means that $\boldsymbol{b}^{\mathrm{T}}(\boldsymbol{A}^n\boldsymbol{v})$ has the same sign as $\boldsymbol{b}^{\mathrm{T}}\boldsymbol{v}$.
- In turn, this implies that if $\boldsymbol{b}^{\mathrm{T}}\boldsymbol{v} > 0$ (if the loop is entered), then the loop does not terminate.

The Method of Tiwari: Intuition (cont.)

- We have seen that if the loop

  $$\{\,\}\ \textbf{while}\ \boldsymbol{b}^{\mathrm{T}}\boldsymbol{x} > 0\ \textbf{do}\ x := \boldsymbol{A}\boldsymbol{x}$$

  is such that $\boldsymbol{A}$ has a positive eigenvalue λ admitting an associated eigenvector $\boldsymbol{v}$ such that $\boldsymbol{b}^{\mathrm{T}}\boldsymbol{v} > 0$, then it will not terminate when initiated with $\boldsymbol{x} = \boldsymbol{v}$.
- The interesting thing is that, if the loop does not terminate for at least one initial value of $\boldsymbol{x}$, then there necessarily exist λ and $\boldsymbol{v}$ enjoying the properties above.
- By contraposition, if $\boldsymbol{A}$ and $\boldsymbol{b}$ do not admit such λ and $\boldsymbol{v}$, then the loop does terminate for all inputs.

The Method of Tiwari: Intuition (cont.)

- If the boolean clause contains two or more inequalities, it is not sufficient to check termination on eigenvectors relative to positive eigenvalues: linear combinations of such eigenvectors come into play.
- Let us consider the loop

$$\{\,\}\ \textbf{while}\ (x > 0 \wedge y > 0)\ \textbf{do}\ y := 2y$$

- The corresponding matrix has the eigenvector $\boldsymbol{v}_1 = (1, 0)$ relative to the eigenvalue $\lambda_1 = 1$, and the eigenvector $\boldsymbol{v}_2 = (0, 1)$ relative to the eigenvalue $\lambda_2 = 2$.
- Since neither eigenvector (nor a negative multiple of it) satisfies the clause, the body loop is not even entered if the input is any multiple of an eigenvector.
- The loop does not terminate if the input is $\boldsymbol{v} = \boldsymbol{v}_1 + \boldsymbol{v}_2 = (1, 1)$, since $\boldsymbol{v}$ satisfies the clause, and after n iterations its image is $(1, 2^n)$, so that the loop is executed again.

TERMINATION OF LINEAR PROGRAMS (TIWARI)

- The method applies when the variables are $\mathbb{R}$-valued and $c[\boldsymbol{x}, \boldsymbol{x}']$ has the form

$$c[\boldsymbol{x}, \boldsymbol{x}'] = I \wedge \bigwedge_{i=1}^{m} \Big(\sum_{j=1}^{n} b_{ij} x_j > 0 \Big) \wedge \bigwedge_{i=1}^{n} \Big(x'_i = \sum_{j=1}^{n} a_{ij} x_j \Big).$$

- Two cases depending on $\mathbf{A} = (a_{ij})_{i,j=1,\dots,n}$:
  - $\mathbf{A}$ has no positive real eigenvalues, then termination is ensured for all possible inputs.
  - Otherwise there is a non-deterministic algorithm that depends on an asymptotic analysis of powers of $\mathbf{A}$.
- One builds a set of linear constraints (both equalities and inequalities) using the m inequalities in c above:
  - if this set is satisfiable, any vector satisfying it is a witness for non termination.
  - if this set is unsatisfiable the constraints are mutually inconsistent, and there is termination on any input.

TERMINATION OF LINEAR PROGRAMS (TIWARI)

- Consider the program $\{\}$ **while** $x > 0$ **do** $x := x + y$. Here

$$c[x, y, x', y'] = (x > 0 \land x' = x + y \land y' = y)$$

- The matrix $\mathbf{A} = \left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right)$ has the eigenvector $\boldsymbol{v}^{\mathrm{T}} = (1, 0)$ relative to the eigenvalue $\lambda = 1$. The program does not terminate on input $\boldsymbol{v}$.
- Consider the program $\{\}$ **while** $(x > 0 \land -y > 0)$ **do** $x := x + y$. Here

$$c[x, y, x', y'] = (x > 0 \land -y > 0 \land x' = x + y \land y' = y)$$

  and the program terminates on any input.
- In fact, assume that the input vector $(x_0, y_0)^{\mathrm{T}}$ satisfies $(x_0 > 0) \land (-y_0 > 0)$, so that the body loop is executed at least once. After n iterations we have $(x_n, y_n)^{\mathrm{T}} = (x_0 + ny_0, y_0)^{\mathrm{T}}$.
- The first coordinate is a ranking function.

Conclusion

- Termination analysis is essential to prove that program components do not get stuck.
- Developing termination proofs by hand is impossible to conduct reliably on programs longer than a few dozens of lines.
- The authomatic synthesis of ranking functions allows, in a significant number of cases, to prove termination without any human intervention.
- Computed ranking functions constitute termination certificates that can be exhibited (in the spirit of proof-carrying code) to certify mobile code.
- The work described in this seminar is part of an ongoing effort at the universities of Parma and La Réunion to sistematize and extend a number of techniques on the automatic synthesis of ranking functions for the purposes of termination, nontermination and complexity analyses.
- A prototype implementation applying this techniques to the analysis of imperative programs is being developed.