The Automatic Synthesis of Linear Ranking Functions[☆]

Roberto Bagnara[a], Fred Mesnard[b], Andrea Pescetti[c], Enea Zaffanella[a]

[a] *Dipartimento di Matematica, Università di Parma, Italy*
[b] *IREMIA, LIM, Université de la Réunion, France*
[c] *Università di Parma, Italy*

Abstract

The classical technique for proving termination of a generic sequential computer program involves the synthesis of a *ranking function* for each loop of the program. *Linear* ranking functions are particularly interesting because many terminating loops admit one and algorithms exist to automatically synthesize it. In this paper we present two such algorithms: one based on work dated 1991 by Sohn and Van Gelder; the other, due to Podelski and Rybalchenko, dated 2004. Remarkably, while the two algorithms will synthesize a linear ranking function under exactly the same set of conditions, the former is mostly unknown to the community of termination analysis and its general applicability has never been put forward before the present paper. In this paper we thoroughly justify both algorithms, we prove their correctness, we compare their worst-case complexity and experimentally evaluate their efficiency, and we present an open-source implementation of them that will make it very easy to include termination-analysis capabilities in automatic program verifiers.

Keywords: Static analysis, computer-aided verification, termination analysis.

1. Introduction

Termination analysis of computer programs (a term that here we interpret in its broadest sense) consists in attempting to determine whether execution of a given program will definitely terminate for a class of its possible inputs. The ability to anticipate the termination behavior of programs (or fragments thereof) is essential to turn assertions of *partial correctness* (*if* the program reaches a certain control point, *then* its state satisfies some requirements) into assertions of *total correctness* (the program *will* reach that point *and* its state will satisfy those requirements). It is worth observing that the property of

[☆]This work has been partly supported by PRIN project "AIDA—Abstract Interpretation: Design and Applications" and by the Faculty of Sciences of the University of Parma.

Email addresses: `bagnara@cs.unipr.it` (Roberto Bagnara), `Frederic.Mesnard@univ-reunion.fr` (Fred Mesnard), `Andrea.Pescetti@unipr.it` (Andrea Pescetti), `zaffanella@cs.unipr.it` (Enea Zaffanella)

termination of a program fragment is not less important than, say, 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.

Developing termination proofs by hand is, as any other program verification task, tedious, error-prone and, to keep it short, virtually impossible to conduct reliably on programs longer than a few dozens of lines. For this reason, automated termination analysis has been a hot research topic for more than two decades. Of course, due to well-known limitative results of computation theory, any automatic termination analysis can only be expected to give the correct answer ("the program does —or does not— terminate on these inputs") for some of the analyzed programs and inputs: for the other programs and inputs the analysis will be inconclusive ("don't know"). It is worth noticing that there is no need to resort to the halting problem to see how hard proving termination can be. A classical example is the $3x + 1$ problem,[1] whose termination for any n has been a conjecture for more than 70 years:

$$
\begin{array}{l}
\textbf{while}\ n > 1\ \textbf{do} \\
\quad \textbf{if}\ (n\ \textbf{mod}\ 2) \neq 0\ \textbf{then}\ n := 3n + 1 \\
\quad \textbf{else}\ n := n\ \textbf{div}\ 2
\end{array}
$$

The classical technique for proving termination of a generic sequential computer program consists in selecting, for each loop w of the program:

1. a set S_w that is well-founded with respect to a relation R_w;[2]
2. a function f_w from the set of program states that are relevant for w (e.g., those concerning the head of the loop and that are reachable from a designated set of initial states) to the set S_w, such that the values of f_w computed at any two subsequent iterations of w are in relation R_w.

The function f_w is called *ranking function*, since it ranks program states according to their "proximity" to the final states. Let us focus on deterministic programs, and consider a loop w and a set of initial states Σ_w^{I} for w. Assume further that the body of w always terminates when w is initiated in a state $\sigma \in \Sigma_w^{\mathrm{I}}$ and that Σ_w^{F} is a set of final states for w, that is, w immediately terminates when it is initiated in a state $\sigma \in \Sigma_w^{\mathrm{F}}$. If we fix any enumeration of $\Sigma_w^{\mathrm{I}} = \{\sigma_0^0, \sigma_1^0, \ldots\}$, then the computations of w we are interested in can be

[1] Also known as the Collatz problem, the Syracuse problem, Kakutani's problem, Hasse's algorithm, and Ulam's problem: see, e.g., [1].

[2] A set S is *well-founded* with respect to a relation $R \subseteq S \times S$ if, for each $U \subseteq S$ such that $U \neq \varnothing$, there exists $v \in U$ such that $(u, v) \notin R$ for each $u \in U \setminus \{v\}$.

represented by the (possibly infinite) sequence of (possibly infinite) sequences

$$\begin{array}{ccc} \sigma_0^0 & \sigma_0^1 & \dots \\ \vdots & \vdots & \ddots \\ \sigma_i^0 & \sigma_i^1 & \dots \\ \vdots & \vdots & \ddots \end{array} \tag{1}$$

Let Σ_w be the set of all states that occur in (1). Suppose that we succeed in finding a ranking function $f_w \colon \Sigma_w \to S_w$, where S_w is well-founded with respect to R_w and, for each $m, n \in \mathbb{N}$, if σ_m^n and σ_m^{n+1} occur in (1), then $\bigl(f_w(\sigma_m^{n+1}), f_w(\sigma_m^n)\bigr) \in R_w$. In this case we know that all the sequences in (1), and hence all the computations they represent, are finite.

Example 1.1. Consider the following loop, where x takes values in $\mathbb{Z}$:

$$\begin{array}{l} \textbf{while } x \neq 0 \textbf{ do} \\ \qquad x := x - 1 \end{array}$$

Here the state at the loop head can be simply characterized by an integer number: the value of x. If we take $\Sigma^{\mathrm{I}} := \mathbb{N}$ then the computation sequences of interest are

$$\begin{array}{ccccc} 0 & & & & \\ 1 & 0 & & & \\ \vdots & \vdots & \ddots & & \\ n & n-1 & \dots & 0 & \\ \vdots & \vdots & & & \ddots \end{array}$$

We thus have $\Sigma = \mathbb{N}$ and $\Sigma^{\mathrm{F}} = \{0\}$. If we define $S := \mathbb{N}$, f as the identity function over $\mathbb{N}$, and $R := \bigl\{\, (h, k) \bigm| h, k \in \mathbb{N}, h < k \,\bigr\}$, then S is well founded with respect to R and f is a ranking function (with respect to Σ, S and R).

Observe that, in the example above, taking R as the predecessor relation would have worked anyway; f could have been defined as the function mapping h to $2h$, in which case S could have been left as before or defined as the set of even nonnegative integers. . . . In general, if a ranking function exists, an infinite number of them do exist.

The next example shows that freedom in the choice of the well-founded ordering can be used to obtain simpler ranking functions.

Example 1.2. Consider the following program, where variables take values in $\mathbb{N}$ and comments in braces describe the behavior of deterministic program fragments that are guaranteed to terminate and whose details are unimportant:

> **var** a : **array**$[1 \mathrel{..} n]$ **of unsigned integer**;
> { all elements of a are written }
> **while** $a[1] > 0$ **do**
>   { i takes a value between 1 and n such that $a[i] \neq 0$ }
> $a[i] := a[i] - 1$
>   { positions $i+1$, $i+2$, $\dots$, n of a are possibly modified }

Here we can take $\Sigma^{\mathrm{I}} = \Sigma = \mathbb{N}^n$ and $\Sigma^{\mathrm{F}} = \{0\} \times \mathbb{N}^{n-1}$. If we define $S := \mathbb{N}^n$, f as the identity function over $\mathbb{N}^n$, and $R \subset \mathbb{N}^n \times \mathbb{N}^n$ as the lexicographic ordering over $\mathbb{N}^n$, then f is a ranking function with respect to Σ, S and R. Finding a ranking function having $\mathbb{N}$ as codomain would have been much more difficult and could not be done without a complete knowledge of the program fragments we have summarized with the comments between braces.

We have seen that, if there exists a ranking function, then all computations summarized by (1) terminate. What is interesting is that the argument works also the other way around: if all the computations summarized by (1) do terminate, then there exists a ranking function (actually, there exists an infinite number of them). In fact, suppose all the sequences in (1) are finite. Since the program is deterministic, any state occurs only once in every sequence. Moreover, if a state σ occurs in more than one sequence, then the suffixes of these sequences that immediately follow σ are all identical (since the future of any computation is completely determined by its current state). The function mapping each σ in Σ_w to the natural number representing the length of such suffixes is thus well defined and is a ranking function with respect to Σ_w and $\mathbb{N}$ with the well-founded ordering given by the '$<$' relation. It is worth observing that the above argument implies that if any ranking function exists, then there exists a ranking function over $(\mathbb{N}, <)$. This observation can be generalized to programs having bounded nondeterminism [2]: therefore, ranking functions on the naturals are sufficient, for instance, when modeling the input of values for commonly available built-in data types. However, as illustrated by Example 1.2, the use of more general well-founded orderings can simplify the search for a ranking function. Moreover, such a generalization is mandatory when dealing with unbounded nondeterminism [2] (see also [3, Section 10]).

The termination of a set of computations and the existence of a ranking function for such a set are thus completely equivalent. On the one hand, this means that trying to prove that a ranking function exists is, at least in principle, not less powerful than any other method we may use to prove termination. On the other hand, undecidability of the termination problem implies that the existence of a ranking function is also undecidable. An obvious way to prove the existence of a ranking function is to *synthesize* one from the program text and a description of the initial states: because of undecidability, there exists no algorithm that can do that in general.

The use of ranking functions as a tool to reason about termination can be traced back to the seminal work of R. W. Floyd in [4], where they are introduced under the name of W*-functions*. Since then, several variations of the method have been proposed so as to extend its applicability from the realm of classical sequential programs to more general constructs (e.g., concurrency). In particular, in [3], seven different '*à la Floyd*' induction principles for nondeterministic transition systems are formally shown to be sound, semantically complete and equivalent. For instance, it is shown that it is sufficient to consider a single, global ranking function, instead of a different ranking function for each program control point, as originally proposed in [4]; and that the decrease of such a global ranking function need not be verified at all program control points, but it is enough to consider a minimal set of *loop cut-points*; moreover, when trying to prove properties that only depend on the current state of the system (e.g., termination of a deterministic program), it is always possible to find a ranking function depending on the current state only, i.e., independent of the initial state of the system. Note that these results have been implicitly exploited in the examples above so as to simplify the presentation of the method.

In this paper we present, in very general terms so as to encompass any programming paradigm, the approach to termination analysis based on the explicit search of ranking functions. We then restrict attention to linear ranking functions obtained from linear approximations of the program's semantics. For this restriction, we present and fully justify two methods to prove the existence of linear ranking functions: one, based on work dated 1991 by Sohn and Van Gelder, that is almost unknown outside the field of logic programming even though, as we demonstrate in the present paper, it is completely general; the other, due to Podelski and Rybalchenko, dated 2004, was proved correct by the authors but the reasons why it works were never presented. We then provide a proof of equivalence of the two methods, thus providing an independent assessment of their correctness and relative completeness. We also compare their theoretical complexity and practical efficiency on three related problems:

1. proving that one linear ranking function exists;
2. exhibiting one such function;
3. computing the space of all linear ranking functions.

The experimental evaluation is based on the implementation of the two methods provided by the *Parma Polyhedra Library* [5], a free software library of numerical abstraction targeted at software/hardware analysis and verification. These implementations are, to the best of our knowledge, the first ones that are being made available, in source form, to the community. For this reason, the implementations should be regarded as complementary to the present paper in the common aim of making the automatic synthesis of linear ranking functions known outside programming language barriers, understandable and accessible.

The plan of the paper is as follows: Section 2 recalls preliminary notions and introduces the notation used throughout the paper; Section 3 introduces the problem of automatic termination analysis of individual loops and its solution technique based on the synthesis of ranking functions; Section 4 presents a simple generalization of the approach of [6] that is generally applicable to termination analysis of any language; Section 5 shows and fully justifies the approach of [7]; Section 6 proves the two methods are equivalent and compares them from the point of view of computational complexity; Section 7 presents the implementation of the two approaches offered by the Parma Polyhedra Library and the corresponding experimental evaluation, providing a comparison of their practical efficiency; Section 8 concludes.

Readers who are interested in a more detailed exposition of the themes treated in this paper are referred to the technical report version of this paper [8].

2. Preliminaries

2.1. Set Theory

The set of all finite sequences of elements of S is denoted by S^*. The empty sequence is denoted by ε and the length of a sequence w is denoted by $|w|$.

The set of non-negative integers, rationals and reals are denoted by $\mathbb{N}$, $\mathbb{Q}_+$ and $\mathbb{R}_+$, respectively.

2.2. Linear Algebra

For each $i \in \{1, \ldots, n\}$, v_i denotes the i-th component of the real (column) vector $\boldsymbol{v} = \langle v_1, \ldots, v_n \rangle \in \mathbb{R}^n$. A vector $\boldsymbol{v} \in \mathbb{R}^n$ can also be interpreted as a matrix in $\mathbb{R}^{n \times 1}$ and manipulated accordingly with the usual definitions for addition, multiplication (both by a scalar and by another matrix), and transposition, which is denoted by $\boldsymbol{v}^{\mathrm{T}}$, so that $\langle v_1, \ldots, v_n \rangle = (v_1, \ldots, v_n)^{\mathrm{T}}$. If $\boldsymbol{v} \in \mathbb{R}^n$ and $\boldsymbol{w} \in \mathbb{R}^m$, we will write $\langle \boldsymbol{v}, \boldsymbol{w} \rangle$ to denote the column vector in $\mathbb{R}^{n+m}$ obtained by "concatenating" $\boldsymbol{v}$ and $\boldsymbol{w}$, so that $\langle \boldsymbol{v}, \boldsymbol{w} \rangle = \langle v_1, \ldots, v_n, w_1, \ldots, w_m \rangle$. The *scalar product* of $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^n$ is the real number $\boldsymbol{v}^{\mathrm{T}} \boldsymbol{w} = \sum_{i=1}^{n} v_i w_i$. The identity matrix in $\mathbb{R}^{n \times n}$ is denoted by $\boldsymbol{I}_n$. We write $\mathbf{0}$ to denote a matrix in $\mathbb{R}^{n \times m}$ having all of its components equal to zero; the dimensions n and m will be clear from context. We sometimes treat scalars as vectors in $\mathbb{R}^1$ or matrices in $\mathbb{R}^{1 \times 1}$.

For any relational operator $\bowtie \in \{<, \leq, =, \geq, >\}$, we write $\boldsymbol{v} \bowtie \boldsymbol{w}$ to denote the conjunctive proposition $\bigwedge_{i=1}^{n} (v_i \bowtie w_i)$. Moreover, $\boldsymbol{v} \neq \boldsymbol{w}$ will denote the proposition $\neg(\boldsymbol{v} = \boldsymbol{w})$. We will sometimes use the convenient notation $a \bowtie_1 b \bowtie_2 c$ to denote the conjunction $a \bowtie_1 b \wedge b \bowtie_2 c$ and we will not distinguish conjunctions of propositions from sets of propositions. The same notation applies to vectors defined over other numeric fields and, for the supported operations, to vectors defined over numeric sets such as $\mathbb{N}$ and $\mathbb{Q}_+$.

2.3. First-Order Logic

A triple $\Sigma = (S, F, R)$ is a *signature* if S is a set of *sort symbols*, $F := (F_{w,s})_{w \in S^*, s \in S}$ is a family of sets of *function symbols* and $R := (R_w)_{w \in S^*}$ is a family of sets of *relation symbols* (or *predicate symbols*). If $F_{s_1 \cdots s_n, s} \ni f$ we use the standard notation for functions and write $F \ni f \colon s_1 \times \cdots \times s_n \to s$.

Similarly, if $R_{s_1 \cdots s_n} \ni p$ we use the standard notation for relations and write $R \ni p \subseteq s_1 \times \cdots \times s_n$. A Σ-*structure* $\mathcal{A} = (S^{\mathcal{A}}, F^{\mathcal{A}}, R^{\mathcal{A}})$ consists of: a set $S^{\mathcal{A}}$ containing one arbitrary set $s^{\mathcal{A}}$ for each sort symbol $s \in S$; a family $F^{\mathcal{A}}$ of sets of functions such that, for each $f \colon s_1 \times \cdots \times s_n \to s$, a function $f^{\mathcal{A}} \colon s_1^{\mathcal{A}} \times \cdots \times s_n^{\mathcal{A}} \to s^{\mathcal{A}}$ belongs to $F^{\mathcal{A}}$; a family $R^{\mathcal{A}}$ of sets of relations defined similarly.

Let X be a denumerable set of *variable symbols*. The set of (Σ, X)-*terms* (or, briefly, *terms*) is inductively defined as usual: elements of X are terms; elements of $F_{\varepsilon,s}$ are terms; and for each $f \in F_{w,s}$ with $|w| = k$, if $t_1, \ldots, t_k$ are terms, then $f(t_1, \ldots, t_k)$ is a term. If $p \in R_w$ with $|w| = k$ and $t_1, \ldots, t_k$ are terms, then $p(t_1, \ldots, t_k)$ is an *atomic* (Σ, X)-*formula*. (Σ, X)-*formulas* are built as usual from atomic formulas and logical connectives and quantifiers. The first-order language $\mathcal{L}(\Sigma, X)$ is the set of all (Σ, X)-*formulas*. The notion of *bound* and *free variable occurrence* in a formula are also defined in the standard way. We will routinely confuse a tuple of variables with the set of its components. So, if ϕ is a (Σ, X)-formula, we will write $\phi[\bar{x}]$ to denote ϕ itself, yet emphasizing that the set of free variables in ϕ is included in $\bar{x}$. Let $\bar{x}, \bar{y} \in X^*$ be of the same length and let ϕ be a (Σ, X)-formula: then $\phi[\bar{y}/\bar{x}]$ denotes the formula obtained by simultaneous renaming of each free occurrence in ϕ of a variable in $\bar{x}$ with the corresponding variable in $\bar{y}$, possibly renaming bound variable occurrences as needed to avoid variable capture. Notice that $\phi[\bar{x}]$ implies $\bigl(\phi[\bar{y}/\bar{x}]\bigr)[\bar{y}]$, for each admissible $\bar{y} \in X^*$.

A formula with no free variable occurrences is termed *closed* or called a *sentence*. If ϕ is a closed (Σ, X)-formula and $\mathcal{A}$ is a Σ-structure, we write $\mathcal{A} \models \phi$ if ϕ is satisfied when interpreting each symbol in Σ as the corresponding object in $\mathcal{A}$. A set $\mathcal{T}$ of closed (Σ, X)-formulas is called a (Σ, X)-*theory*. We write $\mathcal{A} \models \mathcal{T}$ if $\mathcal{A} \models \phi$ for each $\phi \in \mathcal{T}$. If ϕ is a closed (Σ, X)-formula and $\mathcal{T}$ is a (Σ, X)-theory, we write $\mathcal{T} \models \phi$ if, for each Σ-structure $\mathcal{A}$, $\mathcal{A} \models \mathcal{T}$ implies $\mathcal{A} \models \phi$. In this case we say that ϕ is a *logical consequence* of $\mathcal{T}$.

3. Termination Analysis of Individual Loops

We will start by restricting our attention to individual loops of the form

$$\{\, I \,\} \ \textbf{while}\ B\ \textbf{do}\ C \tag{2}$$

where

- I is a loop invariant that a previous analysis phase has determined to hold just before any evaluation of B;
- B is a Boolean guard expressing the condition on the state upon which iteration continues;
- C is a command that, in the context set by (2), is known to always terminate.

Notice that, for maximum generality, we do not impose any syntactic restriction on I, B and C and will only observe their interaction with the program state: I and B express conditions on the state, and C is seen as a state transformer, that is, a condition constraining the program states that correspond to its initial and final states. We assume that such conditions are expressed in a fragment of some first-order language $\mathcal{L} = \mathcal{L}(\Sigma, X)$ that is closed under finite conjunction and implication. We assume further that the meaning of the sentences in $\mathcal{L}$ is given by some theory $\mathcal{T}$ for which we are given a sound inference procedure denoted by '$\vdash$', that is, for each sentence $\phi \in \mathcal{L}$, if $\mathcal{T} \vdash \phi$ then $\mathcal{T} \models \phi$. Finally, we fix a Σ-structure $\mathcal{D}$ such that $\mathcal{D} \models \mathcal{T}$, which captures the domain over which computation and program reasoning take place. Let $\bar{x}$ be the tuple of variables containing (among possible others) all the free variables of (2). The effect of C within the loop can be captured by stipulating that $\bar{x}$ characterizes the state *before* execution of C, introducing a tuple of new variables $\bar{x}'$ that characterizes the state *after* C's execution, and by imposing restrictions on the combined tuple $\bar{x}\bar{x}'$. Our last assumption is that we are given formulas of $\mathcal{L}$ that correctly express the semantics of I, B, and C: let us call these formulas ϕ_I, ϕ_B and ϕ_C, respectively. With these definitions and assumptions, the behavior of loop (2) is correctly approximated as follows:

1. whenever the loop guard B is evaluated, $\phi_I[\bar{x}]$ holds;
2. if $\phi_I[\bar{x}] \wedge \phi_B[\bar{x}]$ is inconsistent, iteration of the loop terminates;
3. just before execution of C, $\phi_I[\bar{x}] \wedge \phi_B[\bar{x}]$ holds;
4. just after execution of C, $\phi_I[\bar{x}] \wedge \phi_B[\bar{x}] \wedge \phi_C[\bar{x}\bar{x}']$ holds.

It is worth observing that the presence of the externally-generated invariant I is not restrictive: on the one hand, $\phi_I[\bar{x}]$ can simply be the "true" formula, when nothing better is available; on the other hand, non trivial invariants are usually a decisive factor for the precision of termination analysis. As observed in [9], the requirement that I must hold before any evaluation of B can be relaxed by allowing I not to hold finitely many times.[3] The same kind of approximation can be applied to ϕ_I, ϕ_B and ϕ_C by only requesting that they eventually hold.

We would like to stress that, at this stage, we have not lost generality. While the formalization of basic iteration units in terms of while loops has an unmistakable imperative flavor, it is general enough to capture iteration in other programming paradigms [8].

The approach to termination analysis based on ranking functions requires that:

1. a set $\mathcal{O}$ and a binary relation $\prec \subseteq \mathcal{O} \times \mathcal{O}$ are selected so that $\mathcal{O}$ is well-founded with respect to '$\prec$';
2. a term $\delta[\bar{y}]$ of $\mathcal{L}$ is found such that

$$\mathcal{T} \vdash \forall\Big(\big(\phi_I[\bar{x}] \wedge \phi_B[\bar{x}] \wedge \phi_C[\bar{x}\bar{x}']\big) \to \omega\big(\delta[\bar{x}'/\bar{y}], \delta[\bar{x}/\bar{y}]\big)\Big), \tag{3}$$

[3] Such an invariant is called *tail invariant* in [9].

where the interpretation of ω over $\mathcal{D}$ corresponds to '$\prec$'; the function associated to δ in $\mathcal{D}$ is called *ranking function* for the loop (2).

Termination of (2) follows by the correctness of ϕ_I, ϕ_B, ϕ_C and '$\vdash$', and by well-foundedness of $\mathcal{O}$ with respect to '$\prec$'. To see this, suppose, towards a contradiction, that loop (2) does not terminate. The mentioned soundness conditions would imply the existence of an infinite sequence of elements of $\mathcal{O}$

$$o_0 \succ o_1 \succ o_2 \succ \cdots \tag{4}$$

Let $U \subseteq \mathcal{O}$ be the (nonempty) set of elements in the sequence. Since $\mathcal{O}$ is well founded with respect to '$\prec$', there exists $j \in \mathbb{N}$ such that, for each $i \in \mathbb{N}$ with $i \neq j$, $o_i \not\prec o_j$. But this is impossible, as, for each $j \in \mathbb{N}$, $o_{j+1} \prec o_j$. This means that the infinite chain (4) cannot exist and loop (2) terminates.

Example 3.1. Let $\Sigma = (S, F, R)$ with $S = \{\mathsf{i}\}$, $F = F_{\varepsilon,\mathsf{i}} \cup F_{\mathsf{i},\mathsf{i}}$, $F_{\varepsilon,\mathsf{i}} = \{\mathsf{0}\}$, $F_{\mathsf{i},\mathsf{i}} = \{\mathsf{s}\}$ and $R = R_{\mathsf{i}\cdot\mathsf{i}} = \{=, <\}$. Let also $\mathcal{D} = \bigl(\{\mathbb{Z}\}, \{0, s\}, \{e, l\}\bigr)$ be a Σ-structure where $s = \{\, (n, n+1) \mid n \in \mathbb{Z} \,\}$, $e = \{\, (n, n) \mid n \in \mathbb{Z} \,\}$ and $l = \{\, (n, m) \mid n, m \in \mathbb{Z}, n < m \,\}$. Let X be a denumerable set of variable symbols and let $\mathcal{T}$ be the theory of arithmetic restricted to $\mathcal{L}(\Sigma, X)$. Consider now the loop

$$\begin{gathered}\{x \geq 0\}\\ \textbf{while } x \neq 0 \textbf{ do}\\ x := x - 1\end{gathered}$$

We have $\phi_I = (x = \mathsf{0} \vee \mathsf{0} < x)$, $\phi_B = \neg(x = \mathsf{0})$ and $\phi_C = \bigl(\mathsf{s}(x') = x\bigr)$. If we take $(\mathcal{O}, \prec) = (\mathbb{N}, l \cap \mathbb{N}^2)$, $\delta[y] = y$, and $\omega(\tau, \upsilon) = (\mathsf{0} < \upsilon \wedge \tau < \upsilon)$, we can substitute into (3) and obtain

$$\mathcal{T} \vdash \forall x, x' : \bigl((x = \mathsf{0} \vee \mathsf{0} < x) \wedge \neg(x = \mathsf{0}) \wedge \mathsf{s}(x') = x\bigr) \rightarrow (\mathsf{0} < x \wedge x' < x),$$

which simplifies to

$$\mathcal{T} \vdash \forall x, x' : \bigl(\mathsf{0} < x \wedge \mathsf{s}(x') = x\bigr) \rightarrow (\mathsf{0} < x \wedge x' < x),$$

which a reasonable inference engine can easily check to be true.

This general view of the ranking functions approach to termination analysis allows us to compare the methods in the literature on a common ground and focusing on what, besides mere presentation artifacts, really distinguishes them from one another. Real differences have to do with:

- the choice of the well-founded ordering $(\mathcal{O}, \prec)$;
- the class of functions in which the method "searches" for the ranking functions;
- the choice of the signature Σ, the domain $\mathcal{D}$ and theory $\mathcal{T}$; this has to accommodate the programming formalism at hand, the semantic characterization upon which termination reasoning has to be based, the axiomatization of $(\mathcal{O}, \prec)$, and the representation of ranking functions;

- the class of algorithms that the method uses to conduct such a search.

We now briefly review these aspects.

The most natural well-founded ordering is, of course, $(\mathbb{N}, <)$. This is especially indicated when the termination arguments are based on quantities that can be expressed by natural numbers. This is the case, for instance, of the work by Sohn and Van Gelder for termination analysis of logic programs [6, 10]. Orderings based on $\mathbb{Q}_+$ or $\mathbb{R}_+$ can be obtained by imposing over them relations like those defined, for each $\epsilon > 0$, by $<_\epsilon := \bigl\{ (h, k) \in \mathbb{S}_+^2 \bigm| h + \epsilon \leq k \bigr\}$, where $\epsilon \in \mathbb{S}_+$ and $\mathbb{S}_+ = \mathbb{Q}_+$ or $\mathbb{S}_+ = \mathbb{R}_+$, respectively. Of course, this is simply a matter of convenience: a ranking function f with codomain $(\mathbb{R}_+, <_\epsilon)$ can always be converted into a ranking function g with codomain $(\mathbb{N}, <)$ by taking $g(\bar{y}) = \lfloor f(\bar{y})\epsilon^{-1} \rfloor$. Similarly, any ranking function over $(\mathbb{R}_+, <_\epsilon)$ can be converted into a ranking function over $(\mathbb{R}_+, <_1)$. On tuples, the *lexicographic* ordering is the most common choice for a well-founded relation: given a finite number of well-founded relations $\prec_i$ for $i = 1, \ldots, n$ over a set $\mathbb{S}$, the lexicographic ordering over $\mathbb{S}^n$ is induced by saying that $\boldsymbol{s} \prec \boldsymbol{t}$ if and only if $s_i \prec_i t_i$ for an index i and $s_j = t_j$ for all indices $j < i$. The termination analyzer of the Mercury programming language [11, 12] first attempts an analysis using the $(\mathbb{N}, <)$ ordering; if that fails then it resorts to lexicographic orderings. Lexicographic orderings on Cartesian products of $(\mathbb{R}+, <_\epsilon)$ are also used in [13].

The synthesis of ranking functions is easily seen to be a search problem. All techniques impose limits upon the universe of functions that is the domain of the search. For instance, in the logic programming community, [14, 12, 15, 11] use ranking functions of the form $f(x_1, \ldots, x_n) = \sum_{i=1}^n \mu_i x_i$, where, for $i = 1, \ldots, n$, $\mu_i \in \{0, 1\}$ and the variable x_i takes values in $\mathbb{N}$. The method of Sohn and Van Gelder [6, 10] is restricted to linear functions of the form $f(x_1, \ldots, x_n) = \sum_{i=1}^n \mu_i x_i$, where, for $i = 1, \ldots, n$, $\mu_i \in \mathbb{N}$ and the variable x_i takes values in $\mathbb{N}$. The generalization by Mesnard and Serebrenik [16, 17] obtains affine functions of the form $f(x_1, \ldots, x_n) = \mu_0 + \sum_{i=1}^n \mu_i x_i$, where $\mu_i \in \mathbb{Z}$ and x_i take values in $\mathbb{Q}$ or $\mathbb{R}$, for $i = 0, \ldots, n$. Recently, Nguyen and De Schreye [18] proposed, in the context of logic programming and following a thread of work in termination of term rewrite systems that can be traced back to [19], to use polynomial ranking functions. These are of the basic form $f(x_1, \ldots, x_n) = \mu_0 + \sum_{j=1}^m \mu_j \prod_{i=1}^n x_i^{k_{ij}}$ where, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$, $\mu_j \in \mathbb{Z}$, $k_{ij} \in \mathbb{N}$ and the variable x_i takes values in $\mathbb{Z}$ [20]. Several further restriction are usually imposed: first a *domain* $A \subseteq \mathbb{N}$ is selected; then it is demanded that, for each $x_1, \ldots, x_n \in A$, $f(x_1, \ldots, x_n) \in A$ and that f is *strictly monotone* over A on all its arguments. The set of all such polynomials is itself well-founded with respect to '$<_A$': $f <_A g$ if and only if, for each $x_1, \ldots, x_n \in A$, $f(x_1, \ldots, x_n) < g(x_1, \ldots, x_n)$. The condition of strict monotonicity, namely, for each $x_1, \ldots, x_n \in A$, each $i = 1, \ldots, n$, and each $y, z \in A$ with $y < z$, $f(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n) < f(x_1, \ldots, x_{i-1}, z, x_{i+1}, \ldots, x_n)$, is ensured if, for each $j = 1, \ldots, m$, we have $\mu_j \in \mathbb{N}$ and, for each $i = 0, \ldots, n$, there exists j such that $\mu_j \neq 0$ and $k_{ij} \neq 0$. Choosing $A \neq \mathbb{N}$ brings some advantages. For example, if $A \subseteq \{ n \in \mathbb{N} \mid n \geq 2 \}$ then multiplication of polynomials is strictly monotone on both its arguments (i.e., $f <_A f \cdot g$ and $g <_A f \cdot g$). Further restrictions are usually imposed in order to make the search of ranking functions tractable: both the maximum degree of polynomials and their coefficients —the μ_j's above— can be severely limited (an upper bound of 2 both on degrees and on coefficients is typical). Quadratic ranking functions of the form $f(x_1, \ldots, x_n) = \langle x_1, \ldots, x_n, 1\rangle^{\mathrm{T}} \boldsymbol{M} \langle x_1, \ldots, x_n, 1\rangle$ are considered in [21], where the variables x_i and the unknown coefficients μ_{ij} of the $(n+1) \times (n+1)$ symmetric matrix $\boldsymbol{M}$ take values in $\mathbb{R}$. [13] considers a search space of tuples of (up to a fixed number of) linear functions.

The logic used in most papers about the synthesis of linear (or affine) ranking functions (such as [22, 6, 7]) is restricted to finite conjunctions of linear equalities or inequalities. In [13] this logic is extended to include disjunction, so as to capture precisely the effect of the loop body.

Concerning algorithms, the restriction to conjunctions of linear equalities or inequalities allows the use of the simplex algorithm (or other algorithms for linear programming) to prove the existence of linear ranking functions in [6, 7] or to synthesize one of them. When a space of ranking functions is sought, these can be obtained by projecting the systems of constraints onto a designated set of variables using, for instance, Fourier-Motzkin elimination. In these approaches, standard algorithms from linear programming work directly on an abstraction of the loop to be analyzed and are able to decide the existence of linear ranking functions for that abstraction. The algorithms used in other approaches belong to the category of "generate and test" algorithms: the "generate" phase consists in the selection, possibly guided by suitable heuristics, of candidate functions, while the "test" phase amounts to prove that a candidate is indeed a ranking function. This is the case, for instance, of [13], where generation consists in the instantiation of *template functions* and testing employs an algorithm based on a variant of Farkas' Lemma. Non-linear constraints generated by the method described in [21] are handled by first resorting to semidefinite programming solvers and then validating the results obtained by using some other tools, since these solvers are typically based on interior point algorithms and hence may incur into unsafe rounding errors. Note that, in principle, the very same observation would apply to the case of linear constraints, if the corresponding linear programming problem is solved using an interior point method or even a floating-point based implementation of the simplex algorithm; however, there exist implementations of the simplex algorithm based on exact arithmetic, so that linear programming problems can be numerically solved incurring no rounding errors at all and with a computational overhead that is often acceptable.[4]

It should be noted that the fact that in this paper we only consider simple while loops is not restrictive. In fact, the *Size-Change Principle*, introduced by [23] (see also [24]), implies that one can safely trade the existence of a potentially complex *global* ranking function for the whole program for the existence

[4]In contrast, an exact solver for non-linear constraints would probably require a truly symbolic computation, incurring a much more significant computational overhead.

of elementary *local* ranking functions of some selected individual simple loops appearing in a transformation of the whole program. Moreover, in a generalization of this work presented in [25], the authors prove that, under a certain hypothesis,[5] linear ranking functions are a large enough class of local ranking functions for a sound and complete termination criterion that encompasses global lexicographic ranking functions.

4. The Approach of Sohn and Van Gelder, Generalized

As far as we know, the first approach to the automatic synthesis of ranking functions is due to Kirack Sohn and Allen Van Gelder [10, 6]. Possibly due to the fact that their original work concerned termination of logic programs, Sohn and Van Gelder did not get the recognition we believe they deserve. In fact, as we will show, some key ideas of their approach can be applied, with only rather simple modifications, to the synthesis of ranking functions for any programming paradigm.

4.1. Ranking Functions for Binary, Directly Recursive CLP($\mathbb{N}$) *Programs*

We refer the interested reader to [8] for a more complete reconstruction of the approach by Sohn and Van Gelder. For the other readers, suffices it to say that termination analysis of logic programs can be mapped onto termination analysis of CLP($\mathbb{N}$) programs that represent the sizes of the terms in the original program. In turn, the termination analysis of general CLP($\mathbb{N}$) programs can be approximated by termination analysis of *directly recursive*, *binary* CLP($\mathbb{N}$) clauses, we deal first with a single clause

$$p(\bar{x}) :- c[\bar{x}, \bar{x}'], p(\bar{x}'), \tag{5}$$

where p is a predicate symbol, $\bar{x}$ and $\bar{x}'$ are disjoint n-tuples of variables, and $c[\bar{x}, \bar{x}']$ is a linear constraint involving variables in $\bar{x} \cup \bar{x}'$.[6] The meaning of such a clause is that, if p is called on some tuple of integers $\bar{x}$, then there are two cases:

- $c[\bar{x}, \bar{x}']$ is unsatisfiable (i.e., there does not exist a tuple of integers $\bar{x}'$ that, together with $\bar{x}$, satisfies it), in which case the computation will fail, and thus terminate;
- there exists $\bar{x}'$ such that $c[\bar{x}, \bar{x}']$ holds, in which case the computation proceeds with the (recursive) calls $p(\bar{x}')$, for each $\bar{x}'$ such that $c[\bar{x}, \bar{x}']$.

The question is now to see whether that recursive procedure is *terminating*, that is whether, for each $\bar{x} \in \mathbb{N}^n$, the call $p(\bar{x})$ will only give rise to chains of

[5] Namely, that the program is approximated by *monotonicity constraints*, constraints of the form $x \leq y$ or $x < y$.

[6] We abuse notation by confusing a tuple with the set of its elements.

recursive calls of finite length. The approach of Sohn and Van Gelder allows to synthesize a function $f_p \colon \mathbb{N}^n \to \mathbb{N}$ such that

$$\forall \bar{x}, \bar{x}' \in \mathbb{N}^n : c[\bar{x}, \bar{x}'] \implies f_p(\bar{x}) > f_p(\bar{x}'). \tag{6}$$

This means that the measure induced by f_p strictly decreases when passing from a call of p to its recursive call. Since the naturals are well founded, this entails that p, as defined in (5), is terminating.

A very important contribution of Sohn and Van Gelder consists in the algorithm they give to construct a class of functions that satisfy (6). The class is constituted by linear functions of the form

$$f_p(y_1, \ldots, y_n) = \sum_{i=1}^{n} \mu_i y_i, \tag{7}$$

where $\mu_i \in \mathbb{N}$, for $i = 1, \ldots, n$. For this class of functions and by letting $\bar{\mu} = (\mu_1, \ldots, \mu_n)$, condition (6) can be rewritten as

$$\exists \bar{\mu} \in \mathbb{N}^n \mathrel{.} \forall \bar{x}, \bar{x}' \in \mathbb{N}^n : c[\bar{x}, \bar{x}'] \implies \sum_{i=1}^{n} \mu_i x_i - \sum_{i=1}^{n} \mu_i x_i' \geq 1. \tag{8}$$

Given that $c[\bar{x}, \bar{x}']$ is a linear constraint, for any choice of $\bar{\mu} \in \mathbb{N}^n$ we can easily express (8) as an optimization problem over the naturals. In order to move from tuple notation to the more convenient vector notation, assume without loss of generality that, for some $m \in \mathbb{N}$, $\boldsymbol{A}_c \in \mathbb{Z}^{m \times 2n}$ and $\boldsymbol{b}_c \in \mathbb{Z}^m$ are such that $\boldsymbol{A}_c \langle \boldsymbol{x}, \boldsymbol{x}' \rangle \geq \boldsymbol{b}_c$ is logically equivalent to $c[\bar{x}, \bar{x}']$ under the obvious, respective interpretations. Then, for any candidate choice of $\boldsymbol{\mu} \in \mathbb{N}^n$, condition (8) is equivalent to imposing that the optimization problem

$$\begin{array}{rl} \text{minimize} & \theta = \langle \boldsymbol{\mu}, -\boldsymbol{\mu} \rangle^{\mathrm{T}} \langle \boldsymbol{x}, \boldsymbol{x}' \rangle \\ \text{subject to} & \boldsymbol{A}_c \langle \boldsymbol{x}, \boldsymbol{x}' \rangle \geq \boldsymbol{b}_c \\ & \boldsymbol{x}, \boldsymbol{x}' \in \mathbb{N}^n \end{array} \tag{9}$$

is either unsolvable or has an optimal solution whose *cost* $\hat{\theta}$ is such that $\hat{\theta} \geq 1$. If this is the case, then $\boldsymbol{\mu}$ induces, according to (7), a function f_p satisfying (6). Notice that, for any fixed choice of $\boldsymbol{\mu} \in \mathbb{N}^n$, θ is a linear expression and hence (9) is an integer linear programming (ILP) problem. This gives us an expensive way (since ILP is an NP-complete problem [26]) to test whether a certain $\boldsymbol{\mu} \in \mathbb{N}^n$ is a witness for termination of (5), but gives us no indication about where to look for such a tuple of naturals.

A first step forward consists in considering the relaxation of (9) obtained by replacing the integrality constraints $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{N}^n$ with $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{Q}_+^n$. This amounts to trading precision for efficiency. In fact, since any feasible solution of (9) is also feasible for the relaxed problem, if the optimum solution of the latter has a cost greater than or equal to 1, then either (9) is unfeasible or $\hat{\theta} \geq 1$. However, we may have $\hat{\theta} \geq 1$ even if the optimum of the relaxation is less than 1.[7] On the other hand, the relaxed problem is a linear problem: so by giving up completeness we have passed from an NP-complete problem to a problem in P for which we have, in addition, quite efficient algorithms. Furthermore, we observe that although the parameters $\boldsymbol{\mu}$ are naturals in (8), this condition can be relaxed as well: if $\boldsymbol{\mu} \in \mathbb{Q}_+^n$ gives a relaxed problem with optimum greater than 1, then we can multiply this vector by a positive natural so as to obtain a tuple of naturals satisfying (8). The relaxation can now be written using the standard linear programming (LP) notation:

$$\begin{array}{rl} \text{minimize} & \langle \boldsymbol{\mu}, -\boldsymbol{\mu} \rangle^{\mathrm{T}} \langle \boldsymbol{x}, \boldsymbol{x}' \rangle \\ \text{subject to} & \boldsymbol{A}_c \langle \boldsymbol{x}, \boldsymbol{x}' \rangle \geq \boldsymbol{b}_c \\ & \langle \boldsymbol{x}, \boldsymbol{x}' \rangle \geq \mathbf{0} \end{array} \tag{10}$$

We still do not know how to determine the vector of parameters $\boldsymbol{\mu}$ so that the optimum of (10) is at least 1, but here comes one of the brilliant ideas of Sohn and Van Gelder: passing to the *dual*. It is a classical result of LP theory that every LP problem can be converted into an *equivalent* dual problem. The dual of (10) is

$$\begin{array}{rl} \text{maximize} & \boldsymbol{b}_c^{\mathrm{T}} \boldsymbol{y} \\ \text{subject to} & \boldsymbol{A}_c^{\mathrm{T}} \boldsymbol{y} \leq \langle \boldsymbol{\mu}, -\boldsymbol{\mu} \rangle \\ & \boldsymbol{y} \geq \mathbf{0} \end{array} \tag{11}$$

where $\boldsymbol{y}$ is an m-column vector of (dual) unknowns. Duality theory ensures that if both (10) and (11) have bounded feasible solutions, then both of them have optimal solutions and these solutions have the same cost. More formally, for every choice of the parameters $\boldsymbol{\mu} \in \mathbb{Q}_+^n$, if $\langle \hat{\boldsymbol{x}}, \hat{\boldsymbol{x}}' \rangle \in \mathbb{Q}^{2n}$ is an optimal solution for (10) and $\hat{\boldsymbol{y}} \in \mathbb{Q}^m$ is an optimal solution for (11), then $\langle \boldsymbol{\mu}, -\boldsymbol{\mu} \rangle^{\mathrm{T}} \langle \hat{\boldsymbol{x}}, \hat{\boldsymbol{x}}' \rangle = \boldsymbol{b}_c^{\mathrm{T}} \hat{\boldsymbol{y}}$. Moreover, if one of (10) and (11) is unfeasible, then the other is either unbounded or unfeasible. In contrast, if one of (10) and (11) is unbounded, then the other is definitely unfeasible.

Thus, thanks to duality theory, the LP problems (10) and (11) are equivalent for our purposes and we can consider any one of them. Suppose we analyze the dual problem (11):

- If (11) is unfeasible then either (10) is unfeasible, which implies trivial termination of (5), or (10) is unbounded, in which case—since we are working on relaxations—nothing can be concluded about whether $\boldsymbol{\mu}$ defines a ranking function for (5).
- If (11) is feasible and unbounded then (10) is unfeasible and (5) trivially terminates.

[7] Let us consider the clause: $p(x) :- 2x \geq 2x' + 1, p(x')$ with $\mu = 1$. The optimization over the integers leads to $\hat{\theta} = 1$, whereas the optimization for the relaxation has $\hat{\theta} = \frac{1}{2}$.

- If (11) is feasible and bounded, then we have proved termination ($\boldsymbol{\mu}$ induces a ranking function) if the cost of the optimal solution is at least 1 (actually, any positive rational could be used instead of 1). The analysis is inconclusive otherwise.

The crucial point is that, in (11), the parameters $\boldsymbol{\mu}$ occur linearly, whereas in (10) they are multiplied by $\langle \boldsymbol{x}, \boldsymbol{x}' \rangle$. So we can treat $\boldsymbol{\mu}$ as a vector of variables and transform (11) into the new LP problem in $m+n$ variables

$$\begin{array}{ll} \text{maximize} & \langle \boldsymbol{b}_c, \boldsymbol{0} \rangle^{\mathrm{T}} \langle \boldsymbol{y}, \boldsymbol{\mu} \rangle \\ \text{subject to} & \begin{pmatrix} \boldsymbol{A}_c^{\mathrm{T}} & -\boldsymbol{I}_n \\ & \boldsymbol{I}_n \end{pmatrix} \langle \boldsymbol{y}, \boldsymbol{\mu} \rangle \leq \boldsymbol{0} \\ & \langle \boldsymbol{y}, \boldsymbol{\mu} \rangle \geq \boldsymbol{0} \end{array} \tag{12}$$

The requirement that, in order to guarantee termination of (5), the optimal solutions of (10) and (11) should not be less than 1 can now be captured by incorporating $\boldsymbol{b}_c^{\mathrm{T}} \boldsymbol{y} \geq 1$ into the constraints of (12), yielding

$$\begin{array}{ll} \text{maximize} & \langle \boldsymbol{b}_c, \boldsymbol{0} \rangle^{\mathrm{T}} \langle \boldsymbol{y}, \boldsymbol{\mu} \rangle \\ \text{subject to} & \begin{pmatrix} \boldsymbol{A}_c^{\mathrm{T}} & -\boldsymbol{I}_n \\ & \boldsymbol{I}_n \\ -\boldsymbol{b}_c^{\mathrm{T}} & \boldsymbol{0}^{\mathrm{T}} \end{pmatrix} \langle \boldsymbol{y}, \boldsymbol{\mu} \rangle \leq \begin{pmatrix} \boldsymbol{0} \\ -1 \end{pmatrix} \\ & \langle \boldsymbol{y}, \boldsymbol{\mu} \rangle \geq \boldsymbol{0} \end{array} \tag{13}$$

There are several possibilities:

1. If (13) is unfeasible, then:
   (a) If (12) is unfeasible, then, for each $\boldsymbol{\mu} \in \mathbb{Q}_+^n$, (11) is unfeasible and:
      i. If (10) is unfeasible, then (5) trivially terminates;
      ii. otherwise (10) is unbounded and we can conclude nothing about the termination of (5).
   (b) If (12) is feasible, then it is bounded by a rational number $q < 1$. Thus, for each $\check{\boldsymbol{\mu}} \in \mathbb{Q}_+^n$ extracted from a feasible solution $\langle \check{\boldsymbol{y}}, \check{\boldsymbol{\mu}} \rangle \in \mathbb{Q}_+^{m+n}$ of (12), the corresponding LP problem (11) is also feasible, bounded, and its optimum $q' \in \mathbb{Q}$ is such that $q' \leq q < 1$. Moreover, we must have $q' \leq 0$. In fact, if $q' > 0$, problem (10) instantiated over $\check{\boldsymbol{\mu}}' := \check{\boldsymbol{\mu}}/q'$ would have an optimal solution of cost 1; the same would hold for the corresponding dual (11), but this would contradict the hypothesis that (12) is bounded by $q < 1$. Hence $q' \leq 0$. Since by duality the optimum of problem (10) is q', the analysis is inconclusive.
2. If (13) is feasible, let $\langle \check{\boldsymbol{y}}, \check{\boldsymbol{\mu}} \rangle \in \mathbb{Q}^{m+n}$ be any of its feasible solutions. Choosing $\check{\boldsymbol{\mu}}$ for the values of the parameters, (11) is feasible. There are two further possibilities:
   (a) either (11) is unbounded, so (5) trivially terminates;
   (b) or it is bounded by a rational $q \geq 1$ and the same holds for its dual (10).

In both cases, $\check{\boldsymbol{\mu}}$, possibly multiplied by a positive natural in order to get a tuple of naturals, defines, via (7), a ranking function for (5).

The above case analysis boils down to the following algorithm:

1. Use the simplex algorithm to determine the feasibility of (13), ignoring the objective function. If it is feasible, then any feasible solution induces a linear ranking function for (5); exit with success.
2. If (13) is unfeasible, then try to determine the feasibility of (9) (e.g., by using the simplex algorithm again to test whether the relaxation (10) is feasible). If (9) is unfeasible then (5) trivially terminates; exit with success.
3. Exit with failure (the analysis is inconclusive).

An example should serve to better clarify the methodology we have employed.

Example 4.1. In the CLP($\mathbb{N}$) program

$$p(x_1, x_2) \mathrel{:-} x_1 \leq 1 \land x_2 = 0,$$
$$p(x_1, x_2) \mathrel{:-} x_1 \geq 2 \land 2x'_1 + 1 \geq x_1 \land 2x'_1 \leq x_1 \land x'_2 + 1 = x_2, p(x'_1, x'_2),$$

$p(x_1, x_2)$ is equivalent to

$$x_2 = \begin{cases} \lfloor \log_2(x_1) \rfloor, & \text{if } x_1 \neq 0; \\ 0, & \text{otherwise.} \end{cases}$$

The relaxed optimization problem in LP notation (10) is:[8]

$$\begin{array}{ll} \text{minimize} & \langle \mu_1, \mu_2, -\mu_1, -\mu_2 \rangle^{\mathrm{T}} \langle x_1, x_2, x'_1, x'_2 \rangle \\ \text{subject to} & \begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & 0 & 2 & 0 \\ 1 & 0 & -2 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x'_1 \\ x'_2 \end{pmatrix} \geq \begin{pmatrix} 2 \\ -1 \\ 0 \\ 1 \\ -1 \end{pmatrix} \\ & \langle x_1, x_2, x'_1, x'_2 \rangle \geq \mathbf{0} \end{array}$$

and the dual optimization problem (11) is:

$$\begin{array}{ll} \text{maximize} & \langle 2, -1, 0, 1, -1 \rangle^{\mathrm{T}} \langle y_1, y_2, y_3, y_4, y_5 \rangle \\ \text{subject to} & \begin{pmatrix} 1 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{pmatrix} \leq \begin{pmatrix} \mu_1 \\ \mu_2 \\ -\mu_1 \\ -\mu_2 \end{pmatrix} \\ & \langle y_1, y_2, y_3, y_4, y_5 \rangle \geq \mathbf{0} \end{array}$$

[8] We will tacitly replace an equality in the form $\alpha = \beta$ by the equivalent pair of inequalities $\alpha \geq \beta$ and $-\alpha \geq -\beta$ whenever the substitution is necessary to fit our framework.

Incorporation of the unknown coefficients of $\boldsymbol{\mu}$ among the problem variables finally yields as the transformed problem (13):

$$\begin{array}{ll}
\text{maximize} & \langle 2,-1,0,1,-1,0,0\rangle^{\mathrm{T}}\langle y_1,y_2,y_3,y_4,y_5,\mu_1,\mu_2\rangle \\
\text{subject to} & \begin{pmatrix} 1 & -1 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & -1 \\ 0 & 2 & -2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 1 \\ -2 & 1 & 0 & -1 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ \mu_1 \\ \mu_2 \end{pmatrix} \leq \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -1 \end{pmatrix} \\
& \langle y_1,y_2,y_3,y_4,y_5,\mu_1,\mu_2\rangle \geq \mathbf{0}
\end{array} \tag{14}$$

This problem is feasible so this CLP($\mathbb{N}$) program terminates. Projecting the constraints of (14) onto $\boldsymbol{\mu}$ we obtain, in addition, the knowledge that every $\boldsymbol{\mu}$ with $\mu_1 + \mu_2 \geq 1$ gives a ranking function. In other words, $\mu_1 x_1 + \mu_2 x_2$ is a ranking function if the non-negative numbers μ_1 and μ_2 satisfy $\mu_1 + \mu_2 \geq 1$.

The following result illustrates the strength of the method:

Theorem 4.2. *Let C be the binary CLP($\mathbb{Q}_+$) clause $p(\bar{x})$:− $c[\bar{x},\bar{x}'], p(\bar{x}')$, where p is an n-ary predicate and $c[\bar{x},\bar{x}']$ is a linear satisfiable constraint. Let $\mathrm{plrf}(C)$ be the set of positive linear ranking functions for C and $\mathrm{svg}(C)$ be the set of solutions of (13) projected onto $\boldsymbol{\mu}$, that is,*

$$\mathrm{plrf}(C) := \left\{ \boldsymbol{\mu} \in \mathbb{Q}_+^n \;\middle|\; \forall \bar{x}, \bar{x}' \in \mathbb{Q}_+^n : c[\bar{x},\bar{x}'] \implies \sum_{i=1}^{n} \mu_i x_i - \sum_{i=1}^{n} \mu_i x_i' \geq 1 \right\},$$
$$\mathrm{svg}(C) := \left\{ \check{\boldsymbol{\mu}} \in \mathbb{Q}_+^n \;\middle|\; \langle \check{\boldsymbol{y}}, \check{\boldsymbol{\mu}} \rangle \text{ is a solution of (13)} \right\}.$$

Then $\mathrm{plrf}(C) = \mathrm{svg}(C)$.

Proof. As $c[\bar{x},\bar{x}']$ is satisfiable, problem (10) is feasible. We prove each inclusion separately.

$\mathrm{svg}(C) \subseteq \mathrm{plrf}(C)$. Assume that (13) is feasible and let $\langle \check{\boldsymbol{y}}, \check{\boldsymbol{\mu}} \rangle$ be a solution of (13). For this choice of $\check{\boldsymbol{\mu}}$, the corresponding LP problems (10) and (11) are bounded by $q \geq 1$ (case 2b of the discussion above). So $\check{\boldsymbol{\mu}} \in \mathrm{plrf}(C)$.

$\mathrm{plrf}(C) \subseteq \mathrm{svg}(C)$. Let us pick $\boldsymbol{\mu} \in \mathrm{plrf}(C)$. For this choice, the corresponding LP problem (10) is bounded by $r \geq 1$, so is its dual (11). Let $\hat{\boldsymbol{y}}$ be an optimal solution for (11). Thus $\langle \hat{\boldsymbol{y}}, \boldsymbol{\mu} \rangle$ is a feasible solution of (12) and (13). Hence $\boldsymbol{\mu} \in \mathrm{svg}(C)$.

As an immediate consequence, the question: *does a given binary recursive clause admit a positive linear mapping?* can be solved in polynomial time.

Corollary 4.3. *Let C be the binary* CLP($\mathbb{Q}_+$) *clause* $p(\bar{x})$:− $c[\bar{x}, \bar{x}'], p(\bar{x}')$, *where* $c[\bar{x}, \bar{x}']$ *is a linear satisfiable constraint. The decision problem* $\mathrm{plrf}(C) = \varnothing$ *is in* P.

PROOF. By Theorem 4.2 the problems $\mathrm{plrf}(C) = \varnothing$ and $\mathrm{svg}(C) = \varnothing$ are equivalent. So, if (13) is feasible then the answer is *no*: as $c[\bar{x}, \bar{x}']$ is satisfiable, we are in case (2)(b). Otherwise, again because of the satisfiability of $c[\bar{x}, \bar{x}']$, either (10) is unbounded (case (1)(a)ii) or it is bounded by $q' < 0$ (case (1)(b)). In both cases, the answer is *yes*. Finally, testing the satisfiability of a linear system, as well as computing one of its solutions —and thus computing one concrete linear ranking function—, is in P (see, e.g., [27]).

For the case where we have more than one directly recursive binary CLP($\mathbb{N}$) clauses, $C_1, \ldots, C_n$, the set of *global* positive linear ranking functions, i.e., that ensure termination whichever clause is selected at each computation step, is given by $\bigcap_{i=1}^{n} \mathrm{svg}(C_i)$. This can be computed by taking the conjunction of the constraints obtained, for each clause, from the projection of the constraints of the corresponding linear problem (13) onto $\boldsymbol{\mu}$.

To summarize, the main contribution of Sohn and Van Gelder lies in their encoding of the ranking function search problem into linear programming and their use of the duality theorem. As we will see, this idea is amenable to a generalization that makes it widely applicable to any programming paradigm, not just (constraint) logic programming.

4.2. The Generalization by Mesnard and Serebrenik

Fred Mesnard and Alexandre Serebrenik have generalized the method of Sohn and Van Gelder from the analysis of logic programs to the analysis of CLP($\mathbb{Q}$) and CLP($\mathbb{R}$) programs in [16, 17]. In the following, for presentation purposes and without loss of generality, we consider the case of rational-valued variables. They use a class of affine ranking functions of the form

$$f_p(y_1, \ldots, y_n) = \mu_0 + \sum_{i=1}^{n} \mu_i y_i, \tag{15}$$

where $\mu_i \in \mathbb{Q}$, for $i = 0, \ldots, n$. Allowing for rational-valued coefficients μ_i and variables y_i (both the μ_i's and the y_i's were naturals in [6]) implies that (15) does not necessarily define a nonnegative function and that Zeno sequences[9] are not automatically excluded. Consequently, to avoid these two problems, condition (6) is strengthened to[10]

$$\forall \bar{x}, \bar{x}' \in \mathbb{Q}^n : c[\bar{x}, \bar{x}'] \implies \big(f_p(\bar{x}) \geq 1 + f_p(\bar{x}') \wedge f_p(\bar{x}) \geq 0\big). \tag{16}$$

[9]Such as $1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots$.

[10]Our presentation is strictly more general than the formulation in [16, 17], which imposes that $f_p(\bar{x}) \geq 1 + f_p(\bar{x}') \wedge f_p(\bar{x}') \geq 0$.

Note that the choice of the numbers 1 and 0 in the right hand side of the above implication preserves generality: the general form of the former condition, i.e., $f_p(\bar{x}) \geq \epsilon + f_p(\bar{x}')$ for a fixed $\epsilon \in \mathbb{Q}_+^*$, can be transformed as shown in Section 3, and the general form of the latter, i.e., $f_p(\bar{x}) \geq b$ for a fixed $b \in \mathbb{Q}$, can be transformed into $f_p(\bar{x}) \geq 0$ by a suitable choice of μ_0. Condition (16) can be rewritten as follows:

$$\forall \bar{x}, \bar{x}' \in \mathbb{Q}^n : c[\bar{x}, \bar{x}'] \implies \left(\sum_{i=1}^{n} \mu_i x_i - \sum_{i=1}^{n} \mu_i x_i' \geq 1 \wedge \mu_0 + \sum_{i=1}^{n} \mu_i x_i \geq 0 \right). \tag{17}$$

Using the same notation chosen for (9), the existence of a ranking function can now be equivalently expressed as the existence of a solution of cost at least 1 to the former and a solution of cost at least 0 to the latter of the following optimization problems:

$$\begin{array}{rl} \text{minimize} & \langle \boldsymbol{\mu}, -\boldsymbol{\mu} \rangle^{\mathrm{T}} \langle \boldsymbol{x}, \boldsymbol{x}' \rangle \\ \text{subject to} & \boldsymbol{A}_c \langle \boldsymbol{x}, \boldsymbol{x}' \rangle \geq \boldsymbol{b}_c \end{array} \qquad \begin{array}{rl} \text{minimize} & \langle \tilde{\boldsymbol{\mu}}, \boldsymbol{0} \rangle^{\mathrm{T}} \langle \tilde{\boldsymbol{x}}, \boldsymbol{x}' \rangle \\ \text{subject to} & \tilde{\boldsymbol{A}}_c \langle \tilde{\boldsymbol{x}}, \boldsymbol{x}' \rangle \geq \tilde{\boldsymbol{b}}_c \end{array} \tag{18}$$

where the extended vectors $\tilde{\boldsymbol{\mu}} := \langle \mu_0, \boldsymbol{\mu} \rangle$ and $\tilde{\boldsymbol{x}} := \langle x_0, \boldsymbol{x} \rangle$ include the parameter μ_0 and the new variable x_0, respectively, and the extended matrix and vector

$$\tilde{\boldsymbol{A}}_c := \begin{pmatrix} 1 & \boldsymbol{0}^{\mathrm{T}} \\ -1 & \boldsymbol{0}^{\mathrm{T}} \\ \boldsymbol{0} & \boldsymbol{A}_c \end{pmatrix} \quad \text{and} \quad \tilde{\boldsymbol{b}}_c := \langle 1, -1, \boldsymbol{b}_c \rangle$$

encode the additional constraint $x_0 = 1$.

Reasoning as in Section 4.1, the problems (18) can then be transformed, applying the suitable form of the duality theorem, into the following dual problems over new vectors of variables $\boldsymbol{y}$ and $\boldsymbol{z}$, ranging over $\mathbb{Q}^m$ and $\mathbb{Q}^{m+2}$, respectively:

$$\begin{array}{rl} \text{maximize} & \boldsymbol{b}_c^{\mathrm{T}} \boldsymbol{y} \\ \text{subject to} & \boldsymbol{A}_c^{\mathrm{T}} \boldsymbol{y} = \langle \boldsymbol{\mu}, -\boldsymbol{\mu} \rangle \\ & \boldsymbol{y} \geq \boldsymbol{0} \end{array} \qquad \begin{array}{rl} \text{maximize} & \tilde{\boldsymbol{b}}_c^{\mathrm{T}} \boldsymbol{z} \\ \text{subject to} & \tilde{\boldsymbol{A}}_c^{\mathrm{T}} \boldsymbol{z} = \langle \tilde{\boldsymbol{\mu}}, \boldsymbol{0} \rangle \\ & \boldsymbol{z} \geq \boldsymbol{0} \end{array} \tag{19}$$

Now the condition that the optimal solution is at least 1 (resp., 0) can be added to the constraints, thus reducing the optimization problems (18) to testing the satisfiability of the system:

$$\begin{cases} \boldsymbol{b}_c^{\mathrm{T}} \boldsymbol{y} \geq 1 \\ \boldsymbol{A}_c^{\mathrm{T}} \boldsymbol{y} = \langle \boldsymbol{\mu}, -\boldsymbol{\mu} \rangle \\ \boldsymbol{y} \geq \boldsymbol{0} \\ \\ \tilde{\boldsymbol{b}}_c^{\mathrm{T}} \boldsymbol{z} \geq 0 \\ \tilde{\boldsymbol{A}}_c^{\mathrm{T}} \boldsymbol{z} = \langle \tilde{\boldsymbol{\mu}}, \boldsymbol{0} \rangle \\ \boldsymbol{z} \geq \boldsymbol{0} \end{cases}$$

or equivalently, after incorporating the parameters $\boldsymbol{\mu}$ (resp., $\tilde{\boldsymbol{\mu}}$) into the variables, to the generalization to $\mathbb{Q}$ of problem (13):

$$\left(\begin{array}{c:c} \boldsymbol{A}_c^{\mathrm{T}} & -\boldsymbol{I}_n \\ & \boldsymbol{I}_n \\ \hdashline -\boldsymbol{A}_c^{\mathrm{T}} & \boldsymbol{I}_n \\ & -\boldsymbol{I}_n \\ \hdashline -\boldsymbol{I}_m & \mathbf{0} \\ \hdashline -\boldsymbol{b}_c^{\mathrm{T}} & \mathbf{0}^{\mathrm{T}} \end{array}\right) \langle \boldsymbol{y}, \boldsymbol{\mu} \rangle \leq \left(\begin{array}{c} \mathbf{0} \\ \hdashline \mathbf{0} \\ \hdashline \mathbf{0} \\ \hdashline -1 \end{array}\right) \bigwedge \left(\begin{array}{c:c} \tilde{\boldsymbol{A}}_c^{\mathrm{T}} & -\boldsymbol{I}_{n+1} \\ & \mathbf{0} \\ \hdashline -\tilde{\boldsymbol{A}}_c^{\mathrm{T}} & \boldsymbol{I}_{n+1} \\ & \mathbf{0} \\ \hdashline -\boldsymbol{I}_{m+2} & \mathbf{0} \\ \hdashline -\tilde{\boldsymbol{b}}_c^{\mathrm{T}} & \mathbf{0}^{\mathrm{T}} \end{array}\right) \langle \boldsymbol{z}, \tilde{\boldsymbol{\mu}} \rangle \leq \mathbf{0} \quad (20)$$

The following completeness result generalizes Theorem 4.2:

Theorem 4.4. *Let C be the binary* CLP($\mathbb{Q}$) *clause $p(\bar{x}) :- c[\bar{x}, \bar{x}'], p(\bar{x}')$, where p is an n-ary predicate and $c[\bar{x}, \bar{x}']$ is a linear satisfiable constraint. Let* lrf(C) *be the set of linear ranking functions for C and* ms(C) *be the set of solutions of* (20) *projected onto μ, that is,*

$$\mathrm{lrf}(C) := \left\{ \tilde{\boldsymbol{\mu}} \in \mathbb{Q}^{n+1} \,\middle|\, \begin{array}{l} \forall \bar{x}, \bar{x}' \in \mathbb{Q}^n : c[\bar{x}, \bar{x}'] \implies \\ \quad \sum_{i=1}^n \mu_i x_i - \sum_{i=1}^n \mu_i x_i' \geq 1 \wedge \mu_0 + \sum_{i=1}^n \mu_i x_i \geq 0 \end{array} \right\},$$

$$\mathrm{ms}(C) := \left\{ \tilde{\boldsymbol{\mu}} \in \mathbb{Q}^{n+1} \,\middle|\, \langle \boldsymbol{y}, \boldsymbol{\mu} \rangle \textit{ and } \langle \boldsymbol{z}, \tilde{\boldsymbol{\mu}} \rangle \textit{ are solutions of the problems } (20) \right\}.$$

Then lrf(C) = ms(C).

Proof. We use l and r as subscripts of our references to the LP problems (18), (19), and (20) to denote the LP problems on the left and the LP problems on the right.

ms$(C) \subseteq$ lrf(C). Assume that (20) is feasible and let $\langle \check{\boldsymbol{y}}, \check{\boldsymbol{\mu}} \rangle$ be a solution of $(20)_l$ and $\langle \check{\boldsymbol{z}}, \check{\boldsymbol{\mu}} \rangle$ be a solution of $(20)_r$. For this choice of $\check{\boldsymbol{\mu}}$, the corresponding LP problems $(19)_l$ and $(18)_l$ are bounded by 1 while the corresponding LP problems $(19)_r$ and $(18)_r$ are bounded by 0. Hence we have:

$$\forall \bar{x}, \bar{x}' \in \mathbb{Q}^n : c[\bar{x}, \bar{x}'] \implies \sum_{i=1}^n \mu_i x_i - \sum_{i=1}^n \mu_i x_i' \geq 1$$

and

$$\forall \bar{x}, \bar{x}' \in \mathbb{Q}^n : c[\bar{x}, \bar{x}'] \implies \mu_0 + \sum_{i=1}^n \mu_i x_i \geq 0$$

Thus:

$$\forall \bar{x}, \bar{x}' \in \mathbb{Q}^n : c[\bar{x}, \bar{x}'] \implies \quad \sum_{i=1}^n \mu_i x_i - \sum_{i=1}^n \mu_i x_i' \geq 1 \wedge \mu_0 + \sum_{i=1}^n \mu_i x_i \geq 0$$

so $\check{\boldsymbol{\mu}} \in$ lrf(C).

$\mathrm{lrf}(C) \subseteq \mathrm{ms}(C)$. Let us pick $\tilde{\boldsymbol{\mu}} \in \mathrm{lrf}(C)$. For this choice, the corresponding LP problem (18) are bounded by 1 and 0, so are their duals (19). Let $\hat{\boldsymbol{y}}$ be an optimal solution for $(19)_l$. Thus $\langle \hat{\boldsymbol{y}}, \boldsymbol{\mu} \rangle$ is a feasible solution of $(20)_l$. Similarly, let $\hat{\boldsymbol{z}}$ be an optimal solution for $(19)_r$. Thus $\langle \hat{\boldsymbol{z}}, \tilde{\boldsymbol{\mu}} \rangle$ is a feasible solution of $(20)_r$. Hence $\tilde{\boldsymbol{\mu}} \in \mathrm{ms}(C)$.

Moreover, even for the case of $\mathrm{CLP}(\mathbb{Q})$ —and $\mathrm{CLP}(\mathbb{R})$— checking for the existence of a linear ranking function is a polynomial problem.

Corollary 4.5. *Let C be the binary $\mathrm{CLP}(\mathbb{Q})$ clause $p(\bar{x}) :- c[\bar{x}, \bar{x}'], p(\bar{x}')$, where $c[\bar{x}, \bar{x}']$ is a linear satisfiable constraint. The decision problem $\mathrm{lrf}(C) = \varnothing$ is in* P.

A space of ranking functions can be obtained (at a computational price that is no longer polynomial) by projecting the constraints of (20) onto $\tilde{\boldsymbol{\mu}}$. Any $\tilde{\boldsymbol{\mu}}$ satisfying all the projected constraints corresponds to one ranking function that, subject to $c[\bar{x}, \bar{x}']$, is bounded from below by 0 and that decreases by at least 1 at each iteration. From these "normalized" ranking functions, the opposite of the transformation outlined in Section 3 allows to recover all affine ranking functions: these are induced by the set of parameters

$$\{\, \langle h, k\boldsymbol{\mu} \rangle \mid \langle \mu_0, \boldsymbol{\mu} \rangle \in \mathrm{lrf}(C), h \in \mathbb{Q}, k \in \mathbb{Q}_+ \setminus \{0\} \,\}. \tag{21}$$

4.3. Application to the Analysis of Imperative While Loops

The generalization of Mesnard and Serebrenik can be used, almost unchanged, to analyze the termination behavior of imperative while loops with integer- or rational-valued variables. Consider a loop of the form (2), i.e., $\{\, I \,\}$ **while** B **do** C where I is known to hold before any evaluation of B and C is known to always terminate in that loop. Termination analysis is conducted as follows:

1. Variables are duplicated: if $\bar{x}$ are the n variables of the original loop, we introduce a new tuple of variables $\bar{x}'$.
2. An analyzer based on convex polyhedra [28] is used to analyze the following program:

$$\begin{array}{l} \{I\} \\ x'_1 := x_1;\ \ldots;\ x'_n := x_n; \\ \textbf{if } B[\bar{x}'/\bar{x}] \textbf{ then} \\ \quad C[\bar{x}'/\bar{x}] \\ \quad \bigstar \end{array} \tag{22}$$

   Let the invariant obtained for the program point marked with '$\bigstar$' be $c[\bar{x}, \bar{x}']$; this is a finite conjunction of linear constraints.
3. The method of Mesnard and Serebrenik is now applied to the $\mathrm{CLP}(\mathbb{Q})$ clause $p(\bar{x}) :- c[\bar{x}, \bar{x}'], p(\bar{x}')$: if termination can be established for that clause, then the while loop we started with will terminate.

Notice how the clause $p(\bar{x}) \coloneq c[\bar{x}, \bar{x}'], p(\bar{x}')$ approximates the termination behavior of the loop: if we interpret the predicate p applied to $\bar{x}$ as "the loop guard is evaluated on values $\bar{x}$," then the clause can be read as "if the loop guard is evaluated on values $\bar{x}$, and $c[\bar{x}, \bar{x}']$ holds, then the loop guard will be evaluated again on values $\bar{x}'$."

We illustrate the overall methodology with an example.

Example 4.6. The following program, where x_1 and y take values in $\mathbb{Z}$, computes and stores in x_2 the integer base-2 logarithm of x_1 if $x_1 > 0$, 0 otherwise:

$$
\begin{array}{l}
x_2 := 0; \\
\{x_1 \geq 0 \wedge x_2 \geq 0\} \\
\textbf{while } x_1 \geq 2 \textbf{ do} \\
\qquad x_1 := x_1 \textbf{ div } 2; \\
\qquad x_2 := x_2 + 1
\end{array}
$$

where the loop invariant $\{x_1 \geq 0 \wedge x_2 \geq 0\}$ has been obtained by static analysis. After the duplication of variables, we submit to the analyzer the program

$$
\begin{array}{l}
\{x_1 \geq 0 \wedge x_2 \geq 0\} \\
x_1' := x_1;\ x_2' := x_2; \\
\textbf{if } x_1' \geq 2 \textbf{ then} \\
\qquad x_1' := x_1' \textbf{ div } 2; \\
\qquad x_2' := x_2' + 1 \\
\qquad \bigstar
\end{array}
$$

and we obtain, for program point '$\bigstar$', the invariant

$$x_1 \geq 2 \wedge 2x_1' + 1 \geq x_1 \wedge 2x_1' \leq x_1 \wedge x_2' = x_2 + 1 \wedge x_2' \geq 1.$$

Applying the method of Mesnard and Serebrenik we obtain that, for each $\mu_0, \mu_1, \mu_2 \in \mathbb{Q}$ such that $\mu_1 - \mu_2 \geq 1$, $\mu_2 \geq 0$, and $\mu_0 + 2\mu_1 \geq 0$, $f(x_1, x_2) := \mu_0 + \mu_1 x_1 + \mu_2 x_2$ is a ranking function for the given while loop. It is interesting to observe that the first constraint guarantees strict decrease (at least 1), the addition of the second constraint guarantees boundedness from below, while the further addition of the third constraint ensures nonnegativity, i.e., that 0 is a lower bound.

4.4. Application to the Conditional Termination Analysis

It is interesting to observe that the method of Mesnard and Serebrenik is immediately applicable in *conditional termination analysis.* This is the problem of (automatically) inferring the preconditions under which code that does not universally terminate (i.e., there are inputs for which it does loop forever) is guaranteed to terminate. This problem has been recently studied in [29], where preconditions are inferred under which functions that are either decreasing or bounded become proper ranking functions. The two systems in (20), projected onto $\tilde{\boldsymbol{\mu}}$, exactly define the space of non-negative candidate ranking functions and the space of decreasing candidate ranking functions, respectively. While this is subject for future research, we believe that the availability of these two spaces allows to improve the techniques presented in [29].

5. The Approach of Podelski and Rybalchenko

Andreas Podelski and Andrey Rybalchenko [7] introduce a method for finding linear ranking functions for a particular class of unnested while loops that, with the help of a preliminary analysis phase, is indeed completely general.

Consider a while loop of the form

$$\begin{array}{l} \{I\} \\ \textbf{while } B \textbf{ do} \\ \qquad \blacktriangledown \\ \qquad C \\ \qquad \bigstar \end{array} \tag{23}$$

in which variables $x_1, \ldots, x_n$ occur. Suppose we have determined (e.g., by a data-flow analysis based on convex polyhedra) that the invariant

$$\sum_{i=1}^{n} g_{i,k} x_i \leq b_k, \qquad \text{for } k = 1, \ldots, r, \tag{24}$$

holds at the program point marked with '$\blacktriangledown$', while the invariant

$$\sum_{i=1}^{n} a'_{i,k} x'_i \leq \sum_{i=1}^{n} a_{i,k} x_i + b_k \qquad \text{for } k = r+1, \ldots, r+s, \tag{25}$$

holds at the program point marked with '$\bigstar$', where unprimed variables represent the values before the update and primed variables represent the values after the update, and all the coefficients and variables are assumed to take values in $\mathbb{Q}$.[11]

The inequalities in (24) can be expressed in the form (25) by just defining $a'_{i,k} := 0$ and $a_{i,k} := -g_{i,k}$ for $i = 1, \ldots, n$ and $k = 1, \ldots, r$. The conjunction of (24) and (25) can now be stated in matrix form as

$$\begin{pmatrix} \boldsymbol{A} & \boldsymbol{A}' \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{x}' \end{pmatrix} \leq \boldsymbol{b}, \tag{26}$$

where the matrix $\begin{pmatrix} \boldsymbol{A} & \boldsymbol{A}' \end{pmatrix}$ is obtained by juxtaposition of the two $(r+s) \times n$ matrices $\boldsymbol{A} := (-a_{i,k})$ and $\boldsymbol{A}' := (a'_{i,k})$, $\boldsymbol{b} := \langle b_1, b_2, \ldots, b_{r+s} \rangle$ and, as explained

[11] In [7] variables are said to have integer domain, but this restriction seems unnecessary and, in fact, it is not present in [30].

in Section 2, $\langle \boldsymbol{x}, \boldsymbol{x}' \rangle$ is obtained by juxtaposing the vectors $\boldsymbol{x} := \langle x_1, x_2, \ldots, x_n \rangle$ and $\boldsymbol{x}' := \langle x'_1, x'_2, \ldots, x'_n \rangle$.

Podelski and Rybalchenko have proved that (23) is guaranteed to terminate on all possible inputs if there exist two $(r+s)$-dimensional non-negative rational vectors $\boldsymbol{\lambda}_1$ and $\boldsymbol{\lambda}_2$ such that:

$$\boldsymbol{\lambda}_1^{\mathrm{T}} \boldsymbol{A}' = \boldsymbol{0}^{\mathrm{T}}, \tag{27a}$$

$$(\boldsymbol{\lambda}_1^{\mathrm{T}} - \boldsymbol{\lambda}_2^{\mathrm{T}}) \boldsymbol{A} = \boldsymbol{0}^{\mathrm{T}}, \tag{27b}$$

$$\boldsymbol{\lambda}_2^{\mathrm{T}} (\boldsymbol{A} + \boldsymbol{A}') = \boldsymbol{0}^{\mathrm{T}}, \tag{27c}$$

$$\boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{b} < 0. \tag{27d}$$

Note that we have either zero or infinitely many solutions, since if $(\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2)$ satisfies the constraints then $(k\boldsymbol{\lambda}_1, k\boldsymbol{\lambda}_2)$ satisfies them as well for any $k \in \mathbb{Q}_+ \setminus \{0\}$. Podelski and Rybalchenko proved also the following completeness result: if the behavior of (23) is *completely characterized* by conditions (24) and (25) —in which case they call it a "simple linear loop"— then $\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2 \in \mathbb{Q}_+^{r+s}$ satisfying conditions (27a)–(27d) exist *if and only if* the program terminates for all inputs.

5.1. Generation of Ranking Functions

For each pair of vectors $\boldsymbol{\lambda}_1$ and $\boldsymbol{\lambda}_2$ satisfying the conditions (27a)–(27d), a linear ranking function for the considered program can be obtained as

$$f(\boldsymbol{x}) := \boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{A}' \boldsymbol{x}. \tag{28}$$

In [7] a slightly more complex form is proposed, namely:

$$g(\boldsymbol{x}) := \begin{cases} \boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{A}' \boldsymbol{x}, & \text{if there exists } \boldsymbol{x}' \text{ such that } \begin{pmatrix} \boldsymbol{A} & \boldsymbol{A}' \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{x}' \end{pmatrix} \leq \boldsymbol{b}, \\ (\boldsymbol{\lambda}_2^{\mathrm{T}} - \boldsymbol{\lambda}_1^{\mathrm{T}}) \boldsymbol{b}, & \text{otherwise,} \end{cases} \tag{29}$$

but the extra provisions are actually necessary only if one is interested into an "extended ranking function" that is strictly decreasing also on the very last iteration of the loop, that is, when the effect of the command C is such that $\boldsymbol{x}$ would violate the loop guard B at the following iteration. As this more complex definition does not seem to provide any additional benefit, we disregard it and consider only the linear ranking function (28).

Example 5.1. Consider again the program of Example 4.6. The invariants in the forms dictated by (24) and (25) are given by the systems $\{-x_1 \leq -2, -x'_2 \leq -1\}$ and $\{2x'_1 \leq x_1,\ -2x'_1 - 1 \leq -x_1,\ -x'_2 \leq -x_2 - 1,\ x'_2 \leq x_2 + 1\}$, respectively. These can be expressed in the matrix form (26) by letting

$$\boldsymbol{A} := \begin{pmatrix} -1 & 0 \\ -1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & -1 \\ 0 & 0 \end{pmatrix}, \quad \boldsymbol{A}' := \begin{pmatrix} 0 & 0 \\ 2 & 0 \\ -2 & 0 \\ 0 & -1 \\ 0 & 1 \\ 0 & -1 \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} -2 \\ 0 \\ 1 \\ -1 \\ 1 \\ -1 \end{pmatrix}.$$

Two non-negative rational vectors solving the system (27) are, for instance, $\boldsymbol{\lambda}_1 = \langle 2, 0, 0, 0, 0, 0 \rangle^{\mathrm{T}}$ and $\boldsymbol{\lambda}_2 = \langle 1, 1, 0, 0, 0, 0 \rangle^{\mathrm{T}}$.

5.2. Interpretation in Terms of Lagrangian Relaxation

A reader of [7] wonders where the method of Podelski and Rybalchenko comes from. In fact, the paper does not give an intuition about why conditions (27a)–(27d) imply termination of (23). In [21, Section 6.2], Patrick Cousot hints that the method can be derived from Lagrangian relaxation[12] applied to the Floyd termination verification conditions. We now show that this is indeed the case.

Assuming we are dealing with affine ranking functions and with affine invariants and adding the limitation that $r = 1$ in (24), in [21] the existence of an affine ranking function is proved to be equivalent to the existence of $\mu_0 \in \mathbb{Q}$, $\boldsymbol{\mu} \in \mathbb{Q}^n$, $\delta \in \mathbb{Q}$, $\alpha \in \mathbb{Q}_+$, $\boldsymbol{\beta} \in \mathbb{Q}_+^m$ such that:

$$\begin{aligned}
\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{x} + \mu_0 - \alpha\sigma_1(\boldsymbol{x}, \boldsymbol{x}') &\geq 0, \\
\boldsymbol{\mu}^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}') - \delta - \sum_{k=1}^{m} \beta_k \sigma_k(\boldsymbol{x}, \boldsymbol{x}') &\geq 0, \\
\delta &> 0,
\end{aligned}$$

where n is the number of variables in $\boldsymbol{x}$, the loop is described by the inequalities $\sigma_k(\boldsymbol{x}, \boldsymbol{x}') \geq 0$, with σ_1 being the inequality in (24), and $\boldsymbol{x}$ and $\boldsymbol{x}'$ range on all $\mathbb{Q}^n$.

The limitation that $r = 1$ can actually be removed as long as α is replaced by a vector $\boldsymbol{\alpha} \in \mathbb{Q}_+^r$. The generalization yields

$$\begin{aligned}
\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{x} + \mu_0 - \sum_{k=1}^{r} \alpha_k \sigma_k(\boldsymbol{x}, \boldsymbol{x}') &\geq 0 \\
\boldsymbol{\mu}^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{x}') - \delta - \sum_{k=1}^{m} \beta_k \sigma_k(\boldsymbol{x}, \boldsymbol{x}') &\geq 0 \\
\delta &> 0.
\end{aligned}$$

If, and this is the case in the Podelski and Rybalchenko method, the constraints $\sigma_k(\boldsymbol{x}, \boldsymbol{x}')$ for $k = 1, \ldots, m$ are affine functions of $\langle \boldsymbol{x}, \boldsymbol{x}' \rangle$, the sums can be interpreted as matrix products and the conditions rewritten as follows, where

[12] Lagrangian relaxation is a standard device to convert entailment into constraint solving: given a finite dimensional vector space $\mathbb{V}$, a positive integer n and functions $f_k \colon \mathbb{V} \to \mathbb{Q}$ for $k = 0, \ldots, n$, the property that, for each $\boldsymbol{x} \in \mathbb{V}$, $\bigwedge_{k=1}^{n} f_k(\boldsymbol{x}) \geq 0 \implies f_0(\boldsymbol{x}) \geq 0$ can be *relaxed* to proving the existence of a vector $\boldsymbol{a} \in \mathbb{Q}_+^n$ such that, for all $\boldsymbol{x} \in \mathbb{V}$, $f_0(\boldsymbol{x}) - \sum_{k=1}^{n} a_k f_k(\boldsymbol{x}) \geq 0$. If the f_k are affine functions, the latter condition is equivalent to the former.

$\boldsymbol{A}$, $\boldsymbol{A}'$ and $\boldsymbol{b}$ are the same as in (26):

$$\Bigl(\langle \boldsymbol{\mu}, \mathbf{0}, \mu_0\rangle^{\mathrm{T}} - \langle \boldsymbol{\alpha}, \mathbf{0}\rangle^{\mathrm{T}} \begin{pmatrix} -\boldsymbol{A} & -\boldsymbol{A}' & \boldsymbol{b}\end{pmatrix}\Bigr)\langle \boldsymbol{x}, \boldsymbol{x}', 1\rangle \geq 0 \tag{30a}$$

$$\Bigl(\langle \boldsymbol{\mu}, -\boldsymbol{\mu}, -\delta\rangle^{\mathrm{T}} - \boldsymbol{\beta}^{\mathrm{T}} \begin{pmatrix} -\boldsymbol{A} & -\boldsymbol{A}' & \boldsymbol{b}\end{pmatrix}\Bigr)\langle \boldsymbol{x}, \boldsymbol{x}', 1\rangle \geq 0 \tag{30b}$$

$$\delta > 0 \tag{30c}$$

Note that the inequalities (30a)–(30c) must hold for *every* possible value of $\boldsymbol{x}$ and $\boldsymbol{x}'$ in the whole space $\mathbb{Q}^n$. Therefore, by a suitable choice of $\boldsymbol{x}$ and $\boldsymbol{x}'$, each element of the coefficient vectors in (30a) and (30b) can be shown to be necessarily zero. We define $\boldsymbol{\lambda}_1 = \langle \boldsymbol{\alpha}, \mathbf{0}\rangle$ [13] and $\boldsymbol{\lambda}_2 = \boldsymbol{\beta}$, obtaining:

$$\begin{aligned}
\boldsymbol{\mu} &= -\boldsymbol{\lambda}_1^{\mathrm{T}}\boldsymbol{A}, & \boldsymbol{\mu} &= -\boldsymbol{\lambda}_2^{\mathrm{T}}\boldsymbol{A}, & & \\
\mathbf{0}^{\mathrm{T}} &= -\boldsymbol{\lambda}_1^{\mathrm{T}}\boldsymbol{A}', & -\boldsymbol{\mu} &= -\boldsymbol{\lambda}_2^{\mathrm{T}}\boldsymbol{A}', & & \\
\mu_0 &= \boldsymbol{\lambda}_1^{\mathrm{T}}\boldsymbol{b}, & -\delta &= \boldsymbol{\lambda}_2^{\mathrm{T}}\boldsymbol{b}, & & \delta > 0.
\end{aligned}$$

These relations can finally be rearranged to yield:

$$\begin{aligned}
\boldsymbol{\lambda}_1^{\mathrm{T}}\boldsymbol{A}' &= \mathbf{0}^{\mathrm{T}}, \\
(\boldsymbol{\lambda}_1^{\mathrm{T}} - \boldsymbol{\lambda}_2^{\mathrm{T}})\boldsymbol{A} &= \mathbf{0}^{\mathrm{T}}, \\
\boldsymbol{\lambda}_2^{\mathrm{T}}(\boldsymbol{A} + \boldsymbol{A}') &= \mathbf{0}^{\mathrm{T}}, \\
\boldsymbol{\lambda}_2^{\mathrm{T}}\boldsymbol{b} &< 0,
\end{aligned}
\qquad
\begin{aligned}
\boldsymbol{\mu} &= \boldsymbol{\lambda}_2^{\mathrm{T}}\boldsymbol{A}', \\
\mu_0 &= \boldsymbol{\lambda}_1^{\mathrm{T}}\boldsymbol{b}, \\
\delta &= -\boldsymbol{\lambda}_2^{\mathrm{T}}\boldsymbol{b},
\end{aligned}$$

where the conditions (27a)–(27d) appear on the left hand side and the conditions on the coefficients of the synthesized ranking functions appear on the right hand side, expressed in terms of $\boldsymbol{\lambda}_1$ and $\boldsymbol{\lambda}_2$.

5.3. An Alternative Implementation Approach

As long as the distinction between invariants (24) and (25) is retained, the method of Podelski and Rybalchenko can be implemented following an alternative approach. The linear invariants (26) are more precisely described by

$$\left(\begin{array}{c|c} \boldsymbol{A}_B & \mathbf{0} \\ \hline \boldsymbol{A}_C & \boldsymbol{A}'_C \end{array}\right) \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{x}' \end{pmatrix} \leq \begin{pmatrix} \boldsymbol{b}_B \\ \boldsymbol{b}_C \end{pmatrix} \tag{31}$$

where $\boldsymbol{A}_B \in \mathbb{Q}^{r\times n}$, $\boldsymbol{A}_C \in \mathbb{Q}^{s\times n}$, $\boldsymbol{A}'_C \in \mathbb{Q}^{s\times n}$, $\boldsymbol{b}_B \in \mathbb{Q}^{r}$, $\boldsymbol{b}_C \in \mathbb{Q}^{s}$. As shown in Section 5.2, the existence of a linear ranking function for the system (31) is equivalent to the existence of three vectors $\boldsymbol{v}_1 \in \mathbb{Q}^r_+$, $\boldsymbol{v}_2 \in \mathbb{Q}^r_+$, $\boldsymbol{v}_3 \in \mathbb{Q}^s_+$ such

[13] We explicitly require that the extra coefficients added to $\boldsymbol{\alpha}$ be zero for consistency with the derivation. However, even though Podelski and Rybalchenko admit any nonnegative rational numbers to appear in those positions of $\boldsymbol{\lambda}_1$, there is no loss of generality: the synthesized ranking functions (28) do not depend on these coefficients.

that

$$
\begin{aligned}
(\boldsymbol{v}_1 - \boldsymbol{v}_2)^{\mathrm{T}} \boldsymbol{A}_B - \boldsymbol{v}_3^{\mathrm{T}} \boldsymbol{A}_C &= \mathbf{0}^{\mathrm{T}}, && \text{(32a)}\\
\boldsymbol{v}_2^{\mathrm{T}} \boldsymbol{A}_B + \boldsymbol{v}_3^{\mathrm{T}} (\boldsymbol{A}_C + \boldsymbol{A}'_C) &= \mathbf{0}^{\mathrm{T}}, && \text{(32b)}\\
\boldsymbol{v}_2 \boldsymbol{b}_B + \boldsymbol{v}_3 \boldsymbol{b}_C &< 0. && \text{(32c)}
\end{aligned}
$$

As already noted, the two vectors of the original Podelski and Rybalchenko method can be reconstructed as $\boldsymbol{\lambda}_1 = \langle \boldsymbol{v}_1, \mathbf{0} \rangle$ and $\boldsymbol{\lambda}_2 = \langle \boldsymbol{v}_2, \boldsymbol{v}_3 \rangle$.

Note that, even though we are solving a different linear programming problem, we are still able to obtain the same space of linear ranking functions we would have obtained by applying the original method, as we prove using the following lemma.

Lemma 5.2. *Let S be the space of linear ranking functions obtained by applying the method of Podelski and Rybalchenko to $\begin{pmatrix} \boldsymbol{A} & \boldsymbol{A}' \end{pmatrix} \langle \boldsymbol{x}, \boldsymbol{x}' \rangle \leq \boldsymbol{b}$, i.e.,*

$$
S := \left\{ \langle \boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{A}', \boldsymbol{\lambda}_1^{\mathrm{T}} \boldsymbol{b} \rangle \in \mathbb{Q}^{n+1} \;\middle|\; \langle \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2 \rangle \text{ is a solution of } (27) \right\},
$$

and let $\boldsymbol{P} \in \mathbb{Q}^{(r+s)\times(r+s)}$ be a permutation matrix.[14] *Then the application of the method of Podelski and Rybalchenko to $\boldsymbol{P} \begin{pmatrix} \boldsymbol{A} & \boldsymbol{A}' \end{pmatrix} \langle \boldsymbol{x}, \boldsymbol{x}' \rangle \leq \boldsymbol{P}\boldsymbol{b}$ yields the same space of linear ranking functions S.*

Proof. The system (27) corresponding to $\boldsymbol{P} \begin{pmatrix} \boldsymbol{A} & \boldsymbol{A}' \end{pmatrix} \langle \boldsymbol{x}, \boldsymbol{x}' \rangle \leq \boldsymbol{P}\boldsymbol{b}$ becomes

$$
\begin{aligned}
\boldsymbol{\eta}_1^{\mathrm{T}} \boldsymbol{P} \boldsymbol{A}' &= \mathbf{0}^{\mathrm{T}}, && \text{(33a)}\\
(\boldsymbol{\eta}_1^{\mathrm{T}} - \boldsymbol{\eta}_2^{\mathrm{T}}) \boldsymbol{P} \boldsymbol{A} &= \mathbf{0}^{\mathrm{T}}, && \text{(33b)}\\
\boldsymbol{\eta}_2^{\mathrm{T}} \boldsymbol{P} (\boldsymbol{A} + \boldsymbol{A}') &= \mathbf{0}^{\mathrm{T}}, && \text{(33c)}\\
\boldsymbol{\eta}_2^{\mathrm{T}} \boldsymbol{P} \boldsymbol{b} &< 0, && \text{(33d)}
\end{aligned}
$$

to be solved for the two $(r+s)$-dimensional non-negative rational vectors $\boldsymbol{\eta}_1$ and $\boldsymbol{\eta}_2$.

Now, $\langle \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2 \rangle$ is a solution of (27) if and only if $\langle \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2 \rangle \boldsymbol{P}^{-1}$ is a solution of (33): on one side, if $\langle \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2 \rangle$ is a solution of (27) then $\langle \boldsymbol{\eta}_1, \boldsymbol{\eta}_2 \rangle$ defined as $\langle \boldsymbol{\eta}_1, \boldsymbol{\eta}_2 \rangle := \langle \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2 \rangle \boldsymbol{P}^{-1}$ is a solution of (33); on the other side, if $\langle \boldsymbol{\eta}_1, \boldsymbol{\eta}_2 \rangle$ is a solution of (33) then $\langle \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2 \rangle$ defined as $\langle \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2 \rangle := \langle \boldsymbol{\eta}_1, \boldsymbol{\eta}_2 \rangle \boldsymbol{P}$ is a solution of (27) and the desired property can be verified by right-multiplying by $\boldsymbol{P}^{-1}$ both solutions.

The space of linear ranking functions for the permuted system is

$$
\begin{aligned}
S_{\boldsymbol{P}} &= \left\{ \langle \boldsymbol{\eta}_2^{\mathrm{T}} \boldsymbol{P} \boldsymbol{A}', \boldsymbol{\eta}_1^{\mathrm{T}} \boldsymbol{P} \boldsymbol{b} \rangle \in \mathbb{Q}^{n+1} \;\middle|\; \langle \boldsymbol{\eta}_1, \boldsymbol{\eta}_2 \rangle \text{ is a solution of } (33) \right\}\\
&= \left\{ \langle \boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{P}^{-1} \boldsymbol{P} \boldsymbol{A}', \boldsymbol{\lambda}_1^{\mathrm{T}} \boldsymbol{P}^{-1} \boldsymbol{P} \boldsymbol{b} \rangle \in \mathbb{Q}^{n+1} \;\middle|\; \langle \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2 \rangle \text{ is a solution of } (27) \right\}\\
&= S,
\end{aligned}
$$

and thus it is unaltered with respect to the space of linear ranking functions S corresponding to the non-permuted system.

[14] We recall that a k-dimensional *permutation matrix* is a square matrix obtained by a permutation of the rows or columns of the k-dimensional identity matrix.

Since the system (31) is obtained by applying a suitable permutation to (26), a straightforward application of this lemma proves that the space of linear ranking functions obtained is the same in both cases.

Moreover, as $\boldsymbol{\lambda}_2^{\mathrm{T}}\boldsymbol{A}' = \boldsymbol{v}_3^{\mathrm{T}}\boldsymbol{A}'_C$ and $\boldsymbol{\lambda}_1\boldsymbol{b} = \boldsymbol{v}_1\boldsymbol{b}_B$, we can express the space of linear ranking functions as

$$S := \left\{ \langle \boldsymbol{v}_3^{\mathrm{T}}\boldsymbol{A}'_C, \boldsymbol{v}_1\boldsymbol{b}_B \rangle \in \mathbb{Q}^{n+1} \;\middle|\; \langle \boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3 \rangle \text{ is a solution of (32)} \right\}.$$

6. Comparison of the Two Methods

In this section we compare the method by Mesnard and Serebrenik with the method by Podelski and Rybalchenko: we first prove that they have the same "inferential power", then we compare their worst-case complexities, then we experimentally evaluate them on a representative set of benchmarks.

6.1. Equivalence of the Two Methods

We will now show that the method proposed in [7] is equivalent to the one given in [31] on the class of simple linear loops, i.e., that if one of the two methods can prove termination of a given simple linear loop, then the other one can do the same. This is an expected result since both methods claim to be complete on the class of programs considered.

It is worth noting that a completeness result was already stated in [32, Theorem 5.1] for the case of *single predicate* $\mathrm{CLP}(\mathbb{Q}_+)$ procedures, which can be seen to be a close variant of the binary, directly recursive $\mathrm{CLP}(\mathbb{Q}_+)$ programs considered in Theorem 4.2 and Corollary 4.3. Probably due to the programming paradigm mismatch, Podelski and Rybalchenko [7] fail to recognize the actual strength and generality of the mentioned result, thereby claiming originality for their completeness result.

Theorem 6.1. *Let C be the binary $\mathrm{CLP}(\mathbb{Q})$ clause $p(\bar{x}) :- c[\bar{x}, \bar{x}'], p(\bar{x}')$, where p is an n-ary predicate and $c[\bar{x}, \bar{x}']$ is a linear satisfiable constraint. Let $\mathrm{pr}(C)$ and $\widehat{\mathrm{ms}}(C)$ be the spaces of linear ranking functions for C obtained through the method of Podelski and Rybalchenko and through the method of Mesnard and Serebrenik, respectively, that is,*

$$\mathrm{pr}(C) := \left\{ \langle \boldsymbol{\lambda}_2^{\mathrm{T}}\boldsymbol{A}', \boldsymbol{\lambda}_1^{\mathrm{T}}\boldsymbol{b} \rangle \in \mathbb{Q}^{n+1} \;\middle|\; \langle \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2 \rangle \textit{ is a solution of } (27) \right\},$$

$$\widehat{\mathrm{ms}}(C) := \left\{ k\tilde{\boldsymbol{\mu}} \in \mathbb{Q}^{n+1} \;\middle|\; \begin{array}{l} \langle \boldsymbol{y}, \boldsymbol{\mu} \rangle \textit{ and } \langle \boldsymbol{z}, \tilde{\boldsymbol{\mu}} \rangle \textit{ are solutions of } (20), \\ k \in \mathbb{Q}_+ \setminus \{0\} \end{array} \right\}.$$

where $c[\bar{x}, \bar{x}']$ is equivalent to $\begin{pmatrix} \boldsymbol{A} & \boldsymbol{A}' \end{pmatrix} \langle \boldsymbol{x}, \boldsymbol{x}' \rangle \leq \boldsymbol{b}$ or to $\boldsymbol{A}_c \langle \boldsymbol{x}, \boldsymbol{x}' \rangle \geq \boldsymbol{b}_c$, respectively. Then $\mathrm{pr}(C) = \widehat{\mathrm{ms}}(C)$.

Proof. We will, as customary, prove the two inclusions $\mathrm{pr}(C) \subseteq \widehat{\mathrm{ms}}(C)$ and $\mathrm{pr}(C) \supseteq \widehat{\mathrm{ms}}(C)$.

$\mathrm{pr}(C) \subseteq \widehat{\mathrm{ms}}(C)$. Suppose that there exist two non-negative rational vectors $\boldsymbol{\lambda}_1$ and $\boldsymbol{\lambda}_2$ satisfying (27), i.e., $\boldsymbol{\lambda}_1^{\mathrm{T}} \boldsymbol{A}' = (\boldsymbol{\lambda}_1^{\mathrm{T}} - \boldsymbol{\lambda}_2^{\mathrm{T}})\boldsymbol{A} = \boldsymbol{\lambda}_2^{\mathrm{T}}(\boldsymbol{A} + \boldsymbol{A}') = \boldsymbol{0}^{\mathrm{T}}$ and $\boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{b} < 0$. By Theorem 4.4, it is enough to prove that $\langle \boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{A}', \boldsymbol{\lambda}_1^{\mathrm{T}} \boldsymbol{b} \rangle \in \widehat{\mathrm{ms}}(C)$, that is, there exists $k \in \mathbb{Q}_+$ such that $\langle \frac{1}{k} \boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{A}', \frac{1}{k} \boldsymbol{\lambda}_1^{\mathrm{T}} \boldsymbol{b} \rangle \in \mathrm{lrf}(C)$, which is in turn equivalent, by definition, to $\boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{A}' \boldsymbol{x} - \boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{A}' \boldsymbol{x}' \geq k$ and $\boldsymbol{\lambda}_1^{\mathrm{T}} \boldsymbol{b} + \boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{A}' \boldsymbol{x} \geq 0$. We have

$$\begin{aligned}
\begin{pmatrix} \boldsymbol{A} & \boldsymbol{A}' \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{x}' \end{pmatrix} \leq \boldsymbol{b} &\implies \boldsymbol{A}\boldsymbol{x} + \boldsymbol{A}'\boldsymbol{x}' \leq \boldsymbol{b} \\
&\implies -\boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{A}'\boldsymbol{x}' - \boldsymbol{b} \\
&\implies -\boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{A}'\boldsymbol{x}' - \boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{b} && \text{by } \boldsymbol{\lambda}_2 \geq \boldsymbol{0} \\
&\implies \boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{A}'\boldsymbol{x} \geq \boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{A}'\boldsymbol{x}' - \boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{b} && \text{by (27c)}
\end{aligned}$$

and the former property is satisfied if we choose $k = -\boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{b}$, which is nonnegative by relation (27d). For the latter property, we have

$$\begin{aligned}
\boldsymbol{A}\boldsymbol{x} + \boldsymbol{A}'\boldsymbol{x}' \leq \boldsymbol{b} &\implies \boldsymbol{\lambda}_1^{\mathrm{T}} \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\lambda}_1^{\mathrm{T}} \boldsymbol{A}'\boldsymbol{x}' \leq \boldsymbol{\lambda}_1^{\mathrm{T}} \boldsymbol{b} && \text{as } \boldsymbol{\lambda}_1^{\mathrm{T}} \text{ is non-negative} \\
&\implies \boldsymbol{\lambda}_1^{\mathrm{T}} \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{\lambda}_1^{\mathrm{T}} \boldsymbol{b} && \text{by (27a)} \\
&\implies \boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{\lambda}_1^{\mathrm{T}} \boldsymbol{b} && \text{by (27b)} \\
&\implies -\boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{A}'\boldsymbol{x} \leq \boldsymbol{\lambda}_1^{\mathrm{T}} \boldsymbol{b} && \text{by (27c)}
\end{aligned}$$

and both properties are thus proved.

$\mathrm{pr}(C) \supseteq \widehat{\mathrm{ms}}(C)$. In order to prove the inverse containment, we will need to recall the affine form of Farkas' Lemma (see [27]).

Lemma 6.2 (Affine form of Farkas' lemma). *Let P be a nonempty polyhedron defined by the inequalities $\boldsymbol{C}\boldsymbol{x} + \boldsymbol{d} \geq \boldsymbol{0}$. Then an affine function $f(\boldsymbol{x})$ is non-negative everywhere in P if and only if it is a positive affine combination of the columns of $\boldsymbol{C}\boldsymbol{x} + \boldsymbol{d}$: $f(\boldsymbol{x}) = \lambda_0 + \boldsymbol{\lambda}^{\mathrm{T}}(\boldsymbol{C}\boldsymbol{x} + \boldsymbol{d})$ with $\lambda_0 \geq 0$, $\boldsymbol{\lambda} \geq \boldsymbol{0}$.*

Let $\tilde{\boldsymbol{\mu}} \in \widehat{\mathrm{ms}}(C)$. Then there exists $h \in \mathbb{Q}_+ \setminus \{0\}$ such that $h\tilde{\boldsymbol{\mu}} \in \mathrm{lrf}(C)$ describes a linear ranking function f for C.

The inequalities $\begin{pmatrix} \boldsymbol{A} & \boldsymbol{A}' \end{pmatrix} \langle \boldsymbol{x}, \boldsymbol{x}' \rangle \leq \boldsymbol{b}$ define a polyhedron; according to the affine form of Farkas' lemma, a function is non-negative on this polyhedron, i.e., throughout the loop, if and only if it is a positive affine combination of the column vectors $\begin{pmatrix} \boldsymbol{A} & \boldsymbol{A}' \end{pmatrix} \langle \boldsymbol{x}, \boldsymbol{x}' \rangle \leq \boldsymbol{b}$. In particular this holds for the ranking function f and its two properties: $f(\boldsymbol{x}) \geq 0$ and $f(\boldsymbol{x}) - f(\boldsymbol{x}') \geq 1$.

Hence there exist two non-negative rational vectors $\boldsymbol{\lambda}_1$ and $\boldsymbol{\lambda}_2$ and two non-negative numbers $\lambda_{0,1}$ and $\lambda_{0,2}$ such that

$$f(\boldsymbol{x}) = \lambda_{0,1} + \boldsymbol{\lambda}_1^{\mathrm{T}} \left(- \begin{pmatrix} \boldsymbol{A} & \boldsymbol{A}' \end{pmatrix} \langle \boldsymbol{x}, \boldsymbol{x}' \rangle + \boldsymbol{b} \right)$$

and

$$f(\boldsymbol{x}) - f(\boldsymbol{x}') - 1 = \lambda_{0,2} + \boldsymbol{\lambda}_2^{\mathrm{T}} \left(- \begin{pmatrix} \boldsymbol{A} & \boldsymbol{A}' \end{pmatrix} \langle \boldsymbol{x}, \boldsymbol{x}' \rangle + \boldsymbol{b} \right).$$

Replacing $f(\boldsymbol{x})$ by $h\boldsymbol{\mu x} + h\mu_0$, we get two equalities—one for the part containing variables and one for the remaining part—for each expression. After simplification we obtain the following equalities:

$$-\boldsymbol{\lambda}_1^{\mathrm{T}} \begin{pmatrix} \boldsymbol{A} & \boldsymbol{A}' \end{pmatrix} \langle \boldsymbol{x}, \boldsymbol{x}' \rangle = h\boldsymbol{\mu x} \tag{34a}$$

$$-\boldsymbol{\lambda}_2^{\mathrm{T}} \begin{pmatrix} \boldsymbol{A} & \boldsymbol{A}' \end{pmatrix} \langle \boldsymbol{x}, \boldsymbol{x}' \rangle = h\boldsymbol{\mu x} - h\boldsymbol{\mu x}' \tag{34b}$$

$$-\boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{b} = 1 + \lambda_{0,2} \tag{34c}$$

From (34a) and (34b) we obtain $\boldsymbol{\lambda}_1^{\mathrm{T}} \boldsymbol{A} = -h\boldsymbol{\mu}^{\mathrm{T}}$, $\boldsymbol{\lambda}_1^{\mathrm{T}} \boldsymbol{A}' = \boldsymbol{0}^{\mathrm{T}}$, $\boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{A} = -h\boldsymbol{\mu}^{\mathrm{T}}$ and $\boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{A}' = h\boldsymbol{\mu}^{\mathrm{T}}$. We can rewrite it as $\boldsymbol{0}^{\mathrm{T}} = \boldsymbol{\lambda}_1^{\mathrm{T}} \boldsymbol{A}' = (\boldsymbol{\lambda}_1^{\mathrm{T}} - \boldsymbol{\lambda}_2^{\mathrm{T}}) \boldsymbol{A} = \boldsymbol{\lambda}_2^{\mathrm{T}} (\boldsymbol{A} + \boldsymbol{A}')$. From (34c) we deduce $\boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{b} < 0$.

The four conditions (27) to prove termination by [7] are thus satisfied.

The combination of Theorems 4.4 and 6.1 gives:

Theorem 6.3. *Let C be the binary* CLP($\mathbb{Q}$) *clause $p(\bar{x}) :- c[\bar{x}, \bar{x}'], p(\bar{x}')$, where p is an n-ary predicate and $c[\bar{x}, \bar{x}']$ is a linear satisfiable constraint. Let $\widehat{\mathrm{lrf}}(C)$ be the set of (positive multiples of) linear ranking functions for C, $\widehat{\mathrm{ms}}(C)$ be the set of (positive multiples of) solutions of the Mesnard and Serebrenik system* (20) *projected onto μ and* $\mathrm{pr}(C)$ *be the set of the ranking function coefficients obtained through the method of Podelski and Rybalchenko, that is,*

$$\widehat{\mathrm{lrf}}(C) := \left\{ k\tilde{\boldsymbol{\mu}} \in \mathbb{Q}^{n+1} \;\middle|\; \begin{array}{l} \forall \bar{x}, \bar{x}' \in \mathbb{Q}^n : c[\bar{x}, \bar{x}'] \implies \\ \qquad \sum_{i=1}^{n} \mu_i x_i - \sum_{i=1}^{n} \mu_i x_i' \geq 1 \\ \qquad \wedge\, \mu_0 + \sum_{i=1}^{n} \mu_i x_i \geq 0, \\ k \in \mathbb{Q}_+ \setminus \{0\} \end{array} \right\},$$

$$\widehat{\mathrm{ms}}(C) := \left\{ k\tilde{\boldsymbol{\mu}} \in \mathbb{Q}^{n+1} \;\middle|\; \begin{array}{l} \langle \boldsymbol{y}, \boldsymbol{\mu} \rangle \textit{ and } \langle \boldsymbol{z}, \tilde{\boldsymbol{\mu}} \rangle \textit{ are solutions of } (20), \\ k \in \mathbb{Q}_+ \setminus \{0\} \end{array} \right\},$$

$$\mathrm{pr}(C) := \left\{ \langle \boldsymbol{\lambda}_2^{\mathrm{T}} \boldsymbol{A}', \boldsymbol{\lambda}_1^{\mathrm{T}} \boldsymbol{b} \rangle \in \mathbb{Q}^{n+1} \;\middle|\; \langle \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2 \rangle \textit{ is a solution of } (27) \right\},$$

where $c[\bar{x}, \bar{x}']$ is equivalent to $\begin{pmatrix} \boldsymbol{A} & \boldsymbol{A}' \end{pmatrix} \langle \boldsymbol{x}, \boldsymbol{x}' \rangle \leq \boldsymbol{b}$ or to $\boldsymbol{A}_c \langle \boldsymbol{x}, \boldsymbol{x}' \rangle \geq \boldsymbol{b}_c$, respectively. Then $\widehat{\mathrm{lrf}}(C) = \widehat{\mathrm{ms}}(C) = \mathrm{pr}(C)$.

6.2. Worst-Case Complexity Using the Simplex Algorithm

The computationally most expensive component in both methods is the resolution of a linear optimization problem that can always be expressed in the standard form

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{c}^{\mathrm{T}} \boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{A x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned}$$

by applying well known transformations: inequalities and *unconstrained* (i.e., not subject to lower or upper bounds) variables can be replaced and the resulting equivalent problem in standard form has one more variable for each inequality or unconstrained variable appearing in the original problem.

The most common way to solve this linear optimization problems involves using the simplex algorithm [33], an iterative algorithm that requires $\binom{e+u}{e}$ pivoting steps in the worst-case scenario, where e and u denote the number of equalities in $\boldsymbol{A}$ and unknowns in $\boldsymbol{x}$ respectively.

For a simple linear loop of m inequalities over n variables, Podelski and Rybalchenko require to solve a linear problem in standard form having $3n$ equalities over $2m$ variables (the opposite of the expression appearing in (27d) can be used as the quantity to be minimized); this gives a worst-case complexity of $\binom{3n+2m}{3n}$ pivoting steps, corresponding, by Stirling's formula, to an exponential complexity of exponent $3n+2m$ approximately.

If the alternative formalization of the Podelski and Rybalchenko method is adopted for the same loop, then we will have the same m constraints as above for the '$\bigstar$' invariant, while the '$\blacktriangledown$' invariant will be described by other ℓ constraints. If redundant constraints are removed, we will have $\ell \leq m$. Hence, the alternative approach will result in a linear programming problem having $2n$ equalities over $m+2\ell$ variables. Hence, the worst-case number of pivoting steps will be an exponential of exponent approximately $2n+m+2\ell$.

For the same simple linear loop, Serebrenik and Mesnard require the resolution of two linear problems, that can be rewritten to contain $2n$ equalities over $m+n$ variables (with n unconstrained variables) and $2n+1$ equalities over $(m+2)+(n+1)$ variables (with $n+1$ unconstrained variables), respectively. They can then be merged to generate a single linear problem of $4n+1$ equalities over $m+(m+2)+(n+1)$ variables, $n+1$ of which unconstrained, and an extra inequality replacing one of the two objective functions. In the end, we get a linear problem in standard form with $4n+2$ equalities over $2m+2n+5$ variables. This means a worst-case complexity of $\binom{6n+2m+7}{4n+2}$ pivoting steps—exponential complexity of exponent $6n+2m$ approximately.

So the method proposed by Podelski and Rybalchenko has, in general, a lower worst-case complexity than the one proposed by Mesnard and Serebrenik, if the single linear problem approach is chosen. The comparison of the two alternative implementation approaches for the Podelski and Rybalchenko method depends on the relations between quantities n, m and ℓ. On the one hand, if ℓ is significantly smaller than m, then the alternative approach could result in an efficiency improvement. On the other hand, if the number of constraints is much higher than the number of variables, then the original implementation approach should be preferred. Note that the need for two loop invariants instead of a single one should not be seen as a big practical problem: in fact, most analysis frameworks will provide the '$\blacktriangledown$' invariant as the original input to the termination analysis tool, which will then use it to compute the '$\bigstar$' invariant (via the abstract execution of a single iteration of the loop); that is, the computational cost for the '$\blacktriangledown$' invariant is implicitly paid anyway.

It is well known, though, that the worst-case scenario for the simplex algorithm is extremely uncommon in practice. An average complexity analysis and, more recently, a smoothed complexity analysis [34] have been carried out on the simplex algorithm and showed why it usually takes polynomial time. Besides the theoretical studies, several experimental evaluations of implementations of the simplex algorithm reported that the average number of pivoting steps seems to grow linearly with the sum $e + u$ of the number of equalities and unknowns of the problem. Therefore, for a more informative and meaningful comparison, the next section presents an experimental evaluation of the methods on a representative set of while loops.

7. Implementation and Experimental Evaluation

The *Parma Polyhedra Library* (PPL) is a free software, professional library for the handling of numeric approximations targeted at static analysis and computer-aided verification of hardware and software systems [5].[15] The PPL is employed by numerous projects in this field, most notably by GCC, the *GNU Compiler Collection*, probably the compilers' suite more in widespread use.

As an integral part of the overall project to which the present paper belongs —whose aim is to make the technology of the automatic synthesis of linear ranking functions thoroughly explained and generally available—, we have extended the PPL with all the methods discussed in the present paper. Previously, only a rather limited demo version of *RankFinder* was available, only in x86/Linux binary format, implementing the method by Podelski and Rybalchenko.[16] In contrast, the PPL implementation is completely general and available, both in source and binary formats, with high-level interfaces to C, C++, Java, OCaml and six different Prolog systems.

For each of the methods —Mesnard and Serebrenik (MS) or Podelski and Rybalchenko (PR)—, for each of the two possibilities to encode the input — either the single $\bigstar$ invariant of (22) in Section 4.3, or the two $\blacktriangledown$ and $\bigstar$ invariants of (23) in Section 5—, for each numerical abstractions supported by the PPL —including (not necessarily closed) convex polyhedra, bounded-difference shapes and octagonal shapes—, the PPL provides three distinct functionalities to investigate termination of the loop being analyzed:

1. a Boolean termination test;
2. a Boolean termination test that, in addition, returns the coefficients of one (not further specified) affine ranking function;
3. a function returning a convex polyhedron that encodes the space of all affine ranking functions.

In addition, using the MS method and for each input method, the PPL provides

4. a function returning two convex polyhedra that encode the space of all decreasing functions and all bounded functions, respectively, for use in conditional termination analysis.

We have evaluated the performance of the new algorithms implemented in the PPL using the termination analyzer built into *Julia*, a state-of-the-art analyzer for Java bytecode [35]. We have thus taken several Java programs in the

[15] See `http://www.cs.unipr.it/ppl/` for more information.
[16] See `http://www7.in.tum.de/~rybal/rankfinder/`, last checked on March 27th, 2010.

Julia test suite and, using Julia, we have extracted the constraint systems that characterize the loops in the program that Julia cannot quickly resolve with syntax-based heuristics. This extraction phase allowed us to measure the performance of the methods described in the present paper, factoring out the time spent by Julia in all the analyses (nullness, sharing, path-length, unfolding, ...) that allow to obtain such constraint systems.

We first tested the performance (and correctness) of the new PPL implementation with the implementation of the MS method, based on CLP(Q), previously used by Julia and with the implementation of PR, still based on CLP(Q), provided by the demo version of RankFinder. The reason we did this comparison is that, while we know that the infinite precision implementation of the simplex algorithm available in the PPL performs better than its direct competitors [5, Section 4, Table 3],[17] we know there is much room for improvement: it could have been the case that the constraint solver employed in modern CLP systems made our implementation useless. The result was quite satisfactory: the PPL implementation is one to two orders of magnitude faster over the considered benchmark suite.

The benchmark programs are: `CaffeineMark`, from Pendragon Software Corporation, measures the speed of Java; `JLex` is a lexical analyzer generator developed by Elliot Berk and C. Scott Ananian; `JavaCC` is a parser generator from Sun Microsystems; `Java_CUP` is a parser generator developed by Scott Hudson, Frank Flannery and C. Scott Ananian; `Jess` is a rule engine written by Ernest Friedman-Hill; `Kitten` is a didactic compiler for a simple imperative object-oriented language written by Fausto Spoto; `NQueens` is a solver of the n-queens problem which includes a library for binary decision diagrams; `Raytracer` is a ray-tracing program; `Termination` is a JAR file containing all the programs of [35, Figure 16]. In Table 1 we report, for each benchmark, the number of loops for which termination was investigated, the interval, mean and standard deviations —with two significant figures— of the quantities n (number of variables) and m (number of constraints) that characterize those loops.

The results of the CPU-time comparison between the MS and PR methods are reported in Table 2. Measurements took place on a GNU/Linux system equipped with an Intel Core 2 Quad CPU Q9400 at 2.66 GHz and 8 Gbytes of main memory; a single core was used and the maximum resident set size over the entire set of tests was slightly above 53 Mbytes. From these we can conclude that the difference in performance between the two methods is rather limited. The PR method is more efficient on the problem of semi-deciding termination, with or without the computation of a witness ranking function, while the MS method is superior on the problem of computing the space of all affine ranking functions.

[17] I.e., *Cassowary* (`http://www.cs.washington.edu/research/constraints/cassowary/`) and *Wallaroo* (`http://sourceforge.net/projects/wallaroo/`). While GLPK, the *GNU Linear Programming Toolkit* (`http://www.gnu.org/software/glpk/`) includes a solver that is termed "exact," it still depends critically on floating point computations; moreover, it has not yet been made available in the public interface.

Table 1: Benchmarks used in the experimental evaluation

benchmark	loops	n	$\overline{n}$	σ_n	m	$\overline{m}$	σ_m
`CaffeineMark`	151	[1,9]	6.0	1.3	[2,26]	17.	3.8
`JLex`	467	[1,14]	7.2	2.5	[2,45]	17.	6.7
`JavaCC`	136	[1,14]	8.6	4.1	[1,45]	22.	12.
`Java_CUP`	29	[2,14]	8.3	4.3	[5,45]	23.	13.
`Jess`	151	[1,9]	6.0	1.3	[2,26]	17.	3.8
`Kitten`	1484	[1,15]	11.	3.6	[2,45]	29.	10.
`NQueens`	359	[1,14]	6.3	3.6	[2,45]	17.	10.
`Raytracer`	8	[2,9]	4.5	2.7	[5,26]	11.	7.8
`Termination`	121	[1,9]	4.2	3.5	[2,27]	12.	9.9

Table 2: MS vs PR: CPU time in seconds

	term. test		one r. f.		all r. f.	
benchmark	MS	PR	MS	PR	MS	PR
`CaffeineMark`	0.42	0.26	0.43	0.25	0.31	0.34
`JLex`	1.62	0.83	1.64	0.84	1.17	1.14
`JavaCC`	0.86	0.43	0.87	0.45	0.67	0.65
`Java_CUP`	0.35	0.14	0.35	0.14	0.29	0.22
`Jess`	0.42	0.26	0.43	0.26	0.29	0.34
`Kitten`	11.8	6.87	11.9	6.84	8.41	10.2
`NQueens`	1.43	0.76	1.44	0.74	0.99	1.03
`Raytracer`	0.04	0.03	0.04	0.03	0.03	0.03
`Termination`	0.25	0.15	0.25	0.15	0.18	0.21

Table 3: Precision results and application to conditional termination

benchmark	loops	term	w/ d.f.	w/o d.f.
`CaffeineMark`	151	149	0	2
`JLex`	467	453	3	11
`JavaCC`	136	120	4	12
`Java_CUP`	29	27	0	2
`Jess`	151	149	0	2
`Kitten`	1484	1454	3	27
`NQueens`	359	271	4	84
`Raytracer`	8	6	0	2
`Termination`	121	119	0	2

We also present, in Table 3, the precision results. For each benchmark, along with the total number of loops, we have the number of loops for which termination is decided positively, either with the MS or the PR method (column 'term'); the remaining loops are divided, using the MS method, between those that admit a linear decreasing function (column 'w/ d.f.') and those who do not (column 'w/o d.f.'). It can be seen that the percentage of loops for which termination is decided positively ranges from 75% to 99%, depending on the benchmark. This means that we are conducting the experimental evaluation with a termination analyzer, Julia, whose analysis algorithms —though certainly improvable— very often provide enough information for termination analysis. This is crucial for the meaningfulness of the experimental evaluation presented in this section.

8. Conclusions

Linear ranking functions play a crucial role in termination analysis, as the termination of many programs can be decided by the existence on one such function. In this paper we have addressed the topic of the automatic synthesis of linear ranking functions with the aim of clarifying its origins, thoroughly explaining the underlying theory, and presenting new, efficient implementations that are begin made available to the general public.

In particular, we have introduced, in general terms independent from any programming paradigm, the problem of automatic termination analysis of individual loops —to which more general control flows can be reconducted— and its solution technique based on the synthesis of ranking functions.

We have then presented and generalized a technique. originally due to Sohn and Van Gelder, that was virtually unknown outside the logic programming field despite its general applicability and its relative completeness (given a linear constraint system approximating the behavior of a loop, if a linear ranking function exists for that system, then the method will find it). This method, due to its ability to characterize the spaces of all the linear decreasing functions and all the linear bounded functions, is also immediately applicable to *conditional* termination analysis; this theme is an excellent candidate for future work.

We have also presented and, for the first time, fully justified, a more recent technique by Podelski and Rybalchenko. For this we also present an alternative formulation that can lead to efficiency improvements.

We have compared the two methods, first proving their equivalence — thus obtaining an independent confirmation on their correctness and relative completeness— and then studying their worst-case complexity.

Finally, we have presented the implementation of all the techniques described in the paper recently included in the Parma Polyhedra Library, along with an experimental evaluation covering both the efficiency and the precision of the analysis.

Acknowledgments

We would like to express our gratitude to David Merchat and Alessandro Zaccagnini for their useful comments and suggestions. The connection between the MS method and conditional termination analysis was indicated to us during a discussion with Samir Genaim.

References

[1] J. C. Lagarias, The $3x+1$ problem and its generalizations, American Mathematical Monthly 92 (1985) 3–23.

[2] E. W. Dijkstra, A Discipline of Programming, Prentice Hall, 1976.

[3] P. Cousot, R. Cousot, 'A la Floyd' induction principles for proving inevitability properties of programs, in: M. Nivat, J. C. Reynolds (Eds.), Algebraic Methods in Semantics, Cambridge University Press, 1985, pp. 277–312.

[4] R. W. Floyd, Assigning meanings to programs, in: J. T. Schwartz (Ed.), Mathematical Aspects of Computer Science, Vol. 19 of Proceedings of Symposia in Applied Mathematics, American Mathematical Society, Providence RI, New York City, USA, 1967, pp. 19–32.

[5] R. Bagnara, P. M. Hill, E. Zaffanella, The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems, Science of Computer Programming 72 (1–2) (2008) 3–21.

[6] K. Sohn, A. Van Gelder, Termination detection in logic programs using argument sizes (extended abstract), in: Proceedings of the Tenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, ACM, Association for Computing Machinery, Denver, Colorado, United States, 1991, pp. 216–226.

[7] A. Podelski, A. Rybalchenko, A complete method for the synthesis of linear ranking functions, in: B. Steffen, G. Levi (Eds.), Verification, Model Checking and Abstract Interpretation: Proceedings of the 5th International Conference, VMCAI 2004, Vol. 2937 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, Venice, Italy, 2004, pp. 239–251.

[8] R. Bagnara, F. Mesnard, A. Pescetti, E. Zaffanella, The automatic synthesis of linear ranking functions: The complete unabridged version, Quaderno 498, Dipartimento di Matematica, Università di Parma, Italy, available at http://www.cs.unipr.it/Publications/ (2010). Also published as arXiv:cs.PL/1004.0944, available from http://arxiv.org/.

[9] M. A. Colón, H. B. Sipma, Practical methods for proving program termination, in: E. Brinksma, K. G. Larsen (Eds.), Computer Aided Verification: Proceedings of the 14th International Conference (CAV 2002), Vol. 2404 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, Copenhagen, Denmark, 2002, pp. 442–454.

[10] K. Sohn, Automated termination analysis for logic programs, Ph.D. thesis, University of California Santa Cruz, Santa Cruz, CA, USA (1993).

[11] J. Fischer, Termination analysis for Mercury using convex constraints, Honours report, Department of Computer Science and Software Engineering, The University of Melbourne, Melbourne, Australia (Aug. 2002).

[12] C. Speirs, Z. Somogyi, H. Søndergaard, Termination analysis for Mercury, in: Van Hentenryck [36], pp. 157–171.

[13] A. R. Bradley, Z. Manna, H. B. Sipma, Linear ranking with reachability, in: Computer Aided Verification: Proceedings of the 15th International Conference, Vol. 3576 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, Edinburgh, Scotland, UK, 2005, pp. 491–504.

[14] K. Verschaetse, D. De Schreye, Deriving termination proofs for logic programs, using abstract procedures, in: K. Furukawa (Ed.), Proceedings of the 8th International Conference on Logic Programming, The MIT Press, Paris, France, 1991, pp. 301–315.

[15] M. Codish, C. Taboch, A semantic basis for the termination analysis of logic programs, Journal of Logic Programming 41 (1) (1999) 103–123.

[16] F. Mesnard, A. Serebrenik, A polynomial-time decidable class of terminating binary constraint logic programs, Tech. Rep. 05-11, Université de la Réunion (2005).

[17] F. Mesnard, A. Serebrenik, Recurrence with affine level mappings is P-time decidable for CLP(R), Theory and Practice of Logic Programming 8 (1) (2008) 111–119.

[18] M. T. Nguyen, D. De Schreye, Polynomial interpretations as a basis for termination analysis of logic programs, in: M. Gabbrielli, G. Gupta (Eds.), Proceedings of the 21st International Conference on Logic Programming, no. 3668 in Lecture Notes in Computer Science, Springer-Verlag, Berlin, Sitges, Spain, 2005, pp. 311–325.

[19] D. S. Lankford, A finite termination algorithm, Internal memo, Southwestern University, Georgetown, TX, USA (1976).

[20] H. Zantema, Termination of term rewriting, Tech. Rep. UU-CS-2000-04, Department of Computer Science, Universiteit Utrecht, Utrecht, The Netherlands (2000).

[21] P. Cousot, Proving program invariance and termination by parametric abstraction, lagrangian relaxation and semidefinite programming, in: Cousot [37], pp. 1–24.

[22] M. A. Colón, H. B. Sipma, Synthesis of linear ranking functions, in: T. Margaria, W. Yi (Eds.), Tools and Algorithms for Construction and Analysis of Systems, 7th International Conference, TACAS 2001, Vol. 2031 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, Genova, Italy, 2001, pp. 67–81.

[23] C. S. Lee, N. D. Jones, A. M. Ben-Amram, The size-change principle for program termination, in: C. Norris, J. J. B. Fenwick (Eds.), Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2001), Vol. 36 of ACM SIGPLAN Notices, ACM Press, London, UK, 2001, pp. 81–92.

[24] N. Dershowitz, N. Lindenstrauss, Y. Sagiv, A. Serebrenik, A general framework for automatic termination analysis of logic programs, Appl. Algebra Eng. Commun. Comput. 12 (1/2) (2001) 117–156.

[25] M. Codish, V. Lagoon, P. Stuckey, Testing for termination with monotonicity constraints, in: M. Gabbrielli, G. Gupta (Eds.), Twenty First International Conference on Logic Programming, Vol. 3668 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, Sitges, Spain, 2005, pp. 326–340.

[26] M. R. Garey, D. S. Johnson, Computers and Intractability; A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., New York, NY, 1990.

[27] A. Schrijver, Theory of linear and integer programming, Wiley, Chichester, New York, 1986.

[28] P. Cousot, N. Halbwachs, Automatic discovery of linear restraints among variables of a program, in: Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages, ACM Press, Tucson, Arizona, 1978, pp. 84–96.

[29] B. Cook, S. Gulwani, T. Lev-Ami, A. Rybalchenko, M. Sagiv, Proving conditional termination, in: A. Gupta, S. Malik (Eds.), Computer Aided Verification: Proceedings of the 20th International Conference (CAV 2008), Vol. 5123 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, Princeton, NJ, USA, 2008, pp. 328–340.

[30] A. Rybalchenko, Temporal verification with transition invariants, Ph.D. thesis, Universität des Saarlandes, Saarbrücken, Germany (2004).

[31] A. Serebrenik, F. Mesnard, On termination of binary CLP programs, in: S. Etalle (Ed.), Logic Based Program Synthesis and Transformation: 14th International Symposium, Revised Selected Papers, no. 3573 in Lecture Notes in Computer Science, Springer-Verlag, Berlin, Verona, Italy, 2005, pp. 231–244.

[32] F. Mesnard, Inferring left-terminating classes of queries for constraint logic programs by means of approximations, in: M. J. Maher (Ed.), Logic Programming: Proceedings of the Joint International Conference and Symposium on Logic Programming, MIT Press Series in Logic Programming, The MIT Press, Bonn, Germany, 1996, pp. 7–21.

[33] C. H. Papadimitriou, K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice Hall, Englewood Cliffs, NJ, USA, 1982.

[34] D. A. Spielman, S.-H. Teng, Smoothed analysis: Why the simplex algorithm usually takes polynomial time, Journal of the ACM 51 (2004) 385–463.

[35] F. Spoto, F. Mesnard, É. Payet, A termination analyzer for Java bytecode based on path-length, ACM Transactions on Programming Languages and Systems 32 (3).

[36] P. Van Hentenryck (Ed.), Static Analysis: Proceedings of the 4th International Symposium, Vol. 1302 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, Paris, France, 1997.

[37] R. Cousot (Ed.), Verification, Model Checking and Abstract Interpretation: Proceedings of the 6th International Conference (VMCAI 2005), Vol. 3385 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, Paris, France, 2005.