Termination Analysis of Linear-Constraint Programs

Suggested Citation: Amir M. Ben-Amram, Samir Genaim, Joël Ouaknine and James Worrell (2024), "Termination Analysis of Linear-Constraint Programs", : Vol. xx, No. xx, pp 1–18. DOI: 10.1561/XXXXXXXXX.

Amir M. Ben-Amram
Qiryat Ono, Israel
benamram.amir@gmail.com

Samir Genaim
Complutense University of Madrid, Spain
sgenaim@ucm.es

Joël Ouaknine
Max Planck Institute for Software Systems, Saarland, Germany
joel@mpi-sws.org

James Worrell
University of Oxford, UK
jbw@cs.ox.ac.uk

now
the essence of knowledge
Boston — Delft

Contents

Termination Analysis of Linear-Constraint Programs

Amir M. Ben-Amram[1], Samir Genaim[2], Joël Ouaknine[3] and James Worrell[4]

[1] *Qiryat Ono, Israel; benamram.amir@gmail.com*

[2] *Complutense University of Madrid, Spain; sgenaim@ucm.es*

[3] *Max Planck Institute for Software Systems, Saarland, Germany; joel@mpi-sws.org*

[4] *University of Oxford, UK; jbw@cs.ox.ac.uk*

ABSTRACT

This survey provides an overview of techniques in termination analysis for programs with numerical variables and transitions defined by linear constraints. This subarea of program analysis is challenging due to the existence of undecidable problems, and this survey systematically explores approaches that mitigate this inherent difficulty. These include foundational decidability results, the use of ranking functions, and disjunctive well-founded transition invariants. The survey also discusses non-termination witnesses, used to prove that a program will not halt. We examine the algorithmic and complexity aspects of these methods, showing how different approaches offer a trade-off between expressive power and computational complexity. The survey does not discuss how termination analysis is performed on real-world programming languages, nor does it consider more expressive abstract models that include non-linear arithmetic, probabilistic choice, or term rewriting systems.

Amir M. Ben-Amram, Samir Genaim, Joël Ouaknine and James Worrell (2024), "Termination Analysis of Linear-Constraint Programs", : Vol. xx, No. xx, pp 1–18.
DOI: 10.1561/XXXXXXXXX.

1

Introduction

Proving termination is a basic building block of establishing program correctness, or analysing the behaviour of systems modelled by programs. The topic of this survey is the termination problem for programs with numerical variables (storing integers, rationals, or reals) whose transitions are specified by linear equations and inequalities. To make this notion concrete, here is an example of a loop whose termination we may want to prove:

```
while (x2-x1<=0 && x1+x2>=1) x2=x2-2*x1+1;
```

While this loop is written in C syntax, we prefer to abstract from any particular programming language and model the loop body as a relation between values x_1, x_2 of the program variables before its execution and their values x'_1, x'_2 after its execution. We thus express the above loop as:

$$\textit{while}\ (x_2 - x_1 \le 0, x_1 + x_2 \ge 1)\ \textit{do}\ x'_2 = x_2 - 2x_1 + 1, x'_1 = x_1\,.$$

This, more mathematical, expression generalises easily by allowing inequalities as well as equations in the specification of the “loop body”, for example we might consider

$$\textit{while}\ (x_2 - x_1 \le 0, x_1 + x_2 \ge 1)\ \textit{do}\ x'_2 = x_2 - 2x_1 + 1, x'_1 \le x_1\,.$$

This is what we call a *simple loop*, or a *single-path loop*. Note that such a loop is, in general, non-deterministic. In the above example, in any execution of the loop body any value of x_1' that satisfies the constraint may be chosen. We will also consider *multi-path loops*, that model branching in the loop body, so that the iteration is represented by several alternatives, each one with its set of constraints; and the most general form, a *control-flow graph* which can represent a branching structure, nested loops *etc*. We sometimes group all these types under the heading *linear-constraint programs*.

Where do such termination problems come from? As stated before, the main motivation is program analysis. In many programs the variables whose behaviour is relevant to program termination are numerical, and in this case the program can be often faithfully modelled by linear-constraint programs, possibly abstracting away operations that are not relevant to termination. Our model is also abstract in the sense that we consider the domain of variables to be either $\mathbb{Z}$, $\mathbb{Q}$, or $\mathbb{R}$ — we do not model the finite universe of machine integers, or the finite precision of floating-point numbers.

There are, of course, computer programs that manipulate non-numerical data; but in many such programs the proof of termination relies on numbers related to these data — for example the length of lists constructed or consumed by the program. Thus several tools for testing the termination of programs abstract structured values into numbers and in essence reduce the problem to the analysis of numerical programs.

The termination of numerical programs defined by linear constraints is a challenging area, since it includes undecidable problems—so it is important to break the area into subproblems, and attempt to understand the decidability and complexity of each subproblem. In Section 3 we provide the complete solution for one subproblem, the termination of simple loops whose body is a linear transformation (thus defined by linear equations and not inequalities). We also present a couple of results that illustrate the limitations of decidability in the termination analysis of programs of the kind we consider, namely sub-classes of programs for which termination is undecidable.

Other subproblems arise by weakening the goal from determining termination *tout court*, to that of determining whether termination can be established by a specific method. The best-known example is the principle of *ranking program states*: if we can associate with each program state a *rank* such that ranks are bound to decrease during computation (but can not decrease forever, *e.g.*, because they are natural numbers), then the program terminates. When we fix the set of admissible functions for ranking states (the so-called *termination witnesses*), we get a well-defined subproblem of the termination problem that may well be solvable, and in fact this is one of the approaches extensively used by termination tools. In Section 4 we survey algorithmic results for ranking-function problems, specifically we consider *linear* ranking functions and *lexicographic-linear* ranking functions. In Section 5 we consider the *disjunctive transition invariant* technique, which breaks the termination proof for a program into multiple sub-proofs, intuitively for different cycles in the program. This technique is too general to allow for a complete solution for all types of programs, but we survey classes of programs for which it is both known that the technique is sufficient to prove termination, and there are effective techniques of implementing it.

Just as there are witnesses that ensure termination, there are also witnesses to non-termination: a trivial example is a state that is repeated. We discuss certain more involved non-termination witnesses in Section 6.

Termination analysis of programs is a broad field and this survey is necessarily limited in scope. In particular, we leave out all discussion of how termination analysis is done in actual programming languages and how the abstract programs we are dealing with are extracted from real code. We leave out certain more expressive abstract models, encompassing for example non-linear arithmetic or probabilistic choice. Furthermore, we do not discuss termination analysis of term rewriting systems, a field that has generated a considerable amount of research. The results we present attempt to show the state of the art for the subproblems we consider—giving complete solutions wherever possible, leaving out partial solutions and heuristic techniques, that may have their own merits. We also focus on presenting algorithms, examples and complexity results, rather than on giving proofs. The latter can be found in the given references. Throughout the survey, we also list 13 *open problems* that may be the subject of further research.

Organisation of this Survey. Section 2 provides the necessary mathematical background and defines the programs we use. The other chapters are independent of each other and can be read in any order, except for Section 5 that has some dependence on Section 4. Section 3 overviews results on the decidability and undecidability of termination for linear-constraint programs, and is mostly dedicated to the decidability of termination of so-called linear loops. Section 4 discusses ranking functions. We then overview works on disjunctive well-founded invariants in Section 5 and witnesses for non-termination in Section 6. Section 7 concludes the discussion.

2

Preliminaries

This section provides the mathematical background (Section 2.1), overviews definitions related to polyhedra and linear programming (Section 2.2), and defines the programs (Section 2.3) we use in this survey.

2.1 Mathematical Background

This section provides the mathematical background used throughout the survey.

2.1.1 Notations

For a set A, $x \in A$ means that x is an element of A, and $x \notin A$ means that x is not an element of A. The empty set is denoted by $\emptyset$. The cardinality of a set A, denoted by $|A|$, is the number of elements in A. For sets A and B, $A \subseteq B$ means that A is a subset of B, $A \subset B$ means that means that A is a strict subset of B, $A \cup B$ is their union, $A \cap B$ is their intersection, and $A \setminus B$ is their difference. The Cartesian product of two sets A and B, denoted by $A \times B$, is the set of all ordered pairs (a, b) where $a \in A$ and $b \in B$. The nth Cartesian power of A is $A^n = A \times \cdots \times A$ (n times).

The set of real, rational, integer and non-negative integer numbers are denoted respectively by $\mathbb{R}$, $\mathbb{Q}$, $\mathbb{Z}$, and $\mathbb{N}$. Note that some literature uses $\mathbb{N}$ to denote the set of positive integers. We also use $\mathbb{R}_{\mathbb{A}}$ to denote the set of real algebraic numbers. For $R \in \{\mathbb{R}, \mathbb{Q}, \mathbb{Z}, \mathbb{N}\}$, we use $R_{\geq 0}$ for the corresponding subset of non-negative values. We use $\vec{x} = (x_1, \ldots, x_n)$, where $x_i \in R$, to represent a column row vector, and $\boldsymbol{x} = (x_1, \ldots, x_n)^\top$ to represent a column vector. The elements of R^n are column vectors, however, abusing notation we might write $\boldsymbol{x} \in R^n$ or $\vec{x} \in R^n$. The set of complex number is denoted by $\mathbb{C}$. For $c = a + bi \in \mathbb{C}$, we use $\bar{c} = a - bi$ for its complex conjugate. A complex number c is said to be a root of the unity if $c^n = 1$ for some integer $n > 0$.

2.1.2 Eigenvectors and Eigenvalues

Used in Section 3

For a given square matrix $A \in \mathbb{C}^{n \times n}$, a non-zero vector $\boldsymbol{v}$ is an eigenvector if it satisfies the relationship $A\boldsymbol{v} = \lambda \boldsymbol{v}$, where λ is a scalar known as the eigenvalue corresponding to $\boldsymbol{v}$. The eigenvalues of a matrix are the roots of its characteristic polynomial, $\det(A - \lambda I) = 0$, where I is the identity matrix. Note that the eigenvalues may be complex numbers even if all entries of A are real numbers. The number of times an eigenvalue λ is a root of the characteristic polynomial is called its *algebraic multiplicity*. The concepts of eigenvalues and eigenvectors are essential for a wide range of applications, including stability analysis of dynamical systems and termination analysis.

2.1.3 Exponential Polynomials

Used in Section 3

Let $\lambda_1, \ldots, \lambda_m \in \mathbb{C}$ be distinct complex numbers and $e_1, \ldots, e_m$ positive integers. Then the family of *exponential-polynomial* functions $p_{i,j} : \mathbb{N} \to \mathbb{C}$, for $j \in \{1, \ldots, m\}$ and $i \in \{0, \ldots, e_j - 1\}$, given by $p_{i,j}(n) = \binom{n}{i}\lambda_j^n$ is linearly independent over $\mathbb{C}$. Moreover if $p : \mathbb{N} \to \mathbb{C}$ is a $\mathbb{C}$-linear combination of the $p_{i,j}$, then p is identically zero if and only if $p(n) = 0$ for $e_1 + \cdots + e_m$ consecutive values $n \in \mathbb{N}$.

Both of the above facts can be proved using generalised Vandermonde determinants (Halava *et al.*, 2005, Proposition 2.11).

2.1.4 Convexity

Used in sections 3–4

The *affine hull* of $S \subseteq \mathbb{R}^n$ is the smallest affine set that contains S, where an affine set is the translation of a vector subspace of $\mathbb{R}^n$. The affine hull of S can be characterised as follows:

$$\texttt{aff}(S) := \left\{ \sum_{i=1}^{k} \alpha_i \boldsymbol{x}_i \mid k > 0, \boldsymbol{x}_i \in S, \alpha_i \in \mathbb{R}, \sum_{i=1}^{k} \alpha_i = 1 \right\} .$$

The *convex hull* of $S \subseteq \mathbb{R}^n$ is the smallest convex set that contains S. The convex hull of S can be characterised as follows:

$$\texttt{conv}(S) := \left\{ \sum_{i=1}^{k} \alpha_i \boldsymbol{x}_i \mid k > 0, \boldsymbol{x}_i \in S, \alpha_i \in \mathbb{R}_{\geq 0}, \sum_{i=1}^{k} \alpha_i = 1 \right\} .$$

Clearly $\texttt{conv}(S) \subseteq \texttt{aff}(S)$.

The *relative interior* of a convex set $S \subseteq \mathbb{R}^n$ is its interior wrt. the restriction of the Euclidean topology to $\texttt{aff}(S)$. For example, the relative interior of a line segment in three dimensions is the line segment minus its endpoints. We have the following easy proposition, characterising the relative interior.

Proposition 2.1. Let $S = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\} \subseteq \mathbb{R}^n$. Then $\boldsymbol{u}$ lies in the relative interior of $\texttt{conv}(S)$ if and only if there exist $\alpha_1, \ldots, \alpha_n > 0$ such that $u = \sum_{i=1}^{n} \alpha_i \boldsymbol{a}_i$ and $\sum_{i=1}^{n} \alpha_i = 1$.

The *conic hull* of $S \subseteq \mathbb{R}^n$ is the smallest conic set that contains S. The conic hull of S can be characterised as follows:

$$\texttt{cone}(S) := \left\{ \sum_{i=1}^{k} \alpha_i \boldsymbol{x}_i \mid k > 0, \boldsymbol{x}_i \in S, \alpha_i \in \mathbb{R}_{\geq 0} \} \right\} .$$

2.1.5 Lattices

Used in Section 3

A *lattice of rank* r in $\mathbb{R}^n$ is a set

$$\Lambda := \{z_1\boldsymbol{v}_1 + \cdots + z_r\boldsymbol{v}_r \mid z_1, \ldots, z_r \in \mathbb{Z}\},$$

where $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_r$ are linearly independent vectors in $\mathbb{R}^n$. Given a convex set $C \subseteq \mathbb{R}^n$, define the *width* of C along a vector $\boldsymbol{u} \in \mathbb{R}^n$ to be

$$\sup\{\boldsymbol{u}^\top(\boldsymbol{x} - \boldsymbol{y}) \mid \boldsymbol{x}, \boldsymbol{y} \in C\}.$$

Furthermore the *lattice width* of C is the infimum over all non-zero vectors $\boldsymbol{u} \in \Lambda$ of the width of C along $\boldsymbol{u}$.

The following result (Banaszczyk *et al.*, 1999; Khinchin, 1948) captures the intuition that a convex set that contains no lattice point in its interior must be "thin" in some direction.

Theorem 2.1 (Flatness Theorem). Given a full-rank lattice Λ in $\mathbb{R}^n$, there exists W such that any convex set $C \subseteq \mathbb{R}^n$ that has non-empty interior and lattice width at least W contains a lattice point in its interior.

Recall that $C \subseteq \mathbb{R}^n$ is said to be *semi-algebraic* if it is definable by a boolean combination of polynomial constraints $p(x_1, \ldots, x_n) > 0$, where $p \in \mathbb{Z}[x_1, \ldots, x_n]$.

Theorem 2.2 (Khachiyan and Porkolab (Din and Zhi, 2010; Khachiyan and Porkolab, 1997)). It is decidable whether a given convex semi-algebraic set $C \subseteq \mathbb{R}^n$ contains an integer point, that is, whether $C \cap \mathbb{Z}^n \neq \emptyset$ and whether it contains a rational point, that is, whether $C \cap \mathbb{Q}^n \neq \emptyset$.

2.1.6 Multiplicative Relations

Used in Section 3

Next we introduce some concepts concerning groups of multiplicative relations among algebraic numbers.

Let $\mathbb{T} = \{z \in \mathbb{C} : |z| = 1\}$. We define the s-dimensional torus to be $\mathbb{T}^s$, considered as a group under component-wise multiplication. Given a tuple of algebraic numbers $\boldsymbol{\gamma} = (\gamma_1, \cdots, \gamma_s) \in \mathbb{T}^s$, the orbit $\{\boldsymbol{\gamma}^n : n \in \mathbb{N}\}$, where $\boldsymbol{\gamma}^n$ is defined to be $(\gamma_1^n, \ldots, \gamma_s^n)$, is a subset of $\mathbb{T}^s$.

In the following we characterise the topological closure of the orbit as an algebraic subset of $\mathbb{T}^s$.

The *group of multiplicative relations* of $\boldsymbol{\gamma} \in \mathbb{T}^s$ is defined as the following additive subgroup of $\mathbb{Z}^s$:

$$L(\boldsymbol{\gamma}) = \{\boldsymbol{v} \in \mathbb{Z}^s : \boldsymbol{\gamma}^{\boldsymbol{v}} = 1\},$$

where $\boldsymbol{\gamma}^{\boldsymbol{v}}$ is defined to be $\gamma_1^{v_1} \cdots \gamma_s^{v_s}$ for $\boldsymbol{v} \in \mathbb{Z}^s$, that is, exponentiation acts coordinate-wise. Since $L(\boldsymbol{\gamma})$ is a subgroup of $\mathbb{Z}^s$, it is a free Abelian group and hence has a finite basis. The following powerful theorem of Masser (Masser, 1988) gives bounds on the magnitude of the components of such a basis in terms of the heights and degrees of the γ_i.[1]

Theorem 2.3 (Masser)**.** The free Abelian group $L(\boldsymbol{\gamma})$ has a basis $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_l \in \mathbb{Z}^s$ for which

$$\max_{1\leq i\leq l, 1\leq j\leq s} |v_{i,j}| \leq (D \log H)^{O(s^2)},$$

where H and D bound respectively the heights and degrees of all the γ_i.

Membership of a tuple $\boldsymbol{v} \in \mathbb{Z}^s$ in $L(\boldsymbol{\gamma})$ can be computed in polynomial time, using exponentiation by squaring method. In combination with Theorem 2.3, it follows that we can compute a basis for $L(\boldsymbol{\gamma})$ in polynomial space by brute-force search.

Corresponding to $L(\boldsymbol{\gamma})$, we consider the following multiplicative subgroup of $\mathbb{T}^s$:

$$T(\boldsymbol{\gamma}) = \{\boldsymbol{\mu} \in \mathbb{T}^s : \forall \boldsymbol{v} \in L(\boldsymbol{\gamma}), \boldsymbol{\mu}^{\boldsymbol{v}} = 1\}.$$

If $\mathcal{B}$ is a basis of $L(\boldsymbol{\gamma})$, we can equivalently characterise $T(\boldsymbol{\gamma})$ as $\{\boldsymbol{\mu} \in \mathbb{T}^s : \forall \boldsymbol{v} \in \mathcal{B}, \boldsymbol{\mu}^{\boldsymbol{v}} = 1\}$. Crucially, this finitary characterisation allows us to represent $T(\boldsymbol{\gamma})$ as an algebraic set in $\mathbb{T}^s$.

We will use the following classical lemma of Kronecker on simultaneous Diophantine approximation to show that the orbit $\{\boldsymbol{\gamma}^n : n \in \mathbb{N}\}$ is a dense subset of $T(\boldsymbol{\gamma})$.

[1]Recall that the degree and height of an algebraic number are specified in terms of its its *defining polynomial* $f(x) = \sum_{i=0}^{d} a_i x^i$ (namely the polynomial $f \in \mathbb{Z}[x]$ of minimal degree such that $\gcd(a_0, \ldots, a_d) = 1$ and $f(\alpha) = 0$). In such a case we say that α has degree d and height $\max(|a_0|, \ldots, |a_d|)$.

Lemma 2.4. Let $\boldsymbol{\theta}, \boldsymbol{\psi} \in \mathbb{R}^s$. Suppose that for all $\boldsymbol{v} \in \mathbb{Z}^s$, if $\boldsymbol{v}^\top \boldsymbol{\theta} \in \mathbb{Z}$ then also $\boldsymbol{v}^\top \boldsymbol{\psi} \in \mathbb{Z}$, *i.e.*, all integer relations among the coordinates of $\boldsymbol{\theta}$ also hold among those of $\boldsymbol{\psi}$ (modulo $\mathbb{Z}$). Then, for each $\varepsilon > 0$, there exist $\boldsymbol{p} \in \mathbb{Z}^s$ and a non-negative integer n such that

$$\|n\boldsymbol{\theta} - \boldsymbol{p} - \boldsymbol{\psi}\|_\infty \leq \varepsilon.$$

Let $\boldsymbol{\theta} \in \mathbb{R}^s$ be such that $\boldsymbol{\gamma} = e^{2\pi i \boldsymbol{\theta}}$ (with exponentiation operating coordinate-wise). Notice that $\boldsymbol{\gamma}^{\boldsymbol{v}} = 1$ if and only if $\boldsymbol{v}^\top \boldsymbol{\theta} \in \mathbb{Z}$. If $\boldsymbol{\mu} \in T(\boldsymbol{\gamma})$, we can likewise define $\boldsymbol{\psi} \in \mathbb{R}^s$ to be such that $\boldsymbol{\mu} = e^{2\pi i \boldsymbol{\psi}}$. Then the premises of Lemma 2.4 apply to $\boldsymbol{\theta}$ and $\boldsymbol{\psi}$. Thus, given $\varepsilon > 0$, there exist a non-negative integer k and $\boldsymbol{p} \in \mathbb{Z}^s$ such that $\|k\boldsymbol{\theta} - \boldsymbol{p} - \boldsymbol{\psi}\|_\infty \leq \varepsilon$. Whence

$$\|\boldsymbol{\gamma}^k - \boldsymbol{\mu}\|_\infty = \|e^{2\pi i (k\boldsymbol{\theta} - \boldsymbol{p})} - e^{2\pi i \boldsymbol{\psi}}\|_\infty \leq \|2\pi(k\boldsymbol{\theta} - \boldsymbol{p} - \boldsymbol{\psi})\|_\infty \leq 2\pi\varepsilon.$$

We thus obtain:

Theorem 2.5. Let $\boldsymbol{\gamma} \in \mathbb{T}^s$. Then the orbit $\{\boldsymbol{\gamma}^k : k \in \mathbb{N}\}$ is a dense subset of $T(\boldsymbol{\gamma})$.

2.2 Polyhedra and Linear Programming

We recall some definitions related to polyhedra, integer polyhedra and linear programming (LP), mostly as presented by Ben-Amram and Genaim (2014). Schrijver (1999) is a useful reference for the theory of polyhedra and LP.

2.2.1 Polyhedra

For $R \in \{\mathbb{R}, \mathbb{Q}\}$, a *convex polyhedron* $\mathcal{P} \subseteq R^n$ (*polyhedron* for short) is the set of solutions of a set of inequalities $A\boldsymbol{x} \leq \boldsymbol{b}$, namely $\mathcal{P} = \{\boldsymbol{x} \in R^n \mid A\boldsymbol{x} \leq \boldsymbol{b}\}$, where $\boldsymbol{x} \in R^n$, $A \in \mathbb{Q}^{m \times n}$ is a rational matrix of n columns and m rows, and $\boldsymbol{b} \in \mathbb{Q}^m$ is a column vectors of m rational values. We say that $\mathcal{P}$ is specified by $A\boldsymbol{x} \leq \boldsymbol{b}$. We use calligraphic letters, such as $\mathcal{P}$ and $\mathcal{Q}$ to denote polyhedra. We sometimes write $\mathcal{P}$ as a set that includes the inequalities of $A\boldsymbol{x} \leq \boldsymbol{b}$.

The set of *recession directions* of a polyhedron $\mathcal{P}$ specified by $A\boldsymbol{x} \leq \boldsymbol{b}$ is the set $\texttt{rec.cone}(\mathcal{P}) = \{\boldsymbol{y} \in R^n \mid A\boldsymbol{y} \leq \boldsymbol{0}\}$, and we denoted by $\texttt{rec.cone}(\mathcal{P})$. $\mathcal{P}$ is said to be bounded if $\texttt{rec.cone}(\mathcal{P}) = \{\boldsymbol{0}\}$.

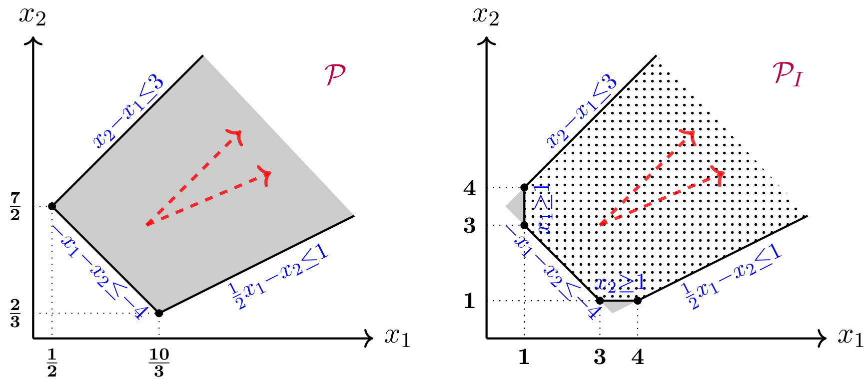


Figure 2.1: A polyhedron $\mathcal{P}$ and its integer hull $\mathcal{P}_I$ (Figure from (Ben-Amram and Genaim, 2014)).

Example 2.1. Consider the polyhedron $\mathcal{P}$ of Figure 2.1 (on the left). The points defined by the Gray area and the black borders are solutions to the system of linear inequalities $\{x_2 - x_1 \leq 3,\ -x_1 - x_2 \leq -4,\ \frac{1}{2}x_1 - x_2 \leq 1\}$.

Let $\mathcal{P} \subseteq R^{n+m}$ be a polyhedron, and let $\binom{\boldsymbol{x}}{\boldsymbol{y}} \in \mathcal{P}$ be such that $\boldsymbol{x} \in R^n$ and $\boldsymbol{y} \in R^m$. The *projection* of $\mathcal{P}$ onto the $\boldsymbol{x}$-space is defined as $\mathtt{proj}_{\boldsymbol{x}}(\mathcal{P}) = \{\boldsymbol{x} \in R^n \mid \exists \boldsymbol{y} \in R^m \text{ such that } \binom{\boldsymbol{x}}{\boldsymbol{y}} \in \mathcal{P}\}$.

2.2.2 Integer Polyhedra

For a given polyhedron $\mathcal{P} \subseteq R^n$ we let $I(\mathcal{P})$ be $\mathcal{P} \cap \mathbb{Z}^n$, *i.e.*, the set of integer points of $\mathcal{P}$. The *integer hull* of $\mathcal{P}$, commonly denoted by $\mathcal{P}_I$, is defined as the convex hull of $I(\mathcal{P})$, *i.e.*, every rational point of $\mathcal{P}_I$ is a convex combination of integer points. This property is fundamental to results presented in the next sections. It is known that $\mathcal{P}_I$ is also a polyhedron. An *integer polyhedron* is a polyhedron $\mathcal{P}$ such that $\mathcal{P} = \mathcal{P}_I$, and in such case we say that $\mathcal{P}$ is *integral*.

Example 2.2. The integer hull $\mathcal{P}_I$ of polyhedron $\mathcal{P}$ of Figure 2.1 (on the left) is given in the same figure (on the right). It is defined by the dotted area and the black border, and is obtained by adding the inequalities $x_1 \geq 1$ and $x_2 \geq 1$ to $\mathcal{P}$. The two Gray triangles next to the edges of $\mathcal{P}_I$ are subsets of $\mathcal{P}$ that were eliminated when computing $\mathcal{P}_I$.

The integer hull of a polyhedron $\mathcal{P}$ can be computed in exponential time (Hartmann, 1988; Charles *et al.*, 2009). Note that this algorithm supports only bounded polyhedra, the integer hull of an unbounded polyhedron is computed by considering a corresponding bounded one (Schrijver, 1999, Th. 16.1, p. 231).

2.2.3 Generator Representation

Polyhedra also have a *generator representation* in terms of vertices and rays[2], written as

$$\mathcal{P} = \texttt{conv}\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\} + \texttt{cone}\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_t\}\,.$$

This means that $\boldsymbol{x} \in \mathcal{P}$ if and only if $\boldsymbol{x} = \sum_{i=1}^{m} a_i \boldsymbol{x}_i + \sum_{j=1}^{t} b_j \boldsymbol{y}_j$ for some rationals $a_i, b_j \geq 0$, where $\sum_{i=1}^{m} a_i = 1$. An important property is that if $\mathcal{P}$ is integral, then there is a generator representation in which all $\boldsymbol{x}_i$ and $\boldsymbol{y}_j$ are integer.

Example 2.3. The generator representations of $\mathcal{P}$ and $\mathcal{P}_I$ of Figure 2.1 are

$$\begin{aligned} \mathcal{P} = &\quad \texttt{conv}\{(\tfrac{1}{2}, \tfrac{7}{2}), (\tfrac{10}{3}, \tfrac{2}{3})\} + \texttt{cone}\{(1,1), (7,3)\} \\ \mathcal{P}_I = &\quad \texttt{conv}\{(1,3), (1,4), (3,1), (4,1)\} + \texttt{cone}\{(1,1), (7,3)\} \end{aligned}$$

The points in `conv` are vertices, they correspond to the points marked with $\bullet$ in Figure 2.1. The rays are the vectors $(1,1), (7,3)$; they describe a direction, rather than a specific point, and are therefore represented in the figure as arrows. Note that the vertices of $\mathcal{P}_I$ are integer points, while those of $\mathcal{P}$ are not. The point $(3,2)$, for example, is defined as $\frac{5}{17} \cdot (\frac{1}{2}, \frac{7}{2}) + \frac{12}{17} \cdot (\frac{10}{3}, \frac{2}{3})\} + \frac{1}{2} \cdot (1,1) + 0 \cdot (7,3)$ in $\mathcal{P}$, and as $0 \cdot (1,3) + \frac{1}{3} \cdot (1,4) + 0 \cdot (3,1) + \frac{2}{3} \cdot (4,1) + 0 \cdot (1,1) + 0 \cdot (7,3)$ in $\mathcal{P}_I$.

2.2.4 Size of Polyhedra

Complexity of algorithms on polyhedra is measured in this survey by running time, on a conventional computational model (polynomially equivalent to a Turing machine), as a function of the *bit-size* of the input. Following Schrijver (1999, Sec. 2.1), we define the bit-size of an integer

[2]Technically, the $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ are only vertices if the polyhedron is *pointed*.

x as $\|x\| = 1 + \lceil \log(|x| + 1) \rceil$; the bit-size of an n-dimensional vector $\boldsymbol{a}$ as $\|\boldsymbol{a}\| = n + \sum_{i=1}^{n} \|a_i\|$; and the bit-size of an inequality $\boldsymbol{a}^\top \boldsymbol{x} \leq c$ as $1 + \|c\| + \|\boldsymbol{a}\|$. For a polyhedron $\mathcal{P} \subseteq R^n$ defined by $A\boldsymbol{x} \leq \boldsymbol{b}$, we let $\|\mathcal{P}\|_b$ be the bit-size of $A\boldsymbol{x} \leq \boldsymbol{b}$, which we can take as the sum of the sizes of the inequalities.

2.2.5 Farkas' Lemma

Used in sections 4–6

Many of the techniques presented in this survey heavily rely on (a variation) of Farkas' Lemma (Schrijver, 1999, p. 94), which states that a polyhedron $\mathcal{P} \subseteq R^n$, with $R \in \{\mathbb{Q}, \mathbb{R}\}$, specified by $A\boldsymbol{x} \leq \boldsymbol{c}$, entails an inequality $\vec{\lambda}\boldsymbol{x} \leq \lambda_0$ if and only if there is a vector of non-negative coefficients $\vec{\mu}$, of appropriate dimension, such that the following holds:

$$\vec{\mu}A = \vec{\lambda} \tag{2.1}$$

$$\vec{\mu}\boldsymbol{c} \leq \lambda_0 \tag{2.2}$$

The vector $\vec{\mu}$ will be called the Farkas' coefficients in the rest of this survey. It is also easy to show that $\vec{\lambda}\boldsymbol{x} \leq \lambda_0$ is entailed by $I(\mathcal{P})$, *i.e.*, by the set of integer points of $\mathcal{P}$, if and only if it is entailed by $\mathcal{P}_I$. This follows from the fact that if the inequality holds for points $\boldsymbol{x}_1 \in I(\mathcal{P})$ and $\boldsymbol{x}_2 \in I(\mathcal{P})$, then it holds for their convex combinations. Note that (2.1,2.2) are linear constraints when considering λ_0, $\vec{\lambda}$, and $\vec{\mu}$ as unknowns, and thus synthesising entailed inequalities can be done in polynomial time by seeking a solution for (2.1,2.2). Note also that for some techniques, such as those based on templates, A and $\boldsymbol{c}$ might also include unknowns, and thus (2.1,2.2) are non-linear in such case.

Example 2.4. Consider a polyhedron $\mathcal{P}$ defined by the following set of inequalities (those of Figure 2.1 on the left)

$$\{-x_1 - x_2 \leq -4,\ x_2 - x_1 \leq 3,\ \frac{1}{2}x_1 - x_2 \leq 1\} \tag{2.3}$$

and its matrix representation $A\boldsymbol{x} \leq \boldsymbol{c}$ where

$$A = \begin{pmatrix} -1 & -1 \\ -1 & 1 \\ \frac{1}{2} & -1 \end{pmatrix} \qquad \boldsymbol{c} = \begin{pmatrix} -4 \\ 3 \\ 1 \end{pmatrix}$$

Let $\lambda_1 x_1 + \lambda_2 x_2 \le \lambda_0$ be an implied inequality template, and $\vec{\mu} = (\mu_0, \mu_1, \mu_2)$. Note that $\vec{\mu}$ has components like the number of inequalities, the rows of A. To synthesise inequalities implied by $A\boldsymbol{x} \le \boldsymbol{c}$, we use (2.1,2.2) to generate the following constraints system:

$$\begin{aligned} -\mu_0 - \mu_1 + \tfrac{1}{2}\mu_2 = \lambda_1,\ -\mu_0 + \mu_1 - \mu_2 = \lambda_2, \\ -4\mu_0 + 3\mu_1 + \mu_2 \le \lambda_0 \\ \mu_0 \ge 0,\ \mu_1 \ge 0,\ \mu_2 \ge 0 \end{aligned} \tag{2.4}$$

The constraints in first line come from (2.1), and correspond to multiplying $\vec{\mu}$ by the columns of A. The constraint in the second line comes from (2.2), and correspond to multiplying $\vec{\mu}$ by $\boldsymbol{c}$. The third line is used to require the coefficients $\vec{\mu}$ to be non-negative.

The valuation $\{\lambda_1 \mapsto -1, \lambda_2 \mapsto 0, \lambda_0 \mapsto -\frac{1}{2}, \mu_0 \mapsto \frac{1}{2}, \mu_1 \mapsto \frac{1}{2}, \mu_2 \mapsto 0\}$ is a solution for (2.4), and thus $-x_1 \le -\frac{1}{2}$ is an implied inequality.

If we are interested in an implied inequality of a specific form, *e.g.*, one in which $\lambda_1 = \lambda_2$ or $\lambda_1 \le \lambda_2$, we can add a corresponding constraint to (2.4). If we are interested in several implied inequalities, that share some coefficients, we can solve several instances of (2.4) at the same time (even if each is implied by a different $A\boldsymbol{x} \le \boldsymbol{c}$). Finally, if we are interested in inequalities that are implied only by $I(P)$, *i.e.*, the integer points of $\mathcal{P}$, we can use the constraints that represent its integer-hull $\mathcal{P}_I$ (the polyhedron of Figure 2.1 on the right).

2.2.6 Linear Programming

A linear programming (LP) problem concerns the maximisation or minimisation of a linear objective function, such as $\boldsymbol{a}\boldsymbol{x}$, subject to a system of linear inequalities, typically represented as $A\boldsymbol{x} \le \boldsymbol{c}$. It can also refer to the problem finding a solution that satisfies the inequalities. When the variables are restricted to take real or rational values, an LP problem can be solved in polynomial time. However, if the variables are restricted to be integers, the problem is known as an integer linear programming problem, which is `NP`-hard.

2.3 Programs

A program is often modelled as a transition relation $T \subseteq S \times S$, where S is a set of possible program states. An execution, or a trace, is a (possibly infinite) sequence $s_0, s_1, \ldots$ where $(s_i, s_{i+1}) \in T$. A transition relation $T \subseteq S \times S$ or a set of states $S' \subseteq S$ are often defined by predicates (formulas whose models define the elements of the set), and thus we write $T(s, s')$ and $S'(s)$ instead of $(s, s') \in T$ and $s \in S'$. The successors operator $\mathsf{post}_T : S \to S$ is $\mathsf{post}_T(X) = \{s' \in S \mid s \in X, (s, s') \in T\}$, and the predecessors operator $\mathsf{pre}_T : S \to S$ is $\mathsf{pre}_T(X) = \{s \in S \mid s' \in X, (s, s') \in T\}$. For an initial set of states S_0, the set of reachable states $\mathsf{RCH}(T, S_0)$ contains the states that can be reached fro S_0 by a finite trace; this is the least fixpoint of $F(X) = S_0 \cup \mathsf{post}_T(X)\}$ over the domain of sets of states $\langle \wp(S), \emptyset, S, \cap, \cup \rangle$. The restriction of T to the reachable states $\mathsf{RCH}(T, S_0)$ is defined as $T_{S_0} = \{(s, s') \in T \mid s \in \mathsf{RCH}(T, S_0)\}$.

We say that T is *terminating* for an initial state $s_0 \in S$, if there are no infinite traces starting with s_0, and *non-terminating* if such an infinite trace exists. We say that T is *universally terminating* if it is terminating for any initial state. Equivalently, T is universally terminating if and only if it is well-founded (when considered as a "greater than" relation). Note that termination of T wrt. S_0 is equivalent to universal termination of T_{S_0}. As in much of the literature, the unqualified term *termination* means universal termination if no reference to particular initial states is made, and *non-termination* means the negation of universal termination. The problem of deciding whether T is terminating for a given *single* initial state $s_0 \in S$ is known as the *halting* problem.

2.3.1 Linear-Constraint Control-Flow Graphs

Structured program representations, such as the *Control-Flow Graph* (CFG), are often employed for practical reasons since they are easily derived from real-world programming languages. Furthermore, our focus is restricted to program states that involve only numerical variables.

A CFG is a tuple $P = (V, R, L, \ell_0, E)$, where:

(i) $V = \{x_1, \ldots, x_n\}$ is a finite set of program variables taking values from a numerical domain $R \in \{\mathbb{R}, \mathbb{Q}, \mathbb{Z}\}$;

(ii) $L = \{l_0, \ldots, l_k\}$ is a finite set of locations, where $\ell_0 \in L$ represents the initial location; and

(iii) $E \subseteq L \times \wp(R^n \times R^n) \times L$ is a set of edges annotated with transition relations over R^n.

An edge $(\ell, T, \ell') \in E$ define how an execution step can move from location ℓ to ℓ': if the execution is at location ℓ, the variables have values $\boldsymbol{x} \in R^n$, and $(\boldsymbol{x}, \boldsymbol{x}') \in T$ then we can move to location ℓ' and set the program variables to $\boldsymbol{x}'$. Sometimes we write $T_{\ell,\ell'} \in E$ to refer to the transition relation directly. We can also write $(\ell, T, \ell') \in P$ and $\ell \in P$ instead of referring to the sets of edges and locations. Viewing states as tuples $(\ell, \boldsymbol{x}) \in L \times R^n$, it is easy to see that a CFG P induces a transition relation $T_P \subseteq (L \times R^n) \times (L \times R^n)$. When the location is known from context, we sometimes omit the location and refer to the variables $\boldsymbol{x}$ as "the state".

A common way of representing a numerical transition relation $T \subseteq R^n \times R^n$ is as a conjunction of linear constraints, where the ith constraint is of the form $\sum_{j=1}^{n} a_{ij} x_j + \sum_{j=1}^{n} a'_{ij} x'_j \leq c_i$. Here, $(x_1, \ldots, x_n)^\top$ represents the current state and $(x'_1, \ldots, x'_n)^\top$ represents a possible successor. Such a transition relation is a polyhedron, and is specified by $A''\boldsymbol{x}'' \leq \boldsymbol{c}''$ where $\boldsymbol{x}'' = \binom{\boldsymbol{x}}{\boldsymbol{x}'}$, $A'' \in \mathbb{Q}^{m \times 2n}$, and $\boldsymbol{c}'' \in \mathbb{Q}^{m \times 1}$ for some $m \geq 1$ (the number of constraints in the conjunction). Note that all coefficients are rational, but in some settings we will assume that they are integer. We call this polyhedron a *transition polyhedron* and denote it by $\mathcal{Q} \subseteq \mathbb{R}^{2n}$. Note that if the domain is the integers, the set of transitions is $I(\mathcal{Q}) \subseteq \mathbb{Z}^{2n}$.

We sometimes write $A''\boldsymbol{x}'' \leq \boldsymbol{c}''$ as $A\boldsymbol{x} + A'\boldsymbol{x}' \leq \boldsymbol{c}''$ for appropriate $A, A' \in \mathbb{Q}^{m \times n}$, or as $B\boldsymbol{x} \leq \boldsymbol{b} \wedge A\boldsymbol{x} + A'\boldsymbol{x}' \leq \boldsymbol{c}$ when we are explicitly interested in the condition ($B\boldsymbol{x} \leq \boldsymbol{b}$) that allows taking the corresponding edge (the *guard* of the edge). We may also use $=$ and $\geq$ instead of $\leq$ when writing constraints, as such constraints can be naturally converted to use $\leq$ only. We also write a conjunction of inequalities as a set, in which case the empty set represents the constraint *true* (*i.e.*, the whole space). We also write $\mathcal{P}_1 \wedge \mathcal{P}_2$ to refer to the polyhedron specified by the constraints of both $\mathcal{P}_1$ and $\mathcal{P}_2$ (even if they use different variables).

We call a transition polyhedron *deterministic* if, for a given state $\boldsymbol{x} \in R^n$ there is at most one state $\boldsymbol{x}' \in R^n$ such that $(\boldsymbol{x}, \boldsymbol{x}') \in \mathcal{Q}$.

A linear-constraint CFG is a CFG where edges are annotated with transition polyhedra. In this survey, the term CFG will refer to a linear-constraint CFG unless otherwise specified.

Remark 2.1. For simplicity, this survey always uses non-strict linear inequalities (*i.e.*, $\leq$). Many of the results presented here can be generalised to include strict inequalities, a point we will explicitly note. This distinction is crucial only for rational and real variables; for integers, strict inequalities can be converted into equivalent non-strict ones, so we may use both in our examples.

Remark 2.2. Linear-constraint CFGs can also represent programs that manipulate data structures. This is usually done by abstracting the data structures into numerical representations—for example, the length of a list, the depth of a tree, *etc.* (Lindenstrauss and Sagiv, 1997; Lee *et al.*, 2001; Bruynooghe *et al.*, 2007; Spoto *et al.*, 2010; Magill *et al.*, 2010). While these abstractions are typically sound for proving termination, they are not always sound for proving non-termination.

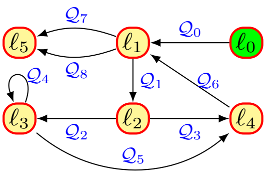
When proving termination and non-termination for CFGs, we are primarily interested in executions that start from the initial location ℓ_0. We may also restrict the input variables to a given set of values $S_0 \subseteq R^n$; we sometimes omit S_0 because it can be represented by adding an initial transition out of ℓ_0. Universal termination for CFGs allows starting at any location with any values for the variables.

Many of the termination and non-termination techniques in this survey rely on local, edge-level reasoning. Consequently, they cannot easily account for information from preceding edges or assumptions about the initial state unless that information is propagated to each location using invariants.

Definition 2.1. We call $I_\ell \subseteq R^n$ an invariant for a location ℓ if, for any execution starting from $(\ell_0, \boldsymbol{x})$ where $\boldsymbol{x} \in S_0$, all reachable states $(\ell, \boldsymbol{x}) \in \mathsf{RCH}(T_P, S_0)$ satisfy $\boldsymbol{x} \in I_\ell$.

In this survey, we focus on polyhedral invariants.

```
while(x >= 0 && y >= 0) {
  if (nondet()) {
    while (y <= z && nondet())
      y++;
    x--;
  } else {
    y--;
  }
}
```



$\mathcal{Q}_0$: $\{x' = x, y' = y, z' = z\}$
$\mathcal{Q}_1$: $\{x \geq 0, y \geq 0, x' = x, y = y, z' = z\}$
$\mathcal{Q}_2$: $\{x' = x, y' = y, z' = z\}$
$\mathcal{Q}_3$: $\{x' = x, y' = y - 1, z' = z\}$
$\mathcal{Q}_4$: $\{y \leq z, x' = x, y' = y + 1, z' = z\}$
$\mathcal{Q}_5$: $\{x' = x - 1, y = y, z' = z\}$
$\mathcal{Q}_6$: $\{x' = x, y = y, z' = z\}$
$\mathcal{Q}_7$: $\{x \leq -1, x' = x, y = y, z' = z\}$
$\mathcal{Q}_8$: $\{y \leq -1, x' = x, y = y, z' = z\}$

$\mathcal{S}_0 = \mathcal{I}_{\ell_0} = \mathcal{I}_{\ell_2} = \mathcal{I}_{\ell_3} = \{x \geq 0, y \geq 0\}$
$\mathcal{I}_{\ell_2} = \mathcal{I}_{\ell_3} = \{x \geq 0, y \geq 0\}$
$\mathcal{I}_{\ell_1} = \mathcal{I}_{\ell_4} = \mathcal{I}_{\ell_5} = \{x \geq -1, y \geq -1\}$

Figure 2.2: A program, its corresponding CFG, and invariants.

Remark 2.3. Inferring polyhedral invariants is outside the scope of this survey; we assume they have been inferred beforehand and are provided as input. However, some techniques combine invariant inference with the search for termination (or non-termination) witnesses, and we will explicitly comment on those.

Example 2.5. Figure 2.2 presents an imperative program (in a C-like language), along with a possible corresponding CFG and its invariants. The `nondet()` instruction produces an arbitrary (integer) value and is typically used to abstract expressions that cannot be modelled with linear arithmetic.

2.3.2 Linear-Constraint Loops

This section presents special cases of CFGs that are in the form of loops.

Multi-path Linear-Constraint Loops

A CFG with a single node and k edges is called a *multipath* linear-constraint loop (*MLC* for short), and can be represented by a set of polyhedra $\mathcal{Q}_1, \ldots, \mathcal{Q}_k$, each specified by $A''_i \boldsymbol{x}'' \leq \boldsymbol{c}''_i$ (the location need not be specified). This kind of CFGs arise in program analysis as an abstraction of an iterative (or recursive) code that includes branching in the loop body. When we are interested in the conditions that allows the corresponding edge to be taken, we rewrite $A_i''\boldsymbol{x}'' \leq \boldsymbol{c}_i''$ as $B_i\boldsymbol{x} \leq \boldsymbol{b}_i \wedge A_i\boldsymbol{x}'' \leq \boldsymbol{c}_i$ where, for some $p_i, q_i > 0$, $B_i \in \mathbb{Q}^{p_i \times n}$, $A_i \in \mathbb{Q}^{q_i \times 2n}$, $\boldsymbol{b}_i \in \mathbb{Q}^{p_i}$, $\boldsymbol{c}_i \in \mathbb{Q}^{q_i}$. For a path i, the constraint $B_i\boldsymbol{x} \leq \boldsymbol{b}_i$ is called *the path guard*, and the other constraint is called *the update*. We say that the loop is a *real*, *rational*, or *integer* loop depending on the domain of the variables. We say that there is a transition from a state $\boldsymbol{x} \in R^n$ to a state $\boldsymbol{x}' \in R^n$, if there is a path i such that $\boldsymbol{x}$ satisfies its guard and $\boldsymbol{x}$ and $\boldsymbol{x}'$ satisfy its update. We also consider *MLC* loops with an initial polyhedral set of states $\mathcal{S}_0$.

Example 2.6. Let $\mathcal{Q}_1 = \{x_1 \geq 0, x_1' = x_1 - 1\}$ and $\mathcal{Q}_2 = \{x_2 \geq 0, x_2' = x_2 - 1, x_1' \leq x_1\}$. Then $\mathcal{Q}_1, \mathcal{Q}_2$ is an *MLC* loop with two paths.

Single-path Linear-Constraint Loops

A *single-path* linear-constraint loop (*SLC* for short) is a special case of *MLC* loop with a single path, *i.e.*, the corresponding CFG has a single edge. We represent such a loop by a single transition polyhedron $\mathcal{Q}$ specified by $A''\boldsymbol{x}'' \leq \boldsymbol{c}''$. If we are explicitly interested in the condition that allows the edge to be taken, we write it as a *while* loop of the following form:

$$\textit{while } (B\boldsymbol{x} \leq \boldsymbol{b}) \textit{ do } A\boldsymbol{x}'' \leq \boldsymbol{c} \tag{2.5}$$

Example 2.7. Consider the *SLC* loop $\mathcal{Q} = \{4x_1 \geq x_2, x_2 \geq 1, 5x_1' \leq 2x_1 + 1, 5x_1' \geq 2x_1 - 3, x_2' = x_2\}$. We can also write this as follows to make the condition and the update explicit:

$$\textit{while } (4x_1 \geq x_2, x_2 \geq 1) \textit{ do } 5x_1' \leq 2x_1 + 1, 5x_1' \geq 2x_1 - 3, x_2' = x_2 \tag{2.6}$$

This loop, interpreted over the integers, represents the C language loop

```
while (4*x1>=x2 && x2>=1) x1=(2*x1+1)/5;
```

Note that if Loop (2.6) is interpreted over the rationals, it becomes nondeterministic.

Affine Single-path Linear-Constraint Loops

An affine *SLC* loop is a special case of *SLC* loops where the update can be described as a linear transformation, and is written as:

$$\textit{while } (B\boldsymbol{x} \leq \boldsymbol{b}) \textit{ do } \boldsymbol{x}' = A\boldsymbol{x} + \boldsymbol{c} \tag{2.7}$$

where $\boldsymbol{x} = (x_1, \ldots, x_n)^\top$ and $\boldsymbol{x}' = (x'_1, \ldots, x'_n)^\top$ are column vectors, and for some $m > 0$, $B \in \mathbb{Q}^{m \times n}$, $A \in \mathbb{Q}^{n \times n}$, $\boldsymbol{b} \in \mathbb{Q}^m$, $\boldsymbol{c} \in \mathbb{Q}^n$. When it is convenient, we also write such loops as an imperative loop

$$\textit{while } (g_1(\boldsymbol{x}) \geq 0 \wedge \ldots \wedge g_m(\boldsymbol{x}) \geq 0) \textit{ do } \boldsymbol{x} := f(\boldsymbol{x})\,, \tag{2.8}$$

where $g_i(\boldsymbol{x}) = -\boldsymbol{b}_i^\top \boldsymbol{x} + b_i$ with $\boldsymbol{b}_i$ being the ith row of B and b_i the ith element of $\boldsymbol{b}$, and $f(\boldsymbol{x}) = A\boldsymbol{x} + \boldsymbol{c}$. The term *linear loops* is frequently used in the literature to refer to affine *SLC* loops.

2.3.3 Counter Programs

Counter programs (also known as counter machines) are a universal computational model (Minsky, 1967) used in this survey to study the decidability of classes of linear programs through reduction.

A (deterministic) counter program P_C with n (integer) counters $X_1, \ldots, X_n$ is a list of labelled instructions $1{:}I_1, \ldots, m{:}I_m, m{+}1{:}\bot$ where each instruction I_k is one of the following:

$$\textit{incr}(X_j) \mid \textit{decr}(X_j) \mid \textit{if } X_j > 0 \textit{ then } k_1 \textit{ else } k_2$$

with $1 \leq k_1, k_2 \leq m{+}1$ and $1 \leq j \leq n$.

A state is of the form $(k, (a_1, \ldots, a_n)^\top)$ which indicates that Instruction I_k is to be executed next, and the current values of the counters are $X_1 = a_1, \ldots, X_n = a_n$. In a valid state, $1 \leq k \leq m + 1$ and all $a_i \in \mathbb{N}$. Any state in which $k = m + 1$ is a halting state. For any other valid state $(k, \langle a_1, \ldots, a_n \rangle)$, the successor state is defined as follows:

- If I_k is $\textit{decr}(X_j)$ (resp. $\textit{incr}(X_j)$), then X_j is decreased (resp. increased) by 1 and the execution moves to label $k + 1$.
- If I_k is "$\textit{if } X_j > 0 \textit{ then } k_1 \textit{ else } k_2$", then the execution moves to label k_1 if X_j is positive, and to k_2 if it is 0. The values of the counters do not change.

Since counter programs are a universal computational model, they have an undecidable halting problem (termination from a provided initial state). We know that the (universal) termination problem is undecidable as well.

Theorem 2.6 ((Blondel *et al.*, 2001)). Universal termination of counter programs is undecidable, even restricted to 2-counter programs.

3

Decidability of Termination of Linear-Constraint Programs

In this section, we overview decidability and undecidability results for termination of the different linear-constraint program types introduced in Section 2.3, both with and without initial states. This is a crucial and challenging research area because it establishes the fundamental limits of termination analysis.

From a theoretical perspective, determining if such programs always terminate is a non-trivial problem that often requires sophisticated mathematical tools from areas like linear algebra, number theory, and geometry. Furthermore, the decidability of termination for linear-constraint programs is highly dependent on the variable domain (integers, rationals, or reals). A loop that terminates for integer variables might not terminate for reals. Typically, integer linear-constraint programs are the most difficult to analyse.

For at least two decades, the decidability of termination for linear programs has received considerable attention. Much of the progress in this area has focused on affine *SLC* loops, for which many decidability results have been established over $\mathbb{R}$, $\mathbb{Q}$ and $\mathbb{Z}$. The main part of this section provides an overview of these results. The more complex case of general *SLC* loops remains a significant open problem, though some special cases and extensions of this model have been considered. For *MLC* loops, the problem becomes even more difficult, and the research has primarily yielded undecidability results, even for a small number of paths or variables.

Organisation of this Section. Section 3.1 discusses termination of affine *SLC* loops, Section 3.2 discusses termination of *SLC* loops, and Section 3.3 discusses termination of *MLC* loops.

3.1 Termination of Affine Single-path Linear-Constraint Loops

In this section, we consider the termination of affine *SLC* loops (like Loop (2.8)), where the loop body has a single control path that performs a simultaneous affine update of the program variables. Analysing these loops, including acceleration and termination, can be part of the analysis for more complex programs (Boigelot, 2003; Jeannet *et al.*, 2014; Kincaid *et al.*, 2019).

We are primarily interested in universal termination—that is, determining whether these loops terminate for all initial values of the program variables, regardless of whether the domain of variables is $\mathbb{R}$, $\mathbb{Q}$, or $\mathbb{Z}$. We also discuss termination from a specific set of initial states in Section 3.1.4.

The following examples, taken from Braverman (2006), illustrate several relevant phenomena, including how termination depends on the domain of the loop variables.

Example 3.1. Consider the loop:

$$\text{while } (4x + y \geq 1) \text{ do } \begin{pmatrix} x \\ y \end{pmatrix} := \begin{pmatrix} -2 & 4 \\ 4 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}.$$

The matrix in the loop body has two eigenvectors:

$$\boldsymbol{v}_1 := (-1 - \sqrt{17}, 4) \quad \text{and} \quad \boldsymbol{v}_2 := (-1 + \sqrt{17}, 4)\,,$$

respectively corresponding to the eigenvalues:

$$\lambda_1 := -1 - \sqrt{17} \quad \text{and} \quad \lambda_2 := -1 + \sqrt{17}\,.$$

The eigenvector $\boldsymbol{v}_2$ satisfies the loop guard and corresponds to a positive eigenvalue. Hence the loop does not terminate over $\mathbb{R}$. However, the line through the origin parallel to $\boldsymbol{v}_2$ does not contain any rational points other than 0, and the loop outside this line is dominated by the negative eigenvalue λ_1, which is larger in absolute value than λ_2. At the limit, the orbit of (x, y) alternates between the directions $\boldsymbol{v}_1$ and $-\boldsymbol{v}_1$. Hence, the loop terminates on $\mathbb{Q}$.

Example 3.2. Consider the loop:

$$\text{while } (4x - 5y \geq 1) \text{ do } \begin{pmatrix} x \\ y \end{pmatrix} := \begin{pmatrix} 2 & 4 \\ 4 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}.$$

The matrix has two eigenvectors:

$$\boldsymbol{v}_1 := (1 + \sqrt{17}, 4) \quad \text{and} \quad \boldsymbol{v}_2 := (1 - \sqrt{17}, 4),$$

respectively corresponding to the eigenvalues:

$$\lambda_1 := 1 + \sqrt{17} \quad \text{and} \quad \lambda_2 := 1 - \sqrt{17}.$$

The eigenvalue λ_1 is positive and dominant and so all points on the half-line $L = \{r\boldsymbol{v}_1 : 4(r-1) \geq \sqrt{17}\}$ in the direction of $\boldsymbol{v}_1$ are non-terminating (note that the lower bound on r ensures that the points satisfy the loop guard). The half-line L does not contain any rational points, however a suitably small perturbation of a point on L remains non-terminating since such a point converges to L as the loop unfolds. Thus there is a cone of non-terminating points around L that contains rational points and even integer points. For example, the point $(9, 7)$ is non-terminating.

Example 3.3. The following loop terminates over the integers but not over the rationals:

$$\textit{while } (x \geq 0) \textit{ do } x := -2x + 1.$$

The only non-terminating initial value is $x = \frac{1}{2}$.

When considering termination over $\mathbb{R}$ and $\mathbb{Q}$, we assume all numerical constants in the loops are rational. Similarly, for termination over $\mathbb{Z}$, we assume all numerical constants are integers. Despite the simplicity of affine SLC loops, the question of deciding termination has proven challenging. Tiwari (2004) showed that termination for these loops is decidable over $\mathbb{R}$. Subsequently, Braverman (2006), using a more refined analysis, showed that termination is decidable over $\mathbb{Q}$ and noted that termination on $\mathbb{Z}$ can be reduced to termination on $\mathbb{Q}$ in the homogeneous case, *i.e.*, when $\boldsymbol{b}, \boldsymbol{c}$ in (2.8) are both all-zero vectors (this result is for loops with strict inequalities, for non-strict ones the loop obviously does not terminate with this change). Finally, Hosseini *et al.* (2019) gave a procedure for deciding termination over the integers without restriction.

Overview of the Section

The rest of this section presents a uniform framework, based on the work of Hosseini *et al.* (2019), that shows how to decide termination over $\mathbb{R}$, $\mathbb{Q}$, and $\mathbb{Z}$. The high-level idea is that for a given linear loop with n variables, one computes a convex semi-algebraic set $PN \subseteq \mathbb{R}^n$ of *potentially non-terminating points*. The key properties of PN are that (i) it contains all non-terminating initial values in $\mathbb{R}^n$; (ii) it is a loop invariant; (iii) all points in the relative interior of PN are non-terminating. These properties can be used to show that for each ring $R \in \{\mathbb{R}, \mathbb{Q}, \mathbb{Z}\}$, the loop is non-terminating over R if and only if PN contains a point in R^n. Then termination of the given loop over $\mathbb{R}$ reduces to checking non-emptiness of PN. Thus, termination over $\mathbb{Q}$ or $\mathbb{Z}$ can respectively be determined using procedures of Khachiyan and Porkolab (Din and Zhi, 2010; Khachiyan and Porkolab, 1997) for determining whether a given convex semi-algebraic set contains a rational point and whether it contains an integer point.

The construction of the set PN of potentially non-terminating points and verification of its properties relies on Kronecker's theorem on simultaneous Diophantine approximation and a result of Masser (Masser, 1988) that allows computing all multiplicative relations among the eigenvalues of the update matrix of a given loop (see Section 2.1.6). To analyse termination over $\mathbb{Z}$ we also use Kinchine's Flatness Theorem, which gives sufficient conditions for a convex set to contain an integer point (see Section 2.1.5).

The rest of this section is structured as follows: Section 3.1.1 classifies the termination behaviour of initial values; Section 3.1.2 discusses the termination of affine *SLC* loops with a single guard; Section 3.1.3 discuss the termination of affine *SLC* loops with a multiple guards; and finally Section 3.1.5 overviews related work.

3.1.1 Classifying Initial Values

Reduction to the Non-Degenerate Case

Recall that the general form of an affine *SLC* loop with n variables is as follows:

$$\textit{while } (g_1(\boldsymbol{x}) \geq 0 \wedge \ldots \wedge g_m(\boldsymbol{x}) \geq 0) \textit{ do } \boldsymbol{x} := f(\boldsymbol{x}) \,,$$

where $g_1, \ldots, g_m : \mathbb{R}^n \to \mathbb{R}$ and $f : \mathbb{R}^n \to \mathbb{R}^n$ are affine functions with rational coefficients, that is, $f(\boldsymbol{x}) = A\boldsymbol{x} + \boldsymbol{a}$ for $A \in \mathbb{Q}^{n \times n}$ and $\boldsymbol{a} \in \mathbb{Q}^n$, and $g_i(\boldsymbol{x}) = \boldsymbol{b}_i^\top \boldsymbol{x} + c_i$ for $\boldsymbol{b}_i \in \mathbb{Q}^n$, $c_i \in \mathbb{Q}$ and $i = 1, \ldots, m$. Note that

$$\begin{pmatrix} f(\boldsymbol{x}) \\ 1 \end{pmatrix} = \begin{pmatrix} A & \boldsymbol{a} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ 1 \end{pmatrix} \text{ and } g_i(\boldsymbol{x}) = (\boldsymbol{b}_i^\top \; c_i) \begin{pmatrix} \boldsymbol{x} \\ 1 \end{pmatrix} . \qquad (3.1)$$

for all $\boldsymbol{x} \in \mathbb{R}^n$. We say that f is *non-degenerate* if no quotient of two distinct eigenvalues of the update matrix $\left(\begin{smallmatrix} A & \boldsymbol{a} \\ 0 & 1 \end{smallmatrix}\right)$ is a root of unity.

We claim that the termination problem for affine *SLC* loops is reducible to the special case of the problem for non-degenerate update functions. To prove the claim, consider an affine *SLC* loop, as described above, whose update matrix has distinct eigenvalues $\lambda_1, \ldots, \lambda_s$. Let L be the least common multiple of the orders of the roots of unity appearing among the quotients $\frac{\lambda_i}{\lambda_j}$ for $i \neq j$. It is known that $L = 2^{O(n\sqrt{\log n})}$ (Everest *et al.*, 2003, Section 1.1.9). The update matrix corresponding to the affine map $f^L = f \circ \cdots \circ f$ (L times) has eigenvalues $\lambda_1^L, \ldots, \lambda_s^L$ and hence is non-degenerate. Moreover the original loop terminates if and only if the following loop terminates:

$$\textit{while } \bigwedge_{i=0}^{L-1} \Big(g_1(f^i(\boldsymbol{x})) \geq 0 \wedge \ldots \wedge g_m(f^i(\boldsymbol{x})) \geq 0 \Big) \textit{ do } \boldsymbol{x} := f^L(\boldsymbol{x}) \,,$$

But this loop is non-degenerate and the argument is complete.

Spectral Analysis

Let us focus now on the case of an affine *SLC* loop of the form

$$\textit{while } (g(\boldsymbol{x}) \geq 0) \textit{ do } \boldsymbol{x} := f(\boldsymbol{x}) \tag{3.2}$$

with a single guard function $g(\boldsymbol{x}) = \boldsymbol{b}^\top \boldsymbol{x} + c$ and with non-degenerate update function $f(\boldsymbol{x}) = A\boldsymbol{x} + \boldsymbol{a}$, with both maps having rational coefficients. We show that a spectral analysis of the matrix underlying the loop update function suffices to classify almost all initial values of the loop as either terminating or eventually non-terminating. We isolate a class of points called *critical points* for the loop for which the spectral analysis does not determine whether or not they are terminating.

With respect to Loop (3.2) we say that $\boldsymbol{x} \in \mathbb{R}^n$ is *terminating* if there exists $m \in \mathbb{N}$ such that $g(f^m(\boldsymbol{x})) < 0$. We say that $\boldsymbol{x}$ is *eventually non-terminating* if the sequence $\langle g(f^m(\boldsymbol{x})) : m \in \mathbb{N}\rangle$ is *ultimately positive*, *i.e.*, there exists N such that for all $m \geq N$, $g(f^m(\boldsymbol{x})) \geq 0$. Let R be a sub-ring of $\mathbb{R}$ that is preserved by f, that is, such that $f(R^n) \subseteq R^n$. Then there exists $\boldsymbol{z} \in R^n$ that is non-terminating if and only if there exists $\boldsymbol{z} \in R^n$ that is eventually non-terminating. Thus we can regard the problem of deciding termination on R^n as that of searching for an eventually non-terminating point in R^n. Note that f certainly preserves $\mathbb{R}$ and $\mathbb{Q}$ and it moreover preserves $\mathbb{Z}$ if we assume that the coefficients of A and $\boldsymbol{a}$ are integer.

Let $\lambda_1, \ldots, \lambda_s$ be the non-zero eigenvalues of $\left(\begin{smallmatrix} A & \boldsymbol{a} \\ 0 & 1 \end{smallmatrix}\right)$ and let $k_{\max}$ be the maximum multiplicity over all these eigenvalues. Define a linear pre-order on $I := \{0, \ldots, k_{\max} - 1\} \times \{1, \ldots, s\}$ by $(i_1, j_1) \preccurlyeq (i_2, j_2)$ if either (i) $|\lambda_{j_1}| < |\lambda_{j_2}|$ or (ii) $|\lambda_{j_1}| = |\lambda_{j_2}|$ and $i_1 \leq i_2$. Write $(i_1, j_1) \prec (i_2, j_2)$ if $(i_1, j_1) \preccurlyeq (i_2, j_2)$ and $(i_2, j_2) \not\preccurlyeq (i_1, j_1)$. Then we have

$$(i_1, j_1) \prec (i_2, j_2) \iff \lim_{m\to\infty} \frac{\binom{m}{i_1}|\lambda_{j_1}|^m}{\binom{m}{i_2}|\lambda_{j_2}|^m} = 0\,,$$

that is, the preorder $\preccurlyeq$ characterises the asymptotic order of growth in absolute value of the terms $\binom{m}{i}\lambda_j^m$ for $(i, j) \in I$. This preorder, moreover, induces an equivalence relation $\approx$ on I where $(i_1, j_1) \approx (i_2, j_2)$ if and only if $(i_1, j_1) \preccurlyeq (i_2, j_2)$ and $(i_2, j_2) \preccurlyeq (i_1, j_1)$.

The following closed-form expression for $g(f^m(\boldsymbol{x}))$ will be the focus of the subsequent development. The expression is obtained from the Jordan-Chevalley decomposition of the affine map f.

Proposition 3.1. There are affine functions $h_{i,j} : \mathbb{R}^n \to \mathbb{C}$ such that for all $\boldsymbol{x} \in \mathbb{R}^n$ and all $m \geq n$ we have $g(f^m(\boldsymbol{x})) = \sum_{(i,j)\in I} \binom{m}{i} \lambda_j^m \, h_{i,j}(\boldsymbol{x})$.

Define $\gamma_i = \frac{\lambda_i}{|\lambda_i|}$ for $i = 1, \ldots, s$, that is, we obtain γ_i by normalising the eigenvalues to have length 1. Recall from Section 2.1.6 the definition of the group $L(\boldsymbol{\gamma})$ of multiplicative relations that hold among $\gamma_1, \ldots, \gamma_s$, namely,

$$L(\boldsymbol{\gamma}) = \{(n_1, \ldots, n_s) \in \mathbb{Z}^s : \gamma_1^{n_1} \cdots \gamma_s^{n_s} = 1\}.$$

Recall also that we have $T(\boldsymbol{\gamma}) \subseteq \mathbb{T}^s$, given by

$$T(\boldsymbol{\gamma}) = \{(\mu_1, \ldots, \mu_s) \in \mathbb{T}^s : \mu_1^{n_1} \cdots \mu_s^{n_s} = 1 \text{ for all } (n_1, \ldots, n_s) \in L(\boldsymbol{\gamma})\}.$$

Given an $\approx$-equivalence class $L \subseteq I$, for all $(i_1, j_1), (i_2, j_2) \in L$ we have $i_1 = i_2$ and $|\lambda_{j_1}| = |\lambda_{j_2}|$. Thus L determines a common multiplicity, which we denote i_L, and a set of eigenvalues that all have the same absolute value, which we denote ρ_L.

Given an $\approx$-equivalence class L, define $\Phi_L : \mathbb{R}^n \times T(\boldsymbol{\gamma}) \to \mathbb{R}$ by[1]

$$\Phi_L(\boldsymbol{x}, \boldsymbol{\mu}) = \sum_{(i,j)\in L} h_{i,j}(\boldsymbol{x})\mu_j \,. \tag{3.3}$$

From the above definition of Φ_L we have

$$\sum_{(i,j)\in L} \binom{m}{i} \lambda_j^m h_{i,j}(\boldsymbol{x}) = \binom{m}{i_L} \rho_L^m \Phi_L(\boldsymbol{x}, \boldsymbol{\gamma}^m) \,. \tag{3.4}$$

for all $\boldsymbol{x} \in \mathbb{R}^n$ and all $m \in \mathbb{N}$.

We say that an $\approx$-equivalence class E of I is *dominant* for $\boldsymbol{x} \in \mathbb{R}^n$ if for all indices (i, j) belonging to an equivalence class $E' \succ E$ we have that $h_{i,j}(\boldsymbol{x})$ is identically zero. Equivalently, E is dominant for $\boldsymbol{x}$ if for all $E' \succ E$ we have that $\Phi_E(\boldsymbol{x}, \cdot)$ is identically zero on $T(\boldsymbol{\gamma})$. The equivalence of these two characterisations follows from the linear independence of the functions $\binom{m}{i}\lambda_j^m$ for $(i, j) \in E$.

[1] That the function Φ_L is real-valued follows from the fact that if eigenvalues λ_{j_1} and λ_{j_2} are complex conjugates then γ_{j_1} and γ_{j_2} are also complex conjugates, as are $h_{i,j_1}(\boldsymbol{z})$ and $h_{i,j_2}(\boldsymbol{z})$.

The following proposition shows how information about termination of Loop (3.2) on an initial value $\boldsymbol{x} \in \mathbb{R}^n$ can be derived from properties of $\Phi_E(\boldsymbol{x}, \cdot)$.

Proposition 3.2. Consider Loop (3.2). Let $\boldsymbol{x} \in \mathbb{R}^n$ and let E be an $\approx$-equivalence class that is dominant for $\boldsymbol{x}$. Then

1. If $\inf_{\boldsymbol{\mu} \in T(\boldsymbol{\gamma})} \Phi_E(\boldsymbol{x}, \boldsymbol{\mu}) > 0$ then $\boldsymbol{x}$ is eventually non-terminating.

2. If $\inf_{\boldsymbol{\mu} \in T(\boldsymbol{\gamma})} \Phi_E(\boldsymbol{x}, \boldsymbol{\mu}) < 0$ then $\boldsymbol{x}$ is terminating.

Proof. By Proposition 3.1 and (3.4) we have that for all $m \geq n$,

$$\begin{aligned} g(f^m(\boldsymbol{x})) &= \sum_{(i,j) \in I} \binom{m}{i} \lambda_j^m h_{i,j}(\boldsymbol{x}) \\ &= \binom{m}{i_E} \rho_E^m \Phi_E(\boldsymbol{x}, \boldsymbol{\gamma}^m) + \sum_{(i,j) \in I \setminus E} \binom{m}{i} \lambda_j^m h_{i,j}(\boldsymbol{x}) \,. \end{aligned} \tag{3.5}$$

Moreover by the dominance of E we have that

$$\lim_{m \to \infty} \frac{\binom{m}{i} |\lambda_j|^m}{\binom{m}{i_E} \rho_E^m} = 0 \tag{3.6}$$

for all $(i, j) \in I \setminus E$ such that $h_{i,j}(\boldsymbol{x}) \neq 0$.

We first prove Item 1. By assumption, in this case there exists $\varepsilon > 0$ such that $\Phi_E(\boldsymbol{x}, \boldsymbol{\mu}) \geq \varepsilon$ for all $\boldsymbol{\mu} \in T(\boldsymbol{\gamma})$. Together with (3.6), this shows that the asymptotically dominant term in (3.5) has positive sign. It follows that $g(f^m(\boldsymbol{x}))$ is positive for m sufficiently large and hence $\boldsymbol{x}$ is eventually non-terminating.

We turn now to Item 2. By assumption there exists $\varepsilon > 0$ and an open subset U of $T(\boldsymbol{\gamma})$ such that $\Phi_E(\boldsymbol{x}, \boldsymbol{\mu}) < -\varepsilon$ for all $\boldsymbol{\mu} \in U$. Moreover by density of $\{\boldsymbol{\gamma}^m : m \in \mathbb{N}\}$ in $T(\boldsymbol{\gamma})$ there exist infinitely many m such that $\boldsymbol{\gamma}^m \in U$. Exactly as in the previous case we can now use the dominance of E to conclude that $g(f^m(\boldsymbol{x})) < 0$ for sufficiently large m such that $\boldsymbol{\gamma}^m \in U$ and hence $\boldsymbol{x}$ is terminating. □

Given $\boldsymbol{z} \in \mathbb{Z}^n$, since $T(\boldsymbol{\gamma})$ is an algebraic subset of $\mathbb{T}^s$, the number $\inf_{\boldsymbol{\mu} \in T(\boldsymbol{\gamma})} \Phi_E(\boldsymbol{z}, \boldsymbol{\mu})$ is algebraic (by quantifier elimination) and its sign can be decided. Note however that Proposition 3.2 does not completely resolve the question of termination with respect to guard g from a given initial value $\boldsymbol{z}$. Indeed, let us define $\boldsymbol{z} \in \mathbb{R}^n$ to be *critical* if $\inf_{\boldsymbol{\mu} \in E} \Phi_E(\boldsymbol{z}, \boldsymbol{\mu}) = 0$, where E is the dominant $\approx$-equivalence class for $\boldsymbol{z}$. Then neither clause in the above proposition suffices to resolve termination of Loop (3.2) on such a $\boldsymbol{z}$.

In general, the question of whether a critical point is eventually non-terminating is equivalent to the *Ultimate Positivity Problem* for linear recurrence sequences: a longstanding and notoriously difficult open problem in number theory, only known to be decidable up to order 5 (Almagor *et al.*, 2018; Ouaknine and Worrell, 2014c). Fortunately in the setting of deciding loop termination we can sidestep such difficult questions. The following section is devoted to handling critical points. The idea is to show that if there is a non-terminating critical initial value then there is another initial value that is eventually non-terminating and whose eventual non-termination can be established by Proposition 3.2.

Example 3.4. Consider the loop:

$$\text{while } (w - z \geq 0) \text{ do} \quad \begin{pmatrix} w \\ x \\ y \\ z \end{pmatrix} \leftarrow \begin{pmatrix} -1 & 5 & 125 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} w \\ x \\ y \\ z \end{pmatrix}$$

The idea is that the variables (w, x, y) store consecutive values of the order-3 linear recurrence sequence

$$u_n = -u_{n-1} + 5u_{n-2} + 125u_{n-3}$$

while the variable z stores values of the sequence $v_n = 2v_{n-1}$.

The update matrix in the loop body has eigenvalues

$$\lambda_1 = 5,\ \lambda_2 = -3 + 4i,\ \lambda_3 = -3 + 4i,\ \lambda_4 = 2\,.$$

For $f : \mathbb{R}^4 \to \mathbb{R}^4$, the linear map computed in the loop body, and $g : \mathbb{R}^4 \to \mathbb{R}$, the map $g(w, x, y, z) = w - z$ in the loop guard, and for the initial value $\boldsymbol{x} = (18, 2, 2, 2)^\top$ we have

$$g(f^m(\boldsymbol{x})) = 5^m + \frac{1}{2}(-3 + 4i)^m + \frac{1}{2}(-3 - 4i)^m - 2 \cdot 2^m\,. \tag{3.7}$$

The first three eigenvalues form an $\approx$-equivalence class E with respect to the dominance preorder and together dominate the fourth eigenvalue. Normalising the eigenvalues to have length one we obtain

$$\gamma_1 := 1,\ \gamma_2 := \frac{-3+4i}{5},\ \gamma_3 := \frac{-3-4i}{5},\ \gamma_4 = 1\,.$$

Given the multiplicative relations $\gamma_1 = \gamma_4 = 1$ and $\gamma_2\gamma_3 = 1$, we have

$$T(\boldsymbol{\gamma}) = \left\{\boldsymbol{\mu} \in \mathbb{T}^4 : \mu_1 = \mu_4 = 1, \mu_2\mu_3 = 1\right\}\,.$$

The coefficients of the dominant eigenvalues in the exponential-sum expression (3.7) determine the map $\Phi_E(\boldsymbol{x}, \cdot) : T(\boldsymbol{\gamma}) \to \mathbb{R}$, leading to

$$\begin{aligned}\inf_{\boldsymbol{\mu}\in T(\boldsymbol{\gamma})} \Phi_E(\boldsymbol{x}, \boldsymbol{\mu}) &= \inf_{\boldsymbol{\mu}\in T(\boldsymbol{\gamma})} \mu_1 + \frac{1}{2}\mu_2 + \frac{1}{2}\mu_3 \\ &= \inf_{\mu\in\mathbb{T}} 1 + \frac{1}{2}\mu + \frac{1}{2}\overline{\mu} \\ &= 0\,.\end{aligned}$$

We conclude that $\boldsymbol{x}$ is a critical point.

Example 3.4 helps illustrate the idea that critical points are initial values for which termination involves considering all eigenvalues of the loop update map, not just the dominant eigenvalues. The initial value $(18, 2, 2, 2)^\top$ is eventually non-terminating if and only if the order-4 linear recurrence sequence (3.7) is ultimately positive: The sum of the three dominant terms in this expression is guaranteed to be non-negative, but establishing ultimate positivity of the whole expression would require a suitable lower bound on the contribution of the dominant terms. In the case at hand, ultimate positivity can be established using Baker's Theorem on linear forms in logarithms (Ouaknine and Worrell, 2014c). However, as noted above, in general it is not known to determine ultimate positivity of linear recurrences from order 6 onwards.

3.1.2 Non-Termination for a Single Guard Affine SLC Loop

In this section we continue to analyse termination of Loop (3.2), and refer to the notation established so far.

Non-Termination over the Reals and Rationals

The following definition encompasses both non-terminating and critical points:

Definition 3.1. For Loop (3.2), we define the set PN of *potentially non-terminating points* by

$$PN := \left\{ \boldsymbol{x} \in \mathbb{R}^n : \inf_{\boldsymbol{\mu} \in T(\boldsymbol{\gamma})} \Phi_E(\boldsymbol{x}, \boldsymbol{\mu}) \geq 0, \text{ where } E \text{ is dominant for } \boldsymbol{x} \right\} .$$

It is evident that PN is convex. The following proposition implies that PN is moreover an invariant of Loop (3.2), that is, if $\boldsymbol{x} \in PN$ then $f(\boldsymbol{x}) \in PN$.

Proposition 3.3. Let $\boldsymbol{x} \in \mathbb{R}^n$ and let $E \subseteq I$ be an $\approx$-equivalence class that is dominant for $\boldsymbol{x}$. Then E is also dominant for $f(\boldsymbol{x})$, and for all $\boldsymbol{\mu} \in T(\boldsymbol{\gamma})$ we have $\Phi_E(f(\boldsymbol{x}), \boldsymbol{\mu}) = \rho_E \, \Phi_E(\boldsymbol{x}, \boldsymbol{\gamma\mu})$, where the product $\boldsymbol{\gamma\mu}$ is defined pointwise.

Proof. By definition we have $\Phi_E(\boldsymbol{x}, \boldsymbol{\mu}) = \sum_{(i,j)\in E} h_{i,j}(\boldsymbol{x})\mu_j$, where the $h_{i,j}$ satisfy

$$(\boldsymbol{b}^\top \; c) \begin{pmatrix} A & \boldsymbol{a} \\ 0 & 1 \end{pmatrix}^m \begin{pmatrix} \boldsymbol{x} \\ 1 \end{pmatrix} = \sum_{(i,j)\in I} h_{i,j}(\boldsymbol{x}) \binom{m}{i} \lambda_j^m \tag{3.8}$$

for all $m \geq n$. Likewise we have $\Phi_E(f(\boldsymbol{x}), \boldsymbol{\mu}) = \sum_{(i,j)\in E} \widetilde{h}_{i,j}(\boldsymbol{x})\mu_j$, where the $\widetilde{h}_{i,j}$ satisfy

$$(\boldsymbol{b}^\top \; c) \begin{pmatrix} A & \boldsymbol{a} \\ 0 & 1 \end{pmatrix}^{m+1} \begin{pmatrix} \boldsymbol{x} \\ 1 \end{pmatrix} = \sum_{(i,j)\in I} \widetilde{h}_{i,j}(\boldsymbol{x}) \binom{m}{i} \lambda_j^m . \tag{3.9}$$

Combining (3.8) and (3.9) we have that for all $m \geq n$,

$$\begin{aligned} \sum_{(i,j)\in I} \widetilde{h}_{i,j}(\boldsymbol{x}) \binom{m}{i} \lambda_j^m &= \sum_{(i,j)\in I} h_{i,j}(\boldsymbol{x}) \binom{m+1}{i} \lambda_j^{m+1} \\ &= \sum_{(i,j)\in I} h_{i,j}(\boldsymbol{x}) \left[\binom{m}{i} + \binom{m}{i-1} \right] \lambda_j \lambda_j^m . \end{aligned}$$

Now the collection of functions $m \mapsto \binom{m}{i}\lambda_j^m$ for $(i,j) \in I$ is linearly independent (see Section 2.1.3). Equating the coefficients of the functions $\binom{m}{i}\lambda_j^m$ for $(i,j) \in E$ in the above equation we have $\widetilde{h}_{i,j} = \lambda_j h_{i,j} = \rho_E \gamma_j h_{i,j}$ for all $(i,j) \in E$; likewise we have that E is dominant for $f(\boldsymbol{x})$. The proposition follows. □

The next lemma is the key to the framework presented in this section. It shows that the non-emptiness of PN entails the existence of an eventually non-terminating point.

Lemma 3.1. If $\boldsymbol{z} \in PN$ then all points in the relative interior of $\texttt{conv}(\{f^m(\boldsymbol{z}) : m \in \mathbb{N}\})$ are eventually non-terminating.

Proof. Let E be the $\approx$-equivalence class that is dominant for $\boldsymbol{z}$. If $\Phi_E(\boldsymbol{z}, \cdot)$ is identically zero then by definition of dominance we must have that $\Phi_{E'}(\boldsymbol{z}, \cdot)$ is identically zero for all $\approx$-equivalence classes E'. By Proposition 3.3 we have that $\Phi_{E'}(f^m(\boldsymbol{z}), \cdot)$ is identically zero for all $\approx$-equivalence classes E' and all $m \in \mathbb{N}$. Hence $f^m(\boldsymbol{z})$ is eventually non-terminating for all $m \in \mathbb{N}$.

We thus suppose that $\Phi_E(\boldsymbol{z}, \cdot)$ is non-negative and not identically zero on $\boldsymbol{\mu} \in T(\boldsymbol{\gamma})$. Fix $\boldsymbol{\mu} \in T(\boldsymbol{\gamma})$. We claim that there exists $m \in \mathbb{N}$ such that $\Phi_E(f^m(\boldsymbol{z}), \boldsymbol{\mu}) > 0$. If this were not the case then by Proposition 3.3 for all $m \in \mathbb{N}$ we would have $\Phi_E(f^m(\boldsymbol{z}), \boldsymbol{\mu}) = \rho_E^m \, \Phi_E(\boldsymbol{z}, \boldsymbol{\gamma}^m \boldsymbol{\mu}) = 0$. But by Theorem 2.5, the set $\{\boldsymbol{\gamma}^m \boldsymbol{\mu} : m \geq 0\}$ is dense in $T(\boldsymbol{\gamma})$ and hence we would have that $\Phi_E(\boldsymbol{z}, \cdot)$ is identically 0 on $T(\boldsymbol{\gamma})$, contradicting our initial assumption. This establishes the claim.

By compactness of $T(\boldsymbol{\gamma})$ there exists $m_0 \in \mathbb{N}$ such that for all $\boldsymbol{\mu} \in T(\boldsymbol{\gamma})$ there exists $m \leq m_0$ such that $\Phi_E(f^m(\boldsymbol{z}), \boldsymbol{\mu}) > 0$.[2] By Proposition 2.1, for all points $\boldsymbol{x}$ lying in the relative interior of

$$\texttt{conv}(\{\boldsymbol{z}, f(\boldsymbol{z}), \ldots, f^{m_0}(\boldsymbol{z})\})$$

there exist $\alpha_0, \ldots, \alpha_{m_0} > 0$ such that: (i) $\sum_{m=0}^{m_0} \alpha_m = 1$; and (ii) $\boldsymbol{x} = \sum_{m=0}^{m_0} \alpha_i f^m(\boldsymbol{z})$. Since Φ_E is an affine map in its first variable, it follows that $\Phi_E(\boldsymbol{x}, \cdot) = \sum_{m=0}^{m_0} \alpha_m \Phi_E(f^m(\boldsymbol{z}), \cdot)$ is strictly positive on $T(\boldsymbol{\gamma})$. Hence $\boldsymbol{x}$ is eventually non-terminating by Proposition 3.2. □

[2]The use of compactness is not essential here. Using basic facts about linear recurrence sequences one can show that $m_0 = 2n + 1$ suffices.

The following Example illustrates Lemma 3.1.

Example 3.5. Consider the loop from Example 3.4. Starting from the critical point $\boldsymbol{x} := (18, 2, 2, 2)^\top$, after one execution of the loop body we arrive at $\boldsymbol{y} := (242, 18, 2, 4)^\top$. By Proposition 3.3 the point $\boldsymbol{y}$ is also critical. Consider the mid-point

$$\boldsymbol{z} := \frac{1}{2}(\boldsymbol{x} + \boldsymbol{y}) = (130, 10, 2, 3)^\top$$

between $\boldsymbol{x}$ and $\boldsymbol{y}$. We claim that $\boldsymbol{z}$ is eventually non-terminating. Indeed we have

$$\Phi_E(\boldsymbol{z}, \boldsymbol{\mu}) = \alpha_1\mu_1 + \alpha_2\mu_2 + \alpha_3\mu_3$$

where $\alpha_1, \alpha_2, \alpha_3$ are uniquely defined by the requirement that the sequence

$$v_n := \alpha_1 5^n + \alpha_2(-3 + 4i)^n + \alpha_3(-3 - 4i)^n$$

have initial values $v_0 = 2, v_1 = 10, v_2 = 130$, respectively. We thus obtain $\alpha_1 = 3$, $\alpha_2 := -\frac{1}{2} + i$, and $\alpha_3 := -\frac{1}{2} - i$. Since for $(\mu_1, \mu_2, \mu_3) \in T(\boldsymbol{\gamma})$ we have $\mu_1 = 1$ and $\mu_2 = \overline{\mu_3}$, we deduce that

$$\inf_{\mu \in T(\gamma)} \Phi_E(\boldsymbol{z}, \boldsymbol{\mu}) = \alpha_1 - 2|\alpha_2| = 3 - \sqrt{5} > 0\,.$$

It follows from Proposition 3.2 that $\boldsymbol{z}$ is eventually non-terminating.

From Lemma 3.1 we obtain the following effective criterion for non-termination over both $\mathbb{R}$ and $\mathbb{Q}$.

Corollary 3.2. Loop (3.2) is non-terminating over $\mathbb{R}$ if and only if PN is non-empty and is non-terminating over $\mathbb{Q}$ if and only if PN contains a rational point.

Proof. Given $\boldsymbol{z} \in PN$, all points in the relative interior of $\texttt{conv}(\{f^m(\boldsymbol{z}) : m \in \mathbb{N}\})$ are eventually non-terminating by Lemma 3.1. Hence the loop is non-terminating over $\mathbb{R}$. If moreover $\boldsymbol{z}$ is rational then the relative interior contains a rational point and hence the loop is non-terminating over $\mathbb{Q}$. □

Non-Termination over the Integers

We now refine the above analysis to obtain an effective criterion of the existence of *integer* non-terminating points. In particular, fixing an initial value $\boldsymbol{z}_0 \in \mathbb{Z}^n$, we show that for m sufficiently large, the set $\texttt{conv}(\{f^m(\boldsymbol{z}_0) : m \in \mathbb{N}\})$ contains an integer point in its relative interior. Recall that when considering termination over integers we consider that the coefficients of the functions f and g that define Loop (3.2) are integer.

Define $V := \texttt{aff}(\{f^m(\boldsymbol{z}_0) : m \in \mathbb{N}\})$ and let the vector subspace $V_0 \subseteq \mathbb{R}^n$ be the unique translate of V containing the origin. Write n_0 for the dimension of V_0 (equivalently the dimension of V).

Proposition 3.4. For all non-zero integer vectors $\boldsymbol{v} \in V_0$ the set $\{|\boldsymbol{v}^\top f^m(\boldsymbol{z}_0)| : m \in \mathbb{N}\}$ is unbounded.

Proof. Consider the sequence $x_m := \boldsymbol{v}^\top f^m(\boldsymbol{z}_0) = \boldsymbol{v}^\top \begin{pmatrix} A & \boldsymbol{a} \\ 0 & 1 \end{pmatrix}^m \begin{pmatrix} \boldsymbol{z}_0 \\ 1 \end{pmatrix}$. If this sequence were constant then $\boldsymbol{v}$ would be orthogonal to V_0, contradicting the fact that $\boldsymbol{v}$ is a non-zero vector in V_0. Since the sequence is non-constant, integer-valued, and satisfies a non-degenerate linear recurrence of order at most $n+1$ (see, *e.g.*, Everest *et al.* (2003, Section 1.1.12)), by the Skolem-Mahler-Lech Theorem we have that $\{|\boldsymbol{v}^\top f^m(\boldsymbol{z}_0)| : m \in \mathbb{N}\}$ is unbounded (see the discussion of growth of linear recurrence by Everest *et al.* (2003, Section 2.2)).[3] □

Proposition 3.5. Given $\boldsymbol{z}_0 \in \mathbb{Z}^n$, the set $\texttt{conv}(\{f^m(\boldsymbol{z}_0) : m \in \mathbb{N}\})$ contains an integer point in its relative interior.

Proof. Since V_0 is spanned by integer vectors, $\Lambda := V_0 \cap \mathbb{Z}^n$ is a lattice of rank n_0 in $\mathbb{R}^n$. Define $C := \texttt{conv}(\{f^m(\boldsymbol{z}_0) : m \in \mathbb{N}\}) \subseteq V$ and

[3] The above argument actually establishes that $\langle x_m : m \in \mathbb{N}\rangle$ diverges to infinity in absolute value. We briefly sketch a more elementary proof of mere unboundedness. If the sequence $\langle x_m : m \in \mathbb{N}\rangle$ were bounded then by van der Waerden's Theorem, for all m' it would contain a constant subsequence of the form $x_\ell, x_{\ell+p}, \ldots, x_{\ell+m'p}$ for some $\ell, p \geq 1$. In particular, if $m' = n$ then since every infinite subsequence $y_m := x_{\ell+pm}$ satisfies a linear recurrence of order at most $m+1$, $\langle x_m : m \in \mathbb{N}\rangle$ would have an infinite constant subsequence $\langle x_{\ell+pm} : m \in \mathbb{N}\rangle$. If $p = 1$ then $\langle x_m : m \in \mathbb{N}\rangle$ is constant and if $p > 1$ then by Salomaa and Soittola (1978, Lemma 9.11) $\langle x_m : m \in \mathbb{N}\rangle$ is degenerate.

$C_0 := C - f^n(\boldsymbol{z}_0) \subseteq V_0$. We may assume that $n_0 \geq 1$ since otherwise V is a singleton, *i.e.*, $\boldsymbol{z}_0$ is a fixed point of f and the proposition is vacuously true (here, note that a singleton set is its own relative interior).

Let $\theta : \mathbb{R}^n \to V_0$ be the orthogonal projection of $\mathbb{R}^n$ onto V_0. Then $\theta(\Lambda)$ is a lattice in V_0 of full rank. We claim that the lattice width of $\theta(C_0)$ with respect to $\theta(\Lambda)$ is infinite. Indeed for any non-zero vector $\boldsymbol{v} \in \theta(\Lambda)$ we have

$$\boldsymbol{v}^\top(\theta(f^m(\boldsymbol{z}_0)) - \theta(f^n(\boldsymbol{z}_0))) = \boldsymbol{v}^\top(f^m(\boldsymbol{z}_0) - f^n(\boldsymbol{z}_0)), \tag{3.10}$$

But $\boldsymbol{v}$ is a non-zero vector in V_0 with rational coefficients and hence Proposition 3.4 entails that the absolute value of (3.10) is unbounded as m runs over $\mathbb{N}$. Since V_0 has positive dimension, this proves the claim.

Since $\theta(C_0)$ is a full-dimensional convex subset of $\mathbb{R}^{n_0}$, by Theorem 2.1 we have that $\theta(C_0)$ contains a point of $\theta(\Lambda)$ in its relative interior and hence C_0 contains a point of Λ (necessarily an integer point) in its relative interior. Since C is the translation of C_0 by an integer vector, we conclude that C also contains an integer point in its relative interior. □

The following theorem characterises when an affine *SLC* loop with a single guard is terminating over the integers.

Theorem 3.3. Loop (3.2) is non-terminating on $\mathbb{Z}$ if and only if the set PN contains an integer point $\boldsymbol{z}$.

Proof. If no such $\boldsymbol{z}$ exists then the loop is terminating by Proposition 3.2.(2). Conversely, if such a $\boldsymbol{z}$ exists then the loop is non-terminating by Lemma 3.1 and Proposition 3.5. □

We postpone the question of the effectiveness of the above characterisation until we handle loops with multiple guards.

3.1.3 Multiple Guards

Next we present a decision procedure for a general affine *SLC* loop

$$\mathcal{Q} : \textit{while } (g_1(\boldsymbol{x}) \geq 0 \wedge \ldots \wedge g_m(\boldsymbol{x}) \geq 0) \textit{ do } \boldsymbol{x} := f(\boldsymbol{x}), \tag{3.11}$$

with multiple guards. Associated to Loop (3.11) we consider m single-guard loops with a common update function:

$$\mathcal{Q}_i \ : \ while \ (g_i(\boldsymbol{x}) \geq 0) \ do \ \boldsymbol{x} := f(\boldsymbol{x}) \, ,$$

for $i = 1, \ldots, m$. Clearly Loop (3.11) non-terminating if and only if there exists $\boldsymbol{z} \in \mathbb{Z}^n$ such that each loop $\mathcal{Q}_i$ is non-terminating on $\boldsymbol{z}$.

Theorem 3.4. Let PN_i be the set of potentially non-terminating points for each loop $\mathcal{Q}_i$ for $i \in \{1, \ldots, m\}$ and write $PN := \bigcap_{i=1}^m PN_i$. Then loop $\mathcal{Q}$ of (3.11) is non-terminating over $\mathbb{R}$ if and only if PN is non-empty and $\mathcal{Q}$ is non-terminating over $\mathbb{Q}$ if and only if PN contains a rational point. If all numerical constants in $\mathcal{Q}$ are integer then the loop is non-terminating over $\mathbb{Z}$ if and only if PN contains an integer point.

Theorem 3.4 leads to the following procedure for deciding termination of a given affine SLC loop $\mathcal{Q}$, as shown in (3.11), over a ring $R \in \{\mathbb{R}, \mathbb{Q}, \mathbb{Z}\}$:

1. Compute the non-zero eigenvalues $\lambda_1, \ldots, \lambda_s$ of the matrix corresponding to the loop update function, as given in (3.1). Let $\gamma_i := \frac{\lambda_i}{|\lambda_i|}$ for $i \in \{1, \ldots, s\}$.

2. Compute the dominance preorder $\preccurlyeq$ among eigenvalues.

3. Compute a basis of the group $L(\boldsymbol{\gamma})$ of multiplicative relations among $\gamma_1, \ldots, \gamma_s$.

4. Compute the set PN_i of potentially non-terminating points for each loop $\mathcal{Q}_i$ using steps 2 and 3.

5. Return "non-terminating" if $PN := \bigcap_{i=1}^m PN_i$ contains a point with all coordinates in R and otherwise return "terminating".

We briefly discuss the effectiveness of each step. Step 1 involves computing the roots of an integer polynomial. These can be represented by rational approximations of sufficient accuracy to distinguish the roots from each other. (The required accuracy is determined by standard polynomial root separation bounds.) Such approximations can be computed in polynomial time in the loop description. These approximations can be used to determine the dominance preorder in Step 2. Step 3 can be accomplished in polynomial time using the algorithm of (Combot, 2025). Thus Steps 1-3 can be carried out in polynomial time in the size of the linear loop. For Step 4 we describe the semi-algebraic set PN, as given in Definition 3.1, by a polynomial-size formula of first-order logic of with two quantifier alternations. Whether such a set contains a real, rational, or integer point can be decided in exponential time in the size of the formula (Khachiyan and Porkolab, 1997, Theorem 1.1). Thus the overall running time of the procedure above is exponential in the size of the input linear loop.

We have thus established the main result of this section:

Theorem 3.5. There is a procedure to decide termination of affine *SLC* loops over $\mathbb{R}$, $\mathbb{Q}$, and $\mathbb{Z}$.

As a final comment, we note that all results presented in this section hold also when the loop guard involve strict inequalities.

3.1.4 Termination with Respect to Initial States

There are not many results on the termination of an affine *SLC* loop with respect to a given initial state (or set of initial states). This is likely because the problem is very difficult; it subsumes *Positivity Problem* for linear recurrence sequences (*e.g.*, see Kenison *et al.*, 2023). This is the problem of determining whether all terms in a given integer linear recurrence sequence are positive. Decidability of the Positivity Problem is a longstanding open problem (going back at least as far as the 1970s (Rozenberg and Salomaa, 1994; Soittola, 1975)), and results by Ouaknine and Worrell (2014c) suggest that a solution to the problem will require significant breakthroughs in number theory.

While decidability of the positivity problem is still open for the general case, partial solutions for some special cases exist (Ouaknine and Worrell, 2014b; Ouaknine and Worrell, 2014a; Akshay *et al.*, 2017; Kenison *et al.*, 2023). Thus, the halting problem (termination wrt. a single initial state) for any subclass of integer affine *SLC* loops whose corresponding recurrence sequences fall in these special cases, is decidable. For example, Ouaknine and Worrell (2014b) show that the positivity problem is decidable for recurrences of order 5 or less, which implies decidability of the halting problem for integer affine SLC loops with at most 4 variables (we need an extra variable to eliminate the constants in the guard and the update). Kincaid *et al.* (2019) show decidability of the halting problem for integer affine SLC loops where every eigenvalue of the update matrix is a radical of a rational number.

Bozga *et al.* (2014) show that for integer affine SLC loops whose update matrix generates a finite monoid, the set of non-terminating initial values is definable in Presburger arithmetic and can be computed effectively. Thus, termination of such loops wrt. a set of initial of states that is definable in this arithmetic, is decidable.

Hark *et al.* (2020) show that the halting problem is decidable for affine SLC loops with a triangular update matrix, over any ring $\mathbb{Z} \subseteq R \subseteq \mathbb{R}_{\mathbb{A}}$ (where $\mathbb{R}_{\mathbb{A}}$ is the ring of algebraic real numbers). Their results go beyond simple linear loops, as they allow the loop condition to be any Boolean formula over atoms of the form $p(\boldsymbol{x}) \geq 0$ or $p(\boldsymbol{x}) > 0$, and the update can also include polynomial assignments that respect the triangular condition, which means that x_i does not depend on x_j for $j < i$, and x_i depends linearly on itself.

The core idea is that the truth value of the condition always stabilises after some iterations, and since such loops have (computable) closed forms, a bound on the number of iterations to stabilisation can be computed.

A method for computing a subset of the non-terminating initial states for affine SLC loops over the real numbers was presented by Li (2017). For linear homogeneous loops with only two program variables (and a strict inequality in the guard), Dai and Xia (2012) provided a complete algorithm to compute the full set of non-terminating initial states.

OPEN PROBLEMS 1. Is termination of affine SLC loops wrt. to an initial value, or a (polyhedral) set of initial states, over $\mathbb{R}$, $\mathbb{Q}$ or $\mathbb{Z}$ decidable?

3.1.5 Other Results Related to Affine *SLC* Loops

Li (2014) gave an alternative algorithm to decide termination of linear programs over $\mathbb{R}$. Whereas the approach of Tiwari (2004) and Braverman (2006) is based on searching for eventually non-terminating initial values, Li's algorithm outputs, in the case of non-termination, a genuinely non-terminating initial value.

Xia *et al.* (2011) show that the decision procedure of Tiwari (2004) suffers from imprecision when implemented using floating-point arithmetic (to compute Jordan forms), and they fix this imprecision by developing a symbolic implementation.

Frohn and Giesl (2019) showed decidability of termination of linear loops over $\mathbb{Z}$ under the assumption that the loop update matrix is upper-triangular, that is, all elements below the main diagonal are zero. Hark *et al.* (2025) extend the approach to loops with nonlinear updates (which is beyond the scope of this survey), but they also generalise the loop guard to be any Boolean combination of inequalities (*i.e.*, not necessarily a convex polyhedron), while still showing decidability over $\mathbb{R}$ and $\mathbb{R}_{\mathbb{A}}$ (the ring of algebraic real numbers). Moreover, in the same work, they consider affine loop where the update matrix has rational spectrum, and show that its termination, over either the integers, rational numbers or algebraic reals is `coNP`-complete. In the more general case of matrices with a real spectrum, they show that termination over the algebraic reals is $\forall\mathbb{R}$-complete; this class includes problems reducible to validity of a universally quantified formula of polynomial inequalities over the reals, and is contained in `PSPACE`.

Zhu and Kincaid (2021) explore how techniques for proving termination of affine *SLC* loops can be used to prove termination of more realistic programs.

Using techniques that ultimately rely on the p-adic Subspace Theorem in Diophantine approximation, Ouaknine *et al.* (2015) gave an effective characterisation of the set of all *eventually non-terminating points*[4] for affine *SLC* loops whose update matrix is diagonalisable.

[4] A point is eventually non-terminating if it evolves into a non-terminating point after a finite number of iterations of the loop body, disregarding the loop guard. The problem of determining whether a given point is eventually non-terminating for a

This suffices to decide whether such a loop terminates over the integers. In contrast, the method presented in this section solves the termination problem without giving an effective characterisation of all non-terminating points (or eventually non-terminating points).

3.2 Termination of Single-path Linear-Constraint Loops

The case of general *SLC* loops constitutes an important open problem:

OPEN PROBLEM 2. Is termination of *SLC* loops, with rational or equivalently integer coefficients, over $\mathbb{R}$, $\mathbb{Q}$, or $\mathbb{Z}$ decidable?

Attempts to solve this problem have lead to results for special cases or extensions of *SLC* loops. Next we overview these results.

Ben-Amram *et al.* (2012) considered *SLC* loops where irrational coefficients are allowed (recall that *SLC* loops, as defined in Section 2.3.2, involve only rational coefficients).

Theorem 3.6. Termination of *SLC* loops, where the coefficients are from $\mathbb{Z} \cup \{r\}$, for a single arbitrary *irrational* constant $r \in \mathbb{R}$, and variables range over integers, is undecidable.

The proof of this result shows that such loops can simulate a counter program. The key idea is to use linear constraints that involve r as a coefficients to simulate the instruction $x_j = \mathit{isPositive}(x_i)$, where *isPositive* returns 1 if $x_i > 0$ and 0 otherwise.

Ben-Amram *et al.* (2012) show that Petri nets can be simulated using integer *SLC* loops, and thus provide an `EXPSPACE` lower-bound on the hardness of proving termination of integer *SLC* loops wrt. to polyhedral set of initial states, even for deterministic *SLC* loops. For nondeterministic *SLC* loop, a similar reduction from Ben-Amram, 2014 proves that termination with a polyhedral set of initial states is `Ackermann`-hard (based on recent results on the hardness of reachability in Vector Addition Systems (Czerwinski and Orlikowski, 2021; Leroux, 2021)).

given loop is equivalent to the *Ultimate Positivity Problem* for linear recurrence sequence. This asks to determine whether all but finitely many terms in a given linear recurrence sequence are positive.

Bozga *et al.* (2014) consider octagonal *SLC* loops, a special case of *SLC* loops where the transition polyhedron is defined by inequalities of the form $\pm x \leq c$ or $\pm x \pm y \leq c$. They prove that termination over the integers is decidable in polynomial time, a result that also holds for the rationals and reals. Furthermore, for loops that do not terminate universally, they can compute a weakest precondition to non-termination, which is definable in Presburger arithmetic.

Guilmant *et al.* (2024) consider *SLC* loops but in two dimensions only (*i.e.*, two variables) and prove that termination is decidable.

3.3 Termination of Multi-path Linear-Constraint Loops

Tiwari (2004) observed that termination of *MLC* loops, and therefore of general CFGs, is undecidable over $\mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{R}$.

Theorem 3.7. The termination problem, with and without initial states, is undecidable for *MLC* loops, over $\mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{R}$.

This undecidability is shown even for *MLC* loops where every path is defined by an affine *SLC* loop and the paths are mutually exclusive, making the *MLC* loop deterministic. This is demonstrated by a reduction from counter programs, where a counter program with n counters is translated to an *MLC* loop with n counter variables and a location variable pc, as follows:

- Increment or decrement of counter X_i at location j generates the path $\{pc = j, x_i' = x_i \pm 1, pc' = j + 1\}$; and
- Conditional statement "*if* $X_i > 0$ *then* k_1 *else* k_2" at location j generates the paths $\{pc = j, x_i \geq 1, pc' = k_1\}$ and $\{pc = j, x_i \leq 0, pc' = k_2\}$.

This reduction implies that termination of integer *MLC* loops, with and without initial states, is undecidable over $\mathbb{Z}$. For undecidability over $\mathbb{R}$ and $\mathbb{Q}$, Tiwari (2004) observes that the generated *MLC* loop is terminating over $\mathbb{Z}$ if and only if it is terminating over $\mathbb{R}$ and $\mathbb{Q}$. Furthermore, due to Theorem 2.6, undecidability already hold for 3 variables.

Ben-Amram *et al.* (2012) show that undecidability already holds when restricting the *MLC* loop to 2 paths where each is an affine *SLC* loop.

Theorem 3.8. The termination problem, with and without initial set of states, is undecidable for loops of the following form

$$\textit{while } (B\boldsymbol{x} \geq \boldsymbol{b}) \textit{ do } \boldsymbol{x} := \begin{cases} A_0\boldsymbol{x} & x_i \leq 0 \\ A_1\boldsymbol{x} & x_i > 0 \end{cases}$$

where the state vector $\boldsymbol{x}$ ranges over $\mathbb{Z}^n$, $A_0, A_1 \in \mathbb{Z}^{n\times n}$, $\boldsymbol{b} \in \mathbb{Z}^p$ for some $p > 0$, $B \in \mathbb{Z}^{p\times n}$, and $x_i \in \boldsymbol{x}$.

The proof of this result is by a reduction from 2-counter programs. Another restricted form of *MLC* loop for which termination is known to be undecidable is a deterministic loop in two variables, of the form

$$\textit{while } (x_1 + x_2 > 0) \textit{ do } (x_1, x_2) := f(x_1, x_2)$$

where f is piecewise-affine, whose pieces are defined by linear inequalities (thus defining the paths of the *MLC* loop). The termination of such loops is undecidable over the rationals and reals (Blondel *et al.*, 2001) as well as over integers (Ben-Amram, 2015).

We note however that Tiwari observes that the decidability of termination of linear loops allows us to decide the termination of multi-path loops in the following favourable case. Let us denote, as in Section 2.3.2, the paths of the loop as transition polyhedra $\mathcal{Q}_1, \ldots, \mathcal{Q}_k$, and consider each $\mathcal{Q}_i$ as a binary relation on $\mathbb{R}^n$ (respectively, $\mathbb{Q}^n$, $\mathbb{Z}^n$), so that $\mathcal{Q}_i \circ \mathcal{Q}_j$ denote the composition of relations.

Theorem 3.9. Let $\mathcal{Q}_1, \ldots, \mathcal{Q}_k$ be a *MLC* loop over the reals (respectively, the rationals or integers). Let $T = \bigcup_i \mathcal{Q}_i$ be the set of all loop transitions. Assume that whenever $i < j$, it is the case that $\mathcal{Q}_j \circ \mathcal{Q}_i \subseteq \mathcal{Q}_i \circ T^*$. Then, the *MLC* loop terminates if and only if each $\mathcal{Q}_i$ does.

4

Ranking Functions

The use of ranking functions to prove termination goes back to Turing (1948) and was subsequently popularised by Floyd (1967).

Definition 4.1. Let $T \subseteq S \times S$ be a transition relation, $S_0 \subseteq S$ a set of initial states, T_{S_0} the restriction of T to the reachable states $\mathsf{RCH}(T, S_0)$, and $\langle W, \preceq \rangle$ a partially ordered set such that $\preceq$ is well-founded. We say that $\rho : S \to W$ is a ranking function for T wrt. S_0, if for every $(s, s') \in T_{S_0}$, $\rho(s) \succ \rho(s')$, where $\succ$ is the strict order relation on W.

Note that if $S_0 = S$ then $T_{S_0} = T$, a fact used when we consider universal termination.

The fact that ρ proves termination of T wrt. the set of initial states S_0 is immediate from the definition: a non-terminating computation staring in $s_0 \in S_0$ would yield an infinite descending chain in W, contradicting the well-foundedness assumption. On the other hand, every terminating transition relation wrt. the set of initial states S_0 has a ranking function. Let $W = \mathsf{RCH}(T, S_0) \cup \{\bot\}$, ordered by the reachability relation with a least element $\bot$, and let $\rho(s) = s$ if $s \in \mathsf{RCH}(T, S_0)$, otherwise $\rho(s) = \bot$.[1]

[1]There is some room for explanation regarding whether W is partially or totally ordered. Our statement is easy to see if partial orders are allowed, but also holds if total orders are required, since the partial order can be extended to a total one.

The last observation shows that to obtain practical methods for proving termination one must restrict the search to a specific class of ranking functions, otherwise the problem is as hard as termination itself. Clearly, the choice of the class determines the decidability and computational complexity of the resulting decision problems.

In this section, we are concerned with ranking functions that are based on linear combinations of state variables, for the different kinds of programs defined in Section 2.3, and with or without restricting the initial states, *i.e.*, termination and universal termination.

We begin, in Section 4.1, with *linear ranking functions* (LRFs); we discuss the complexity of finding such ranking functions in various settings. Then in Section 4.2 we discusses *lexicographic-linear ranking functions* (LLRFs). This kind of ranking function appeared in the literature in various variants, and our goal in this survey is to present multiple variants in a unified manner as much as possible. Finally, Section 4.3 lists some references regarding other kinds of ranking functions, which we do not expand upon.

4.1 Linear Ranking Functions

In this section we survey algorithmic and complexity aspects of linear ranking functions (briefly, LRFs) for *SLC* loops, *MLC* loops, and the general case of CFGs. The domain of program variables is assumed, by default, to be the rationals, but all results apply also to the case of real valued variables. The integer case is discussed separately. For each case, we first consider termination without any assumption on the input values, *i.e.*, universal termination, and then treat the case when a polyhedral set of initial states is given.

Recall that an affine linear function $\rho : \mathbb{Q}^n \to \mathbb{Q}$ is a function of the form $\rho(\boldsymbol{x}) = \vec{\lambda}\boldsymbol{x} + \lambda_0$, where $\vec{\lambda} \in \mathbb{Q}^n$ is a row vector and $\lambda_0 \in \mathbb{Q}$. For such a function, and a transition $\boldsymbol{x''} = \binom{\boldsymbol{x}}{\boldsymbol{x}'}$, we write $\Delta\rho(\boldsymbol{x''})$ for the difference $\rho(\boldsymbol{x}) - \rho(\boldsymbol{x}')$.

Definition 4.2 (LRF)**.** Given a rational *MLC* loop $\mathcal{Q}_1, \ldots, \mathcal{Q}_k \subseteq \mathbb{Q}^{2n}$, we say that an affine linear function ρ is an LRF for the loop if the following hold for every $\boldsymbol{x}'' \in \mathcal{Q}_1 \cup \cdots \cup \mathcal{Q}_k$:

$$\rho(\boldsymbol{x}) \geq 0\,, \tag{4.1}$$

$$\Delta\rho(\boldsymbol{x}'') \geq 1\,. \tag{4.2}$$

Remark 4.1. Note that the co-domain of ρ is $\mathbb{Q}$ which is not well-founded under the usual order. However it is easy to see that such a function proves termination, and it can be converted to match Definition 4.1 by considering $\max(0, \lceil \rho + 1 \rceil) : \mathbb{Q}^n \to \mathbb{N}$. Such a consideration will apply to all the following definitions which are based on this one.

Remark 4.2. We could replace (4.2) with $\Delta\rho(\boldsymbol{x}'') \geq \delta$ for an arbitrary constant $\delta > 0$. Indeed, it suffices to multiply ρ by $1/\delta$ to obtain the original condition of Definition 4.2. This is again an observation that we will take for granted when considering variants of this definition.

Remark 4.3. When considering integer loops, we can use a strict inequality $\Delta\rho(\boldsymbol{x}'') > 0$ instead of (4.2), because we may assume that ρ used integer coefficients. This change is not obviously safe when dealing with the rationals, so when we do use the strict inequality, we refer cautiously to a *weak* ranking function (versus a *strict* one). Interestingly, in the case of LRF and loops given by polyhedra, it is easy to prove that a weak LRF is also a strict one, due to the fact that a bounded LP minimisation problem always attains its minimum (thus if $\Delta\rho(\boldsymbol{x}'') > 0$ holds over $\mathcal{Q}$, then there is $\delta > 0$ such that $\Delta\rho(\boldsymbol{x}'') \geq \delta$ holds as well).

The rest of this section is structured as follows: Sections 4.1.1 and 4.1.2 review results on the LRF problem for rational and integer *SLC* loops, respectively; Section 4.1.2 reviews results on the LRF problem for *MLC* loops; Section 4.1.4 reviews results on the LRF problem for CFGs; Section 4.1.5 provides a historical perspective on the LRF problem; and finally, Section 4.1.6 concludes. Table 4.1 summarises the results that we present in this Section.

4.1.1 LRFs Over the Rationals for *SLC* Loops

In what follows we assume a given *SLC* loop, specified by a transition polyhedron $\mathcal{Q} \subseteq \mathbb{Q}^{2n}$. When variables range over the rationals,

Domain	LRF	$\text{LRF}_{\mathcal{S}_0}$
$\mathbb{R}$	`PTIME`	`PSPACE`-hard
$\mathbb{Q}$	`PTIME`	`PSPACE`-hard
$\mathbb{Z}$	`coNP`-complete	`Ackermann`-hard

Table 4.1: Complexity of deciding existence of LRFs (over $\mathbb{R}$, $\mathbb{Q}$, and $\mathbb{Z}$) for *SLC* loops, *MLC* loops, and CFGs (with and without initial states).

there is an algorithm to find LRFs which is *complete* (always finds an LRF if there is one) and has *polynomial time complexity*. This algorithm is based on seeking inequalities of the form (4.1,4.2) that are entailed by the transition polyhedron $\mathcal{Q}$, which can be done using Farkas' Lemma. Specifically, this approach involves turning the conditions for an LRF (4.1,4.2) into a set of linear constraints where the variables are the coefficients of ρ, and then solving these constraints using an LP algorithm to find values for the coefficients, if possible. Next we explain the details of such an algorithm.

Let us write $\rho(\boldsymbol{x})$ as $\vec{\lambda}\boldsymbol{x} + \lambda_0$, where $\vec{\lambda} \in \mathbb{Q}^n$ is a row vector and $\lambda_0 \in \mathbb{Q}$. Recall that the transition polyhedron can be specified as $A''\boldsymbol{x}'' \leq \boldsymbol{c}''$; then we have the deduction problem (the entailed inequalities are rewritten to use $\leq$ instead of $\geq$):

$$\begin{array}{ccc l} A''\boldsymbol{x}'' & \leq & \boldsymbol{c} & \\ \hline -\vec{\lambda}\boldsymbol{x} - \vec{0}\boldsymbol{x} & \leq & -\lambda_0 & \text{– obtained from (4.1)} \\ -\vec{\lambda}\boldsymbol{x} + \vec{\lambda}\boldsymbol{x}' & \leq & -1 & \text{– obtained from (4.2)} \end{array}$$

Using Farkas' Lemma (see Section 2.2.5), synthesising the two entailed inequalities can be done by solving the following LP problem, where $\vec{\mu}, \vec{\eta}$ are (row) vectors of variables representing the Farkas' coefficients, and $\vec{\lambda} \in$ and λ_0 are rational variables representing the coefficients and constant of ρ:

$$\vec{\mu}A'' = (-\vec{\lambda}, \vec{0}),\ \vec{\mu}\boldsymbol{c} \leq -\lambda_0,\ \vec{\mu} \geq 0 \tag{4.3}$$

$$\vec{\eta}A'' = (-\vec{\lambda}, \vec{\lambda}),\ \vec{\eta}\boldsymbol{c} \leq -1,\ \vec{\eta} \geq 0 \tag{4.4}$$

Any solution of (4.3,4.4) over the reals (or rationals) defines a corresponding LRF, and any LRF yields a corresponding solution to (4.3,4.4).

Example 4.1. Consider the *SLC* loop:

$$while\ (x_1 \geq 0, x_2 \geq 1)\ do\ x'_1 \leq x_1 - x_2, x'_2 \geq x_2 \tag{4.5}$$

and its corresponding matrix representations $A''\boldsymbol{x} \leq \boldsymbol{c}''$ where

$$A'' = \begin{pmatrix} x_1 & x_2 & x'_1 & x'_2 \\ \hline -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ -1 & 1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix} \qquad \boldsymbol{c}'' = \begin{pmatrix} 0 \\ -1 \\ 0 \\ 0 \end{pmatrix}$$

Let $\rho(x_1, x_2) = \lambda_1 x_1 + \lambda_2 x_2 + \lambda_0$ be an LRF template, *i.e.*, λ_i are unknowns, $\vec{\mu} = (\mu_0, \ldots, \mu_3)$ and $\vec{\eta} = (\eta_0, \ldots, \eta_3)$. To synthesise an LRF for loop (4.5), we first use (4.3,4.4) to generate the constraint system

$$\begin{array}{l} -\mu_0 - \mu_2 = -\lambda_1,\ -\mu_1 + \mu_2 + \mu_3 = -\lambda_2,\ \mu_2 = 0, -\mu_3 = 0 \\ -\mu_1 \leq -\lambda_0,\ \mu_0 \geq 0,\ \mu_1 \geq 0,\ \mu_2 \geq 0, \mu_3 \geq 0 \\ -\eta_0 - \eta_2 = -\lambda_1,\ -\eta_1 + \eta_2 + \eta_3 = -\lambda_2,\ \eta_2 = \lambda_1, -\eta_3 = \lambda_2 \\ -\eta_1 \leq -1,\ \eta_0 \geq 0,\ \eta_1 \geq 0,\ \eta_2 \geq 0, \eta_3 \geq 0 \end{array} \tag{4.6}$$

The constraints in the first 2 lines come from (4.3), and the last 2 lines from (4.4). The following is a possible solution for (4.6)

$$\begin{array}{l} \lambda_0 \mapsto 0,\ \lambda_1 \mapsto 1,\ \lambda_2 \mapsto 0, \\ \mu_0 \mapsto,\ \mu_1 \mapsto 0\ \mu_2 \mapsto 0\ \mu_3 \mapsto 0, \\ \eta_0 \mapsto 0,\ \eta_1 \mapsto 1\ \eta_2 \mapsto 1\ \eta_3 \mapsto 0 \end{array} \tag{4.7}$$

which means that $\rho(x_1, x_2) = x_1$ is an LRF for (4.5).

Podelski and Rybalchenko (2004a) simplified (4.3,4.4) using the fact that $A'' = (A\ A')$ for some matrices A, A' with n columns each, to the following equivalent one (they eliminate $\vec{\lambda}$ and λ_0 to reduce the number of variables for efficiency):

$$\begin{aligned} \vec{\mu}A' &= \vec{0}, \\ (\vec{\mu} - \vec{\eta})A &= \vec{0}, \\ \vec{\eta}(A + A') &= \vec{0}, \\ \vec{\eta}\boldsymbol{c} &\leq -1, \\ \vec{\mu}, \vec{\eta} &\geq \mathbf{0}. \end{aligned} \tag{4.8}$$

Solving (4.8) answers the existence question (*i.e.*, if (4.8) has a solution then an LRF exists) and furthermore, the LRF coefficients can be computed as $\vec{\lambda} = -\vec{\mu}A$ and λ_0 can be any value satisfying $\vec{\mu}\boldsymbol{c} \leq \lambda_0$ (in particular $\lambda_0 = \vec{\mu}\boldsymbol{c}$).

Theorem 4.1 **((Podelski and Rybalchenko, 2004a)).** An *SLC* loop $\mathcal{Q}$, specified by $A''\boldsymbol{x}'' \leq \boldsymbol{c}''$, has an LRF if and only if the linear program (4.8) has a solution.

Let us now consider the case in which we seek an LRF wrt. to a polyhedral set of initial states $\mathcal{S}_0 \subseteq \mathbb{Q}^n$. We refer to such LRF as $\text{LRF}_{\mathcal{S}_0}$. As we have mentioned in Section 2.3, it is enough to consider the universal termination of $\mathcal{Q}_{\mathcal{S}_0}$ instead of termination of $\mathcal{Q}$ wrt. to $\mathcal{S}_0$.

Example 4.2. Consider the *SLC* loop $\mathcal{Q} = \{x \geq 0, x' \leq x - y, y' \geq y + 1\}$, and note that $\max(0, x+1)$ is a ranking function, according to Definition 4.1 when restricting the initial states to $\mathcal{S}_0 = \{y = 1\}$. However, $\mathcal{Q}$ does not have an LRF according to Definition 4.2, unless we apply it to $\mathcal{Q}_{\mathcal{S}_0} = \{\, y \geq 1\,, x \geq 0, x' \leq x' - y, y \geq y + 1\}$ instead of $\mathcal{Q}$, which then admits $\rho(x, y) = x$ as an LRF.

This example suggests the following approach for seeking LRFs for loops with initial states: (1) compute the set of reachable states $\mathsf{RCH}(\mathcal{Q}, \mathcal{S}_0)$ and use it to compute $\mathcal{Q}_{\mathcal{S}_0}$; and (2) seek an LRF for $\mathcal{Q}_{\mathcal{S}_0}$. However, there is a problem with this approach: we do not know, in general, how to compute (or even express) the set of reachable states, and it is certainly not guaranteed to be polyhedral. To address this in practice, we over-approximate $\mathsf{RCH}(\mathcal{Q}, S_0)$ using a polyhedral invariant $\mathcal{I}(\boldsymbol{x})$ (called a *supporting invariant*) and then analyse the transition relation $\mathcal{Q}' = \mathcal{Q}(\boldsymbol{x}, \boldsymbol{x}') \wedge \mathcal{I}(\boldsymbol{x})$. This sacrifices completeness because $\mathcal{Q}'$ is an over-approximation of $\mathcal{Q}_{\mathcal{S}_0}$.

Polyhedral invariants (more precisely, inductive polyhedral invariants) can be inferred either beforehand using dedicated tools (Cousot and Halbwachs, 1978), or by using a *template-based* approach (Colón *et al.*, 2003; Bradley *et al.*, 2005a; Larraz *et al.*, 2013) to synthesise an LRF and a supporting polyhedral invariant simultaneously. This has the advantage that the search for an invariant is “automatically” guided by the requirements of the LRF. Let us briefly explain this approach.

A template invariant $\mathcal{I}(\boldsymbol{x})$ is a conjunction of linear inequalities over variables $\boldsymbol{x}$ where the coefficients are unknowns, *e.g.*, $\mathcal{I}(x,y) = \{a_1 x + a_2 y \leq a_0\}$ where a_i represent the unknown coefficients. Our interest is to seek a linear function $\rho(x,y) = \vec{\lambda}\boldsymbol{x} + \lambda_0$ and values for a_i, such that $\mathcal{I}(\boldsymbol{x})$ is an invariant for $\mathcal{Q}$ wrt. the initials states $\mathcal{S}_0$ and ρ is an LRF for $\mathcal{Q}(\boldsymbol{x},\boldsymbol{x}') \wedge \mathcal{I}(\boldsymbol{x})$ which can be stated as follows:

$$\mathcal{S}_0(\boldsymbol{x}) \implies \mathcal{I}(\boldsymbol{x})\,, \tag{4.9}$$
$$\mathcal{Q}(\boldsymbol{x},\boldsymbol{x}') \wedge \mathcal{I}(\boldsymbol{x}) \implies \mathcal{I}(\boldsymbol{x}')\,, \tag{4.10}$$
$$\mathcal{Q}(\boldsymbol{x},\boldsymbol{x}') \wedge \mathcal{I}(\boldsymbol{x}) \implies \rho(\boldsymbol{x}) \geq 0\,, \tag{4.11}$$
$$\mathcal{Q}(\boldsymbol{x},\boldsymbol{x}') \wedge \mathcal{I}(\boldsymbol{x}) \implies \Delta\rho(\boldsymbol{x}'') \geq 1\,. \tag{4.12}$$

The first two formulas ensure that $\mathcal{I}(\boldsymbol{x})$ is an inductive invariant for $\mathcal{Q}$, while the remaining formulas ensure that ρ is an LRF for $\mathcal{Q}(\boldsymbol{x},\boldsymbol{x}') \wedge \mathcal{I}(\boldsymbol{x})$, and therefore an $\text{LRF}_{\mathcal{S}_0}$ for $\mathcal{Q}$. This entire problem can be solved using Farkas' Lemma, which transforms it into solving a corresponding system of constraints over the reals in which, among others, a_i and λ_i are variables. However, since the template $\mathcal{I}(\boldsymbol{x})$ appears on the left-hand side of the implications, the resulting constraints are non-linear, and thus solving them is not guaranteed to be polynomial-time (it might be exponential, since the corresponding decision problem is `PSPACE` (Canny, 1988)). Note that such an algorithm is complete for a slightly different problem: Is there a polyhedral invariant $\mathcal{I}(\boldsymbol{x})$ for $\mathcal{Q}$ and $\mathcal{S}_0$, *matching a given template*, such that the rational loop $\mathcal{Q}(\boldsymbol{x},\boldsymbol{x}') \wedge \mathcal{I}(\boldsymbol{x})$ has an LRF?

Example 4.3. Let us apply the template based approach to the *SLC* loop $\mathcal{Q} = \{x \geq 0, x' \leq x - y, y' \geq y + 1\}$ and initial condition $\mathcal{S}_0 = \{y = 1\}$ of Example 4.2, and a template invariant $\mathcal{I}(x,y) = \{a_1 x + a_2 y \leq a_0\}$. We first note that:

$$\mathcal{S}_0(x,y) \equiv \begin{pmatrix} 0 & -1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \leq \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

$$\mathcal{Q}(x,y,x',y') \wedge \mathcal{I}(x,y) \equiv \begin{pmatrix} -1 & 0 & 0 & 0 \\ -1 & 1 & 1 & 0 \\ 0 & 1 & 0 & -1 \\ a_1 & a_2 & 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ x' \\ y' \end{pmatrix} \leq \begin{pmatrix} 0 \\ 0 \\ -1 \\ a_0 \end{pmatrix}$$

Let $\rho(x,y) = \lambda_1 x + \lambda_2 y + \lambda_0$ be an LRF template, *i.e.*, λ_i are unknowns. To synthesise an LRF and an invariant simultaneously, we translate (4.9)-(4.12) into a set of existential constraints using Farkas' lemma which results in ($\vec{\mu}, \vec{\eta}, \vec{\xi}, \vec{\alpha}$ are the Farkas' coefficients):

(4.9)	$0 = a_1,\ -\mu_0 + \mu_1 = a_2,\ -\mu_0 + \mu_1 \leq a_0,\ \mu_0 \geq 0,\ \mu_1 \geq 0$
(4.10)	$-\eta_0 - \eta_1 + \eta_3 a_1 = 0,\ \eta_1 + \eta_2 + \eta_3 a_2 = 0,\ \eta_1 = a_1,$ $-\eta_2 = a_2,\ -\eta_2 + \eta_3 a_0 \leq a_0,\ \eta_0 \geq 0,\ \eta_1 \geq 0,\ \eta_2 \geq 0,\ \eta_3 \geq 0$
(4.11)	$-\xi_0 - \xi_1 + \xi_3 a_1 = -\lambda_1,\ \xi_1 + \xi_2 + \xi_3 a_2 = -\lambda_2,\ \xi_1 = 0,$ $-\xi_2 = 0,\ -\xi_2 + \xi_3 a_0 \leq -\lambda_0,\ \xi_0 \geq 0,\ \xi_1 \geq 0,\ \xi_2 \geq 0,\ \xi_3 \geq 0$
(4.12)	$-\alpha_0 - \alpha_1 + \alpha_3 a_1 = -\lambda_1,\ \alpha_1 + \alpha_2 + \alpha_3 a_2 = -\lambda_2,\ \alpha_1 = \lambda_1,$ $-\alpha_2 = \lambda_2,\ -\alpha_2 + \alpha_3 a_0 \leq -1,\ \alpha_0 \geq 0,\ \alpha_1 \geq 0,\ \alpha_2 \geq 0,\ \alpha_3 \geq 0$

Note that they include nonlinear terms. Solving these constraints we find the following possible solution:

$$\begin{array}{l}
\lambda_0 \mapsto 0,\ \lambda_1 \mapsto 1,\ \lambda_2 \mapsto 0,\\
a_0 \mapsto -1,\ a_1 \mapsto 0\ a_2 \mapsto -1,\\
\mu_0 \mapsto 1,\ \mu_1 \mapsto 0,\\
\eta_0 \mapsto 0,\ \eta_1 \mapsto 0,\ \eta_2 \mapsto 1,\ \eta_3 \mapsto 1,\\
\xi_0 \mapsto 1,\ \xi_1 \mapsto 0,\ \xi_2 \mapsto 0,\ \xi_3 \mapsto 0,\\
\alpha_0 \mapsto 0,\ \alpha_1 \mapsto 1,\ \alpha_2 \mapsto 0,\ \alpha_3 \mapsto 1,
\end{array}$$

Thus, $\rho(x,y) = x$ is an LRF and $y \geq 1$ is a supporting invariant.

OPEN PROBLEM 3. Is it decidable whether a given rational *SLC* loop $\mathcal{Q}$ has an LRF wrt. to a polyhedral set of initial states $\mathcal{S}_0$ and, if yes, what is the complexity of this problem?

Ben-Amram (2014) provides a lower bound on the hardness of this problem.

Theorem 4.2. Deciding if a given rational *SLC* $\mathcal{Q}$ has an LRF wrt. a polyhedral set of initial states $\mathcal{S}_0$ is PSPACE-hard (even if we know that the loop is terminating).

OPEN PROBLEM 4. Are polyhedral invariants sufficient for deciding if an LRF exists for a given *SLC* loop $\mathcal{Q}$ wrt. a polyhedral set of initial states $\mathcal{S}_0$? That is, does $\mathcal{Q}_{\mathcal{S}_0}$ have an LRF if and only if there exists a polyhedral invariant $\mathcal{I}(\boldsymbol{x})$ such that $\mathcal{Q}(\boldsymbol{x}, \boldsymbol{x}') \wedge \mathcal{I}(\boldsymbol{x})$ has an LRF? If

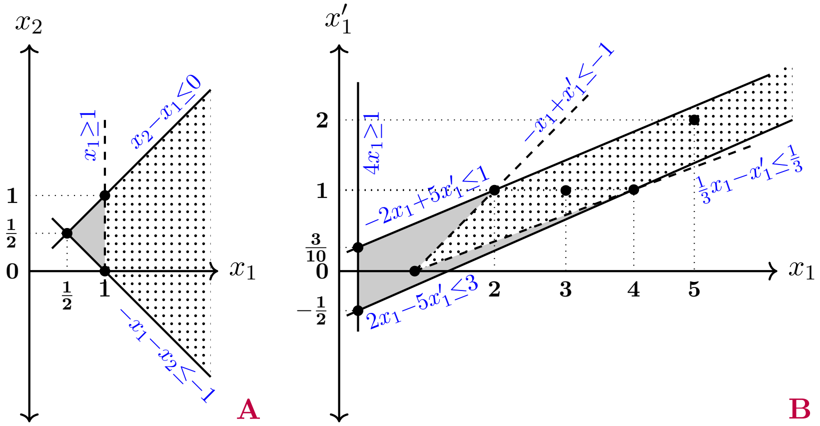


Figure 4.1: The polyhedra associated with two of our examples, projected to two dimensions: **(A)** corresponds to Loop (4.13) on Page 53; **(B)** corresponds to Loop (2.6) on Page 20. Dashed lines are added when computing the integer hull; dotted areas represent the integer hull; Gray areas are rational points eliminated when computing the integer hull (Figure from (Ben-Amram and Genaim, 2014)).

the answer is no, a different question arises: Is it decidable whether a polyhedral supporting invariant $\mathcal{I}(\boldsymbol{x})$ exists such that $\mathcal{Q}(\boldsymbol{x}, \boldsymbol{x}') \wedge \mathcal{I}(\boldsymbol{x})$ has an LRF?

4.1.2 LRFs Over the Integers for *SLC* Loops

When variables range over integers, the *SLC* loop can still be understood in terms of the transition polyhedron $\mathcal{Q} \subseteq \mathbb{Q}^{2n}$, but this time we are interested not in all the rational points in this polyhedron but just in its integer points, *i.e.*, in the set of transitions $I(\mathcal{Q})$. This means that for ρ to be an LRF we require (4.1,4.2) to hold only for $\boldsymbol{x}'' \in I(\mathcal{Q})$.

Example 4.4. Consider the following loop:

$$\textit{while } (x_2 - x_1 \leq 0, x_1 + x_2 \geq 1) \textit{ do } x'_2 = x_2 - 2x_1 + 1, x'_1 = x_1 \qquad (4.13)$$

When considered as an integer loop, it has the LRF $\rho(x_1, x_2) = x_1 + x_2$. On the contrary, over rationals the loop does not always terminate — consider its computation from $(\frac{1}{2}, \frac{1}{2})$.

In the above example, the restriction to integers excludes the non-terminating state $(\frac{1}{2}, \frac{1}{2})$. So a natural step towards analysing a loop over the integers is to reduce the polyhedron to its *integer hull*, since it eliminates all points that are not convex combinations of points from $I(\mathcal{Q})$. Indeed, the integer hull of Loop (4.13) is the following loop, which adds the constraints $x_1 \geq 1$ to the guard (see Figure 4.1(a))

$$\begin{aligned} &\textit{while } (x_2 - x_1 \leq 0, x_1 + x_2 \geq 1, x_1 \geq 1) \textit{ do} \\ &\qquad x_2' = x_2 - 2x_1 + 1, x_1' = x_1 \end{aligned} \tag{4.14}$$

and this loop has the LRF mentioned above, since $(\frac{1}{2}, \frac{1}{2})$ is excluded by the guard. Similarly, Loop (2.6) does not terminate over the rationals, *e.g.*, for initial point $(\frac{1}{4}, 1)$, but terminates, and has an LRF, over integers (see Figure 4.1(b)).

Synthesising LRFs over the integers, can be also reduced to seeking implied inequalities of the form (4.1,4.2), but using $I(\mathcal{Q})$ instead of $\mathcal{Q}$. This can be also be done using Farkas' lemma and $\mathcal{Q}_I$, because an inequality is entailed by $I(\mathcal{Q})$ if and only if it is entailed by $\mathcal{Q}_I$. This was observed independently by several researchers (Feautrier, 1992a; Cook *et al.*, 2013; Ben-Amram and Genaim, 2014).

Theorem 4.3. An integer *SLC* loop $I(\mathcal{Q})$ has an LRF if and only if its integer hull $\mathcal{Q}_I$ has an LRF (as a rational loop).

This gives us a complete algorithm to solve the LRF problem for integer *SLC* loops: compute the integer hull of $\mathcal{Q}$ and use a polynomial-time LRF algorithm. The complexity of computing integer hulls is, in general, exponential. Ben-Amram and Genaim (2014) list a number of special cases which can be solved in polynomial time, since the integer hull can be computed in polynomial time for these cases, but also prove that in general, the LRF problem over integers is `coNP`-complete.

The exponential complexity of computing the integer hull, in the general case, gives the correct intuition as to why the problem is hard. For *inclusion* in `coNP`, Ben-Amram and Genaim (2014) show that $I(\mathcal{Q})$ does *not* have an LRF if and only if there are finite sets $X \neq \emptyset \subseteq I(\mathcal{Q})$ and $Y \subseteq I(\texttt{rec.cone}(\mathcal{Q}))$, of polynomial size, such that the loop $\texttt{conv}\{X\} + \texttt{cone}\{Y\} \subseteq \mathcal{Q}_I$ does not have an LRF, and that this last check can be done in polynomial time.

Let us now consider the case in which the initial states are restricted to a polyhedral set $\mathcal{S}_0 \subset \mathbb{Q}^n$, and recall that our interest is in the integer states $I(\mathcal{S}_0)$. The algorithmic aspects of this case are similar to the one of the rational case (but using $\mathcal{Q}_I$ instead of $\mathcal{Q}$), *i.e.*, either we infer a supporting invariant beforehand and add it to the transition polyhedron, or we use the template approach to synthesise a supporting invariant and an LRF simultaneously. However, there is one important difference regarding the problem of inferring a supporting invariant (that matches a template) and an LRF at the same time: In the rational case the algorithm is complete, but this does not hold for the integer case since $\mathcal{Q}_I(\boldsymbol{x}, \boldsymbol{x}') \wedge \mathcal{I}(\boldsymbol{x})$ is not necessarily an integer polyhedron, and we cannot compute its integer hull because $\mathcal{I}(\boldsymbol{x})$ includes template parameters.

Problems 3 and 4 are also still open for the integer case. Ben-Amram (2014) provided lower bounds on the hardness for related problems.

Theorem 4.4. Deciding whether a given integer *SLC* loop $\mathcal{Q}$ has an LRF wrt. a polyhedral set of initial states $\mathcal{S}_0$ is `Ackermann`-hard[2].

Theorem 4.5. Deciding whether a given integer *SLC* loop $\mathcal{Q}$ has a polyhedral inductive invariant $\mathcal{I}(\boldsymbol{x})$ wrt. a polyhedral set of initial states $\mathcal{S}_0$ (not necessarily matching a template) such that $\mathcal{Q}(\boldsymbol{x}, \boldsymbol{x}') \wedge \mathcal{I}(\boldsymbol{x})$ has an LRF over the integers is `PSPACE`-hard.

4.1.3 LRFs for *MLC* Loops

An LRF for an *MLC* loop $\mathcal{Q}_1, \dots, \mathcal{Q}_k$, is a function ρ which is an LRF for all its transitions $T = \mathcal{Q}_1 \cup \cdots \cup \mathcal{Q}_k$, that is all the paths. The following complexity results follow quite easily.

Polynomial-time Synthesis for Rational Loops

We create for each path $\mathcal{Q}_i$ a constraint system as in (4.3,4.4), where each system uses different $\vec{\mu}$ and $\vec{\eta}$, say $\vec{\mu}_i$ and $\vec{\eta}_i$, but the same $(\vec{\lambda}, \lambda_0)$. This results in a bigger, still polynomial-sized LP problem, and its solutions define LRFs that hold for all paths. We can also do the

[2] This follows from a reduction in Ben-Amram, 2014 along with recent results on the hardness of reachability in Vector Addition Systems (Czerwinski and Orlikowski, 2021; Leroux, 2021).

same using (4.8) instead of (4.3,4.4), but in this case we have to add constraints requiring the LRF coefficients arising from each of these sub-problems to coincide, namely $\vec{\lambda} = -\vec{\mu}_i A$ and $\lambda_0 \geq \vec{\eta}_i \boldsymbol{c}$ for each $\mathcal{Q}_i$.

Example 4.5. Consider the *MLC* loop of Example 2.6, and note that x_1 is an LRF for $\mathcal{Q}_1$ and x_2 is an LRF for $\mathcal{Q}_2$. However, the *MLC* loop defined by both paths does not have an LRF. Modifying the paths to

$$\mathcal{Q}_1 = \{x_1 \geq 0, x_2 \geq 0, x_1' = x_1 - 1, x_2' = x_2\}$$
$$\mathcal{Q}_2 = \{x_1 \geq 0, x_2 \geq 0, x_1' \leq x_1, x_2' = x_2 - 1\}$$

the loop has an LRF $\rho(x_1, x_2) = x_1 + x_2$.

LRFs Over the Integers for *MLC* Loops

For integer loops we get a complete algorithm by first computing the integer hulls of all paths, namely $(\mathcal{Q}_1)_I, \ldots, (\mathcal{Q}_k)_I$, and then applying the algorithm of the rational case. The completeness of this method follows from the same considerations as the ones of *SLC* loops. Ben-Amram and Genaim (2014) show that deciding if a given integer *MLC* loop has an LRF is `coNP`-complete. The hardness is clear since it is already hard for *SLC* loops. Inclusion in `coNP` is shown by generalising the witnesses of the *SLC* case to cover all paths.

Example 4.6. Let use consider an *MLC* $\mathcal{Q}_1, \mathcal{Q}_2, \mathcal{Q}_3$, where the first two paths are those of Example 4.5, and the last is that of the *SLC* loop (4.13). This loop does not have an LRF over the rationals since $\mathcal{Q}_3$ does not, however, over the integers it has the LRF $\rho(x_1, x_2) = x_1 + x_2$. To synthesise this LRF we have to compute the integer hull of all paths first (note that $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are already integral, and $(\mathcal{Q}_3)_I$ is Loop (4.14)).

LRFs for *MLC* Loops with Polyhedral Set of Initial States

The same consideration for the case of *SLC* loop applies to *MLC* loops as well, both for the rational and the integer case. In particular we can use the template based approach which in this case requires (4.10)-(4.12) for all paths. As for the complexity of related problems (*e.g.*, problems 3 and 4), nothing is known for the *MLC* case.

4.1.4 LRFs for CFGs

In this section we discuss how the algorithmic and complexity aspects of the LRF problem extend to the case of CFGs. In what follows, we assume a given CFG $P = (V, R, L, \ell_0, E)$ where R is $\mathbb{Q}$ or $\mathbb{Z}$ (recall that the case of $\mathbb{R}$ is the same as that of $\mathbb{Q}$). We first consider the case where the execution can start at any location, and then restrict to locations ℓ_0.

To generalise Definition 4.2 of an LRF to CFGs, all we need is to require (4.1,4.2) to hold for any $\boldsymbol{x}'' = (\boldsymbol{x}, \boldsymbol{x}') \in \mathcal{Q}_{\ell,\ell'} \in E$, *i.e.*, for all transitions on all edges. In such case, the LRF ρ guarantees universal termination, meaning that an execution can start from any location, not just ℓ_0, and with any values $\boldsymbol{x} \in R$ for the program variables. With this adjustment, all complexity and algorithmic aspects, of the LRF problem, previously discussed for universal termination of *MLC* loops also apply to CFGs, both for rational and integer variables.

However, due to their complex structure, CFGs are unlikely to admit an LRF of this form. For instance, a CFG might include several (simple) loops, each potentially having a distinct LRF, and even if they shared the same LRF, the edges connecting these loops are not likely to satisfy Condition (4.2). Moreover, a loop might be represented by several edges in the CFG where only in one of them the loop counter decreases, while in the rest it stays the same (*i.e.*, it is impossible to have a single function that decreases on all these edges).

It is therefore desirable to use a more general definition, where we allow each node to use a different function ρ_ℓ, and change (4.1,4.2) to require that each $\boldsymbol{x}'' \in \mathcal{Q}_{\ell,\ell'} \in E$ satisfy:

$$\rho_\ell(\boldsymbol{x}) \geq 0 \tag{4.15}$$

$$\rho_\ell(\boldsymbol{x}) - \rho_{\ell'}(\boldsymbol{x}') \geq 1\,. \tag{4.16}$$

Now an LRF is a collection of linear functions, where each node is assigned one. The algorithmic and complexity aspects of synthesising such an LRF are the same as in the case of LRF for *MLC* loops.

Example 4.7. Consider the CFG in Figure 4.2, and assume that invariants have been added to the corresponding transitions (this is what

```
assert(x>=0);
int y = 1;
while(x >= 0) {
  if (nondet()) {
    y=2*y;
    if (nondet()) break;
  } else y++;
  x--;
}
x = y;
while (y>=0) {
  y--;
  x = 3*x;
}
```

$\mathcal{Q}_0$: $\{x \geq 0, x' = x, y' = 1\}$
$\mathcal{Q}_1$: $\{x \geq 0, x' = x, y = y\}$
$\mathcal{Q}_2$: $\{x \leq -1, x' = x, y' = y\}$
$\mathcal{Q}_3$: $\{x' = x, y' = 2 * y\}$
$\mathcal{Q}_4$: $\{x' = x, y' = y + 1\}$
$\mathcal{Q}_5$: $\{x' = x, y = y\}$
$\mathcal{Q}_6$: $\{x' = x - 1, y = y\}$
$\mathcal{Q}_7$: $\{x' = x, y = y\}$
$\mathcal{Q}_8$: $\{x' = y, y' = y\}$
$\mathcal{Q}_9$: $\{y \geq 0, y' = y - 1, x' = 3 * x\}$
$\mathcal{Q}_{10}$: $\{y \leq -1, x' = x, y = y\}$

$\mathcal{S}_0 = \mathcal{I}_{\ell_0} = \{\}$
$\mathcal{I}_{\ell_1} = \mathcal{I}_{\ell_5} = \{x \geq -1, y \geq 1\}$
$\mathcal{I}_{\ell_2} = \mathcal{I}_{\ell_3} = \mathcal{I}_{\ell_4} = \{x \geq 0, y \geq 1\}$
$\mathcal{I}_{\ell_6} = \mathcal{I}_{\ell_7} = \{x \geq 1, y \geq -1\}$

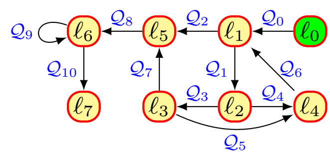


Figure 4.2: A program (taken from (Alias *et al.*, 2010)), its corresponding CFG, and invariants when starting location l_0).

we usually do when starting from ℓ_0, but we apply it here to keep the example simple and meaningful). Let us also ignore the second loop for now (and thus nodes ℓ_6 and ℓ_7); we will consider it later. If we seek an LRF that assigns the same function ρ to all nodes, we will not find one, because in many transitions we have $x' = x$. Instead, we look for an LRF that assigns a (possibly) different function ρ_ℓ to each node, and we find the following:

$$\begin{array}{lll} \rho_{\ell_0}(x,y) = 3x+5 & \rho_{\ell_2}(x,y) = 3x+3 & \rho_{\ell_4}(x,y) = 3x+2 \\ \rho_{\ell_1}(x,y) = 3x+4 & \rho_{\ell_3}(x,y) = 3x+2 & \rho_{\ell_5}(x,y) = 3x+1 \end{array}$$

These functions are only different in the constant, which means that we could use templates for the different ρ_ℓ that are different only in the constants. This would be more efficient in practice since the corresponding LP problems will have fewer variables. Note that from this LRF (*i.e.*, the collection of all ρ_i) we can construct a ranking function as in Definition 4.1, namely: $\rho(\ell, (x, y)) = \max(0, \lceil \rho_\ell(x, y) + 1 \rceil)$.

In the example above, we have limited ourselves to one loop, because if we seek an LRF for the whole CFG, even when using different functions for the different nodes, we would fail: while the LRF of the first loop is based on the loop counter x, the second is based on the loop counter y. Instead, we could analyse the strongly connected components (SCCs) separately—note that for a termination proof this suffices: if there were an infinite execution, it would eventually stay within a single SCC. In this case it is not always possible to construct a global "linear" ranking function (there may be a global ranking function of a more complex form).

Example 4.8. Let us analyse the SCCs of the CFG of Figure 4.2 separately. We start by seeking an LRF for the SCC of $\mathcal{Q}_1$, $\mathcal{Q}_3$, $\mathcal{Q}_4$, $\mathcal{Q}_5$ and $\mathcal{Q}_6$. We find the same functions as in the previous example for the corresponding nodes. Next we continue with the SCC of $\mathcal{Q}_9$, and we find $\rho_{\ell_6}(x, y, z) = y$.

Let us now consider the case in which we seek an LRF wrt. a polyhedral set of initial states $\mathcal{S}_0 \subseteq \mathbb{Q}^n$, and starting at ℓ_0. Similarly to the case of *MLC* loops, we can solve the problem by first inferring supporting polyhedral invariants (for each location), add them to the transition relations of corresponding outgoing edges, and then use the algorithm of universal termination as described above—this is what we have done in the examples above actually. We can also simultaneously infer invariants and seek the functions ρ_ℓ using the template approach, which is very similar to the case of *SLC* and *MLC* loops, except that here we have an invariant for each location. Also in this case we obtain a complete algorithm, for the rational case, to the problem of deciding whether the template can be instantiated such that the CFG (or a given SCC) has an LRF. Finally, as for the complexity of related problems (*e.g.*, problems 3 and 4), nothing is known for the CFG case.

4.1.5 History of LP-based LRFs Algorithms

Algorithms to find an LRF for *SLC* loops have been proposed by several researchers (Sohn and Gelder, 1991; Colón and Sipma, 2001; Feautrier, 1992a; Podelski and Rybalchenko, 2004a; Mesnard and Serebrenik, 2008).

All these works, even if originating from an application where variables are integer, relax the problem to the rationals. Bagnara *et al.* (2012) overview and compare the methods of Sohn and Gelder (1991), Podelski and Rybalchenko (2004a), and Mesnard and Serebrenik (2008).

It may be interesting to note that while most of these works concern termination, Feautrier (1992a) employs ranking functions for a different purpose, solving a *scheduling problem* for parallel computation. It is also the only one among these works that discusses the integer case and its complexity, and in doing so it precedes the works of Ben-Amram and Genaim (2014) and Cook *et al.* (2013). Bradley *et al.* (2005b) also studied LRFs for integer linear-constraint loops

4.1.6 Other Approaches for LRFs

In contrast to work that are based on the use of Farkas' lemma, Li *et al.* (2020) show that, in the rational case, one can compute a witness against the existence of an LRF in polynomial time. A generalisation of this approach has been reported by Ben-Amram *et al.* (2019) for multiphase ranking functions (see Section 4.2.4), and used to show the following result for bounded SLC loops.

Theorem 4.6. Let $\mathcal{Q}$ be an SLC loop such that the set of enabled states $\mathtt{proj}_{\boldsymbol{x}}(\mathcal{Q})$ is a bounded polyhedron, then: either $\mathcal{Q}$ is non-terminating and has a fixpoint $\binom{\boldsymbol{x}}{\boldsymbol{x}} \in \mathcal{Q}$, or it is terminating and has an LRF.

Maurica *et al.* (2016) consider the problem of synthesising LRFs for floating-point SLC loops. They show that the decision problem is at least coNP-hard and provide an incomplete algorithm for synthesising LRFs for such loops.

4.2 Lexicographic-Linear Ranking Functions

The notion of lexicographic ranking functions is ubiquitous in termination analysis because they naturally arise when analysing nested loops or programs with complex control flow, as in the following example.

Example 4.9. Consider an *MLC* loop defined by the following paths

$$\begin{aligned} \mathcal{Q}_1 &= \{x_1 \geq 0, x_2 \geq 0, x_1' = x_1 - 1\} \\ \mathcal{Q}_2 &= \{x_1 \geq 0, x_2 \geq 0, x_2' = x_2 - 1, x_1' = x_1\} \end{aligned} \tag{4.17}$$

In $\mathcal{Q}_1$, x_1 decreases towards zero and x_2 is changed unpredictably, since there is no constraint on x_2'; this could arise, for instance, from x_2 being set to the result of an input from the environment, an expression that cannot be modelled using linear constraints, or a function call for which we have no input-output summary. In $\mathcal{Q}_2$, x_2 decreases towards zero and x_1 is unchanged. Clearly, $\langle x_1, x_2 \rangle$ always decreases lexicographically, while there can be no single LRF for this loop. Similarly, the same tuple decreases lexicographically for the *MLC* loop of Example 2.6, that does not have an LRF as well.

Interestingly, Alan Turing's early demonstration (Turing, 1948) of how to verify a program used a lexicographic ranking function for the termination proof. For the sake of developing practical tools, and for studying properties of lexicographic ranking functions, one typically restricts the form of functions allowed as components. A common such restriction considers components that are linear affine functions, yielding *lexicographic-linear ranking functions* (LLRFs). In the rest of this section, we use ρ_i to denote a linear affine function that maps states to rational values, as in the case of LRFs. The most general definition for an LLRF is the following.

Definition 4.3. Given a transition relation $T \subseteq R^{2n}$, where $R \in \{\mathbb{R}, \mathbb{Q}, \mathbb{Z}\}$, we say that $\tau = \langle \rho_1, \dots, \rho_d \rangle$ is an LLRF (of depth d) for T, if for every $\boldsymbol{x}'' \in T$ there is an index i such that:

$$\forall j < i \,.\, \Delta\rho_j(\boldsymbol{x}'') \geq 0\,, \tag{4.18}$$

$$\Delta\rho_i(\boldsymbol{x}'') \geq 1\,, \tag{4.19}$$

$$\rho_i(\boldsymbol{x}) \geq 0\,, \tag{4.20}$$

We say that $\boldsymbol{x}''$ is *ranked by* ρ_i (for the minimal such i).

The justification that an LLRF implies termination uses the fact that the lexicographic order over $\mathbb{N}^d$ is well-founded. Given an LLRF $\langle \rho_1, \dots, \rho_d \rangle$, we coerce the component ρ_i to $\max(0, \lceil \rho_i + 1 \rceil)$ and get a tuple $\langle \max(0, \lceil \rho_1 + 1 \rceil), \dots, \max(0, \lceil \rho_d + 1 \rceil) \rangle$ that decreases lexicographically over $\mathbb{N}^d$. This works since each ρ_i decreases by at least 1 on the transitions that it ranks.

Remark 4.4. Replacing (4.19) by $\Delta\rho_i(\boldsymbol{x}'') > 0$, we obtain a definition for a *weak* LLRFs. While weak LLRFs do not clearly imply termination (over the rationals or reals), they are useful to infer LLRFs as we will see later. Over the integers, weak LLRFs are equivalent to LLRFs since we may assume that all coefficients of ρ_i are integer, and thus $\Delta\rho_i(\boldsymbol{x}'') > 0$ means $\Delta\rho_i(\boldsymbol{x}'') \geq 1$.

It is easy to see that a given tuple $\langle \rho_1, \dots, \rho_d \rangle$ is an LLRF for T if and only if the following formula holds:

$$\left(\bigwedge_{i=1}^{d} (T_i(\boldsymbol{x}, \boldsymbol{x}') \implies \Delta\rho_i(\boldsymbol{x}'') \geq 0) \right) \wedge (T_{d+1}(\boldsymbol{x}, \boldsymbol{x}') \implies \mathit{false}) \quad (4.21)$$

where $T_i(\boldsymbol{x}, \boldsymbol{x}') = T(\boldsymbol{x}, \boldsymbol{x}') \wedge (\wedge_{j=1}^{i-1} (\rho_j(\boldsymbol{x}) < 0 \vee \Delta\rho_j(\boldsymbol{x}'') < 1))$, *i.e.*, we remove all transitions that are ranked by any component ρ_j with $j < i$.

This formulation gives rise to the *template based approach* for synthesising an LLRF of a given depth (Leike and Heizmann, 2015). We start from template functions $\rho_i(\boldsymbol{x}) = \vec{\lambda}_i \boldsymbol{x} + \lambda_{0,i}$, where $\vec{\lambda}_i$ and $\lambda_{0,i}$ are variables ("template parameters"), and then using the Motzkin transposition theorem, which is similar to Farkas' Lemma, we translate (4.21) into a set of existential constraints over the template parameters (and some other variables) that can be solved using off-the-shelf SMT solvers, and thus get concrete values for the coefficients of each ρ_i.

The resulting existential constraints, however, are non-linear since the constraints that we add in each T_i use template parameters. They can be solved within polynomial space complexity since the corresponding decision problem, over the reals, is PSPACE (Canny, 1988). Note we only propose this approach for loops over the reals, and assuming that T is given by polyhedra. To decide existence of an LLRF, we can search iteratively for increasing values of depth d, however if there is no LLRF this method does not terminate. Note also that we could incorporate inference of supporting invariants, similarly to what we have done for LRFs.

Algorithm 1: Synthesizing Lexicographical Linear Ranking Functions

LLRFSYN(T)
Input: A set of transition $T \subseteq R^{2n}$, where $R \in \{\mathbb{R}, \mathbb{Q}, \mathbb{Z}\}$
Output: An LLRF τ for T, if exists, otherwise NONE
begin
1 $\quad \tau := \langle\rangle$
2 $\quad T' := T$
3 $\quad$ **while** T' *is not empty* **do**
4 $\qquad$ **if** T' *has a non-trivial quasi*-LRF ρ *wrt.* T **then**
5 $\qquad\quad T' = T' \setminus \{\boldsymbol{x}'' \in T' \mid \boldsymbol{x}'' \text{ is (weakly) ranked by } \rho\}$
6 $\qquad\quad \tau = \tau :: \rho$
$\qquad$ **else**
8 $\qquad\quad \tau =$ NONE
9 $\qquad\quad$ **break**
10 $\quad$ **return** τ

An alternative and widely used approach for synthesising LLRFs is based on a greedy algorithm (Algorithm 1), which incrementally builds the LLRF by seeking a *quasi*-LRFs. We first give the definition of a *quasi*-LRF, and then explain the method, shown as Algorithm 1.

Definition 4.4. We say that an affine linear function ρ is *quasi*-LRF (QLRF for short) for $T' \subseteq T \subseteq R^{2n}$ if the following holds for all $\boldsymbol{x}'' \in T'$:

$$\Delta\rho(\boldsymbol{x}'') \geq 0 \tag{4.22}$$

We say that it is *non-trivial* if, in addition, $\Delta\rho(\boldsymbol{x}'') > 0$ and $\rho(\boldsymbol{x}) \geq 0$ for at least one $\boldsymbol{x}'' \in T'$. We say that $\boldsymbol{x}''$ is (weakly) ranked by ρ.

This definition of QLRFs will be specialised later by adding more conditions; these variants correspond to variants of LLRFs, that are special cases of Definition 4.3. In some of these specialised definitions, the set T (which is redundant in the above definition) will play a role.

Algorithm 1 incrementally builds an LLRF, in each iteration of the while loop, as follows: at Line 4 it seeks a QLRF ρ for the current set of transitions T', and if it fails it exits the loop with $\tau =$ NONE; at Line 5 it eliminates all transitions that are (weakly) ranked by ρ from T', and then appends ρ to τ. When all transitions are eliminated from T', it exits the loop and returns τ at Line 10 which can be an LLRF, possibly weak, or None in case of failure.

The LLRF is possibly weak because depending on the specific definition of the QLRF and the domain of variables, the transitions that are eliminated at Line 5 might be weakly ranked. For example, if $T \subseteq \mathbb{Q}^{2n}$ and we eliminate all those weakly ranked by ρ, *i.e.*, the transitions on which ρ is decreasing ($\Delta\rho(\boldsymbol{x}'') > 0$) and non-negative ($\rho(\boldsymbol{x}) \geq 0$), then we get a weak LLRF which is not enough for proving termination over $\mathbb{Q}$. Some approaches solve this issue by converting the weak LLRF into an LLRF (of the same depth) afterwards, other approaches guarantee that transitions that are eliminated at Line 5 actually satisfy $\Delta\rho(\boldsymbol{x}'') \geq 1$ and thus directly build an LLRF. Recall that over the integers, weak LLRFs are enough since we may assume that all coefficients of ρ are integer, and thus $\Delta\rho(\boldsymbol{x}'') > 0$ means $\Delta\rho(\boldsymbol{x}'') \geq 1$. Termination of the algorithm also depends on the choice of the QLRF, and on how transitions are eliminated from T'.

The following is a fundamental property that is used to prove completeness of corresponding algorithms for synthesising LLRFs.

OBSERVATION 4.7. If $T \subseteq R^{2n}$ has an LLRF $\langle \rho_1, \dots, \rho_d \rangle$, then any subset of transitions T' must have a non-trivial QLRF, namely ρ_j for $j = \max\{i \mid \boldsymbol{x}'' \in T' \text{ is ranked by } \rho_i\}$).

A natural question to ask, given a definition of a QLRF, is whether there is an optimal QLRF ρ that eliminates as many transitions as possible (*i.e.*, if $\boldsymbol{x}''$ is eliminated by some QLRF ρ', then it is eliminated by ρ as well). This has the following consequence: if there is an optimal one, and it is picked in each iteration of Algorithm 1, then the returned LLRF is of minimal depth (the number of components of the LLRF). Unfortunately, there does not have to be an optimal choice for QLRFs as in Definition 4.4. In certain variants of QLRFs, as we will see later, there actually is an optimal choice.

The minimal depth is of interest when LLRFs are used to infer bounds on the number of execution steps, for example this is the case in Alias *et al.* (2010) where such bound is typically a polynomial of degree d, where d is the depth of the LLRF. It is also natural to ask whether there is an *a priori* upper bound on the depth, in terms of parameters of the loop (such as the number of variables). Such an upper bound is useful, for example, for fixing the template in the template-based approach, and plays a role in analysing the complexity of corresponding algorithms.

The research problems we are interested in this context, for integer and rational *MLC* loops (and CFGs), are:

Q1 Is there a complete algorithm for synthesising LLRFs? If so, what is its complexity.

Q2 How difficult is it to decide if an LLRF exists for a given *MLC* loop?

Q3 Is there an *a priori* bound on the depth, in terms of the number of variables and paths of a given *MLC* loop?

Q4 Is there a complete algorithm for synthesising LLRFs of a given depth? If so, what is its complexity.

Q5 How difficult is it to find an LLRF of minimal depth, or as a relaxation of this optimisation problem, how difficult to decide if there exists an LLRF that satisfies a given bound on the depth?

All these problems are still open for LLRFs as in Definition 4.3. The only approach we are aware of for synthesising such LLRFs, for *integer MLC* loops, is that of Larraz *et al.* (2013). Their algorithm uses max-SMT to synthesise QLRFs as follows: they use Farkas' lemma to generate a set of constraints whose solutions define all functions that satisfy (4.22) for all paths, but in addition they add *soft constraints* that require some paths to be ranked – the idea is that the max-SMT solver will try to maximise the number of soft constraints that are satisfied. Moreover, in addition to the QLRF, they infer a supporting invariant which makes the generated constraints non-linear as we have seen in the case of LRFs. Importantly, their algorithm is not complete, and they do not consider any question related to complexity of the underlying decision problems.

Different researchers had come up with different variants of the notion of LLRF for which there are answers to these questions. These

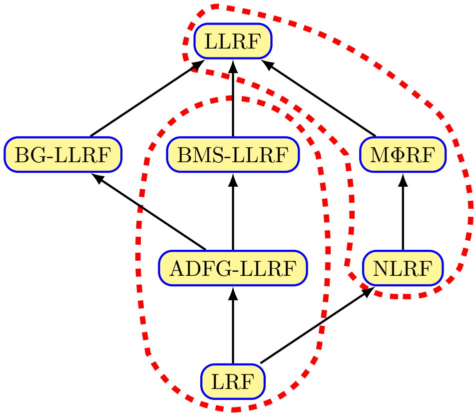


Figure 4.3: Classes of ranking functions for *MLC* loop, ordered by their relative power. The classes surrounded by dashed lines become equivalent when restricted to *SLC* loops.

variants, and their relative power, are summarised in Figure 4.3, and Table 4.2 includes a summary of answers to the corresponding questions.

We note that a loop might have an LLRF according to one of these variants but not another, for example the following *SLC* loop

$$\begin{array}{l} \textit{while } (x \geq 0,\ y \leq 10,\ z \geq 0,\ z \leq 1)\ \textit{do} \\ \qquad x' = x + y + z - 10,\ y' = y + z,\ z' = 1 - z \end{array} \tag{4.23}$$

has the LLRF $\langle 4y, 4x - 4z + 1 \rangle$ according to Definition 4.3, but it is not admitted by any of the variants that we will discuss. In addition, it is possible for a loop to have LLRFs of all variants, but such that the minimal depth is not the same in all of them (see Example 4.20 in Section 4.2.3). Interestingly, all these variants can be described using Algorithm 1, where the main differences between them are: (1) the additional conditions they impose on QLRFs; and (2) the way (weakly) ranked transitions are eliminated. We discuss the details in the next sections. For each variant, we first discuss the case of *MLC* (and *SLC*) loops without initial states, then with initial states, and finally the case of CFGs. As in the case of LRFs, by default we assume that variables range over $\mathbb{Q}$, and the case of $\mathbb{Z}$ will always be discussed separately. The case when variables range over $\mathbb{R}$ is equivalent of that of $\mathbb{Q}$.

		Q1	Q2	Q3	Q4	Q5
Over $\mathbb{Q}$	LLRF	?	?	?	EXPTIME	?
	BG-LLRF	PTIME	PTIME	n	PTIME	PTIME
	ADFG-LLRF	PTIME	PTIME	$\min(n,k)$	PTIME	PTIME
	BMS-LLRF	PTIME	PTIME	k	EXPTIME	NP-complete
	MΦRF	?	?	?	EXPTIME	?
	MΦRF (*SLC*)	?	?	?	PTIME	PTIME
Over $\mathbb{Z}$	LLRF	?	?		?	?
	BG-LLRF	EXPTIME	coNP-complete	n	EXPTIME	coNP-complete
	ADFG-LLRF	EXPTIME	coNP-complete	$\min(n,k)$	EXPTIME	coNP-complete
	BMS-LLRF	EXPTIME	coNP-complete	k	EXPTIME	Σ_2^P
	MΦRF	?	?	?	?	?
	MΦRF (*SLC*)	?	?	?	EXPTIME	coNP-complete

Table 4.2: Summary of results, for the research questions **Q1-5** on Page 65, for the different notions of LLRFs for *MLC* loops (with k paths and n variables). For CFGs, the results are the same as in the case of *MLC*. For *SLC* loops all results are the same as in the case of *MLC*, except for MΦRFs that we report explicitly in separated lines. The case of $\mathbb{R}$ is the same as $\mathbb{Q}$.

4.2.1 BG-LLRFs

The following definition of an LLRF is due to Ben-Amram and Genaim (2014), which is obtained by strengthening (4.20) of Definition 4.3 to require $\rho_j(\boldsymbol{x}) \geq 0$ for all $j \leq i$ – this is reflected in (4.24) of Definition 4.5.

Definition 4.5. Given an *MLC* loop $\mathcal{Q}_1, \ldots, \mathcal{Q}_k \subseteq \mathbb{Q}^{2n}$, we say that $\tau = \langle \rho_1, \ldots, \rho_d \rangle$ is a BG-LLRF (of depth d) for the loop, if for every $\boldsymbol{x}'' \in \mathcal{Q}_1 \cup \cdots \cup \mathcal{Q}_k$ there is an index i such that:

$$\forall j \leq i \;.\quad \rho_j(\boldsymbol{x}) \geq 0\,, \tag{4.24}$$

$$\forall j < i \;.\; \Delta\rho_j(\boldsymbol{x}'') \geq 0\,, \tag{4.25}$$

$$\Delta\rho_i(\boldsymbol{x}'') \geq 1\,. \tag{4.26}$$

We say that $\boldsymbol{x}''$ is *ranked by* ρ_i (for the minimal such i).

Example 4.10. Consider the *SLC* loop

$$\begin{array}{l} \textit{while } (x_1 \geq 0, x_2 \geq 0, x_3 \geq -x_1) \textit{ do} \\ \qquad\qquad x_2' = x_2 - x_1,\; x_3' = x_3 + x_1 - 2\,. \end{array} \tag{4.27}$$

This loop has a BG-LLRF $\tau = \langle x_2, x_3 \rangle$ as in Definition 4.5 (over both rationals and integers). Note that when x_2 decreases, x_3 can be negative, *e.g.*, for $x_1 = 1$, $x_2 = 2$ and $x_3 = -1$. The *MLC* of Example 4.9 has a

BG-LLRF $\tau = \langle x_1, x_2 \rangle$. The *MLC* loop of Example 2.6 does not have a BG-LLRF (recall that it has a LLRF $\langle x_1, x_2 \rangle$).

Replacing $\Delta\rho_i(\boldsymbol{x}'') \geq 1$ by $\Delta\rho_i(\boldsymbol{x}'') > 0$ in (4.26) we obtain a class of functions that Ben-Amram and Genaim (2014) call *weak* BG-LLRFs, which are similar to weak LLRFs that we have discussed previously. For integer loops, it is easy to see that weak and non-weak BG-LLRFs are equivalent for proving termination, since we may assume that all ρ_i have integer coefficients and thus $\Delta\rho_i(\boldsymbol{x}'') > 0$ means $\Delta\rho_i(\boldsymbol{x}'') \geq 1$. Ben-Amram and Genaim (2013) show that this equivalence is also true for rational loops, and provide a polynomial-time algorithm for converting a weak BG-LLRF into a BG-LLRF of the same depth. We rely on this algorithm to convert the weak LLRF returned by Algorithm 1 to an LLRF.

Definition 4.6. Let $\mathcal{Q}_1, \ldots, \mathcal{Q}_k$ be an *MLC* loop. We say that an affine linear function ρ is a BG-QLRF for $\mathcal{Q}'_1 \cup \cdots \cup \mathcal{Q}'_k \subseteq \mathbb{Q}^{2n}$, where $\mathcal{Q}'_i \subseteq \mathcal{Q}_i$, if the following holds for all $\boldsymbol{x}'' \in \mathcal{Q}'_1 \cup \cdots \cup \mathcal{Q}'_k$:

$$\rho(\boldsymbol{x}) \geq 0 \tag{4.28}$$

$$\Delta\rho(\boldsymbol{x}'') \geq 0 \tag{4.29}$$

We say that it is *non-trivial* if, in addition, inequality (4.29) is strict, *i.e.*, $\Delta\rho(\boldsymbol{x}'') > 0$, for at least one $\boldsymbol{x}'' \in \mathcal{Q}'_1 \cup \cdots \cup \mathcal{Q}'_k$.

When compared to QLRFs as in Definition 4.4, the difference is that ρ is required to be non-negative on the set of transitions under consideration and not only on the transitions for which $\Delta\rho(\boldsymbol{x}'') > 0$ holds. This is a stronger requirement, however, it has the following consequence: any non-trivial conic combination of BG-QLRFs ρ_1 and ρ_2 results in a BG-QLRF that ranks all transitions ranked by ρ_1 and ρ_2, which means that there exists an optimal BG-QLRF, given the loop.

Example 4.11. Consider the *SLC* loop (4.27): $\rho(x_1, x_2, x_3) = x_2$ is a non-trivial BG-QLRF; $\rho(x_1, x_2, x_3) = x_1$ is not because $x_1 - x'_1 \geq 0$ does not hold for all transitions; and $\rho(x_1, x_2, x_3) = x_3$ is not because $\rho(2, 1, -1) = -1 < 0$. For the *MLC* loop of Example 4.9: $\rho(x_1, x_2) = x_1$ is a non-trivial BG-QLRF, while $\rho(x_1, x_2) = x_2$ is not because $x_2 - x'_2 \geq$

0 does not hold for all transitions. The *MLC* loop of Example 2.6 does not have a BG-QLRF because x_1 and x_2 can be arbitrarily negative.

Ben-Amram and Genaim (2013) provide a complete polynomial-time algorithm for seeking an optimal non-trivial BG-QLRF $\rho(\boldsymbol{x}) = \vec{\lambda}\boldsymbol{x} + \lambda_0$ for a set of transitions defined by an *MLC* loop $\mathcal{Q}_1, \ldots, \mathcal{Q}_k$. The algorithm is as follows:

(1) Set up an LP problem (using Farkas' Lemma) requiring all $\mathcal{Q}_j$ to imply (4.28,4.29) for all $1 \leq j \leq k$. This generates a set of linear constraints over the variables $(\vec{\lambda}, \lambda_0)$ and some other variables for the Farkas' coefficients; we denote the polyhedron specified by these constraints by $\mathcal{S}$;

(2) Pick a point from the *relative interior* of $\mathcal{S}$, which fixes values for $(\vec{\lambda}, \lambda_0)$ and thus define ρ; and

(3) If $\rho(\boldsymbol{x}) > 0$ holds for some $\boldsymbol{x}'' \in \mathcal{Q}_1 \cup \cdots \cup \mathcal{Q}_k$, then ρ is an optimal BG-QLRF, otherwise there is no non-trivial BG-QLRF.

The key point of this algorithm is that any $(\vec{\lambda}, \lambda_0)$ that comes from the relative interior of $\mathcal{S}$ leads to an optimal BG-QLRF ρ.

When this algorithm is used within Algorithm 1, once ρ has been found at Line 4, we eliminate all (weakly) ranked transitions by adding $\Delta\rho(\boldsymbol{x}'') = 0$ to each $\mathcal{Q}_j$ at Line 5. It easy to see that when the algorithm reaches Line 10, the tuple τ is a weak BG-LLRF, and, moreover, it is of minimal depth since we use optimal BG-QLRFs. As we have mentioned before, τ can be always converted to a BG-LLRF of the same depth, in polynomial time.

Completeness is due to the following two properties: (1) The algorithm is guaranteed to terminate, because $\mathcal{Q}_j \wedge \Delta\rho(\boldsymbol{x}'') = 0$ is a proper face of $\mathcal{Q}_j$, and thus its dimension is smaller than that of $\mathcal{Q}_j$ (the dimension of the empty polyhedron is -1); and (2) When it returns None, then indeed there is no BG-LLRF for the loop. This is because it has found a subset of transitions for which no non-trivial BG-QLRF exists, which would be impossible if the loop had a BG-LLRF (see Observation 4.7).

The complexity of Algorithm 1 in this case is polynomial since every iteration is polynomial. In fact, this is not immediate since reducing $\mathcal{Q}_j$ to $\mathcal{Q}_j \wedge \Delta\rho(\boldsymbol{x}'') = 0$ might potentially increase the bit-size of that path exponentially during the iterations. However, Ben-Amram and Genaim (2014) show that this reduction can be done by changing one of the inequalities of $\mathcal{Q}_j$ to an equality since $\mathcal{Q}_j \wedge \Delta\rho(\boldsymbol{x}'') = 0$ is a face of $\mathcal{Q}_j$, and thus we at most double the size of the constraint representation of $\mathcal{Q}_j$ during *all* iterations. The number of iterations is bounded by the maximum dimension of $\mathcal{Q}_1, \ldots, \mathcal{Q}_k$.

Theorem 4.8 ((Ben-Amram and Genaim, 2014))**.** There is a complete polynomial-time algorithm for finding a BG-LLRF of minimal depth, if one exists, for a given rational *MLC* loop $\mathcal{Q}_1, \ldots, \mathcal{Q}_k$.

Example 4.12. Let us demonstrate the algorithm on the *SLC* loop (4.27) of Example 4.10, which is defined by

$$\mathcal{Q} = \{x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq -x_1,\ x_2' = x_2 - x_1,\ x_3' = x_3 + x_1 - 2\}.$$

`LLRFSYN` is called with $\mathcal{Q}$, and then, in the first iteration of the while loop, at Line 4 it finds the non-trivial BG-QLRF $\rho_1(x_1, x_2, x_3) = x_2$ for $\mathcal{Q}$, at Line 5 it eliminates all transitions for which $x_2 - x_2' = 0$, and appends ρ_1 to τ. In the next iteration, at Line 4 it finds the non-trivial BG-QLRF $\rho_2(x_1, x_2, x_3) = x_3$ for $\mathcal{Q} \wedge x_2 - x_2' = 0$, at Line 5 it eliminates all transitions for which $x_3 - x_3' = 0$, which results in an empty set, and appends ρ_2 to τ. Since the set of transitions is empty, we exit the while loop and arrive at Line 10 with the weak BG-LLRF $\langle x_2, x_3 \rangle$. Converting it to an LLRF results in the same tuple, as it is already a BG-LLRF in this case.

Example 4.13. Let us demonstrate the algorithm on the *MLC* loops of Example 4.9. `LLRFSYN` is called with $\mathcal{Q}_1, \mathcal{Q}_2$, and then, in the first iteration of the while loop, at Line 4 it finds the non-trivial BG-QLRF $\rho_1(x_1, x_2) = x_1$, at Line 5 it eliminates all transitions for which $x_1 - x_1' = 0$, which eliminates $\mathcal{Q}_1$ and leaves $\mathcal{Q}_2$ unchanged, and appends ρ_1 to τ. In the next iteration, at Line 4 it finds the non-trivial BG-QLRF $\rho_2(x_1, x_2) = x_2$ for $\mathcal{Q}_2$, at Line 5 it eliminates all transitions for which $x_2 - x_2' = 0$, which eliminates $\mathcal{Q}_2$, and appends ρ_2 to τ. Since both paths were eliminated, we exit the while loop and arrive at Line 10 with the weak BG-LLRF $\langle x_1, x_2 \rangle$. Converting it to an LLRF results in the same tuple, as it is already a BG-LLRF in this case. Applying `LLRFSYN` to the *MLC* loop of Example 2.6 fails in the first iteration, because $\mathcal{Q}_1, \mathcal{Q}_2$ does not have a BG-QLRF.

As for the upper bound on the depth of BG-LLRFs, Ben-Amram and Genaim (2014) show that it is n, the number of variables.

Theorem 4.9 ((Ben-Amram and Genaim, 2014))**.** If there is a BG-LLRF for a given *MLC* loop $\mathcal{Q}_1, \ldots, \mathcal{Q}_k$, then there is one with at most n components.

Let us now consider the integer case. A complete algorithm for synthesising BG-QLRFs for $I(\mathcal{Q}_1), \ldots, I(\mathcal{Q}_k)$ can be obtained by applying the one of the rational case on the corresponding integer hulls $(\mathcal{Q}_1)_I, \ldots, (\mathcal{Q}_k)_I$.

OBSERVATION 4.10 ((Ben-Amram and Genaim, 2014))**.** The integer *MLC* loop $I(\mathcal{Q}_1), \ldots, I(\mathcal{Q}_k)$ has a BG-LLRF of depth d, if and only if $(\mathcal{Q}_1)_I, \ldots, (\mathcal{Q}_k)_I$ has a (weak) BG-LLRF of depth d.

Using this observation, synthesising BG-QLRFs for $I(\mathcal{Q}_1), \ldots, I(\mathcal{Q}_k)$ can be done by applying the algorithm of the rational case on the corresponding integer hulls $(\mathcal{Q}_1)_I, \ldots, (\mathcal{Q}_k)_I$, however, one needs to guarantee that when reducing $(\mathcal{Q}_j)_I$ to $(\mathcal{Q}_j)_I \wedge \Delta\rho(\boldsymbol{x}'') = 0$, we still have an integer polyhedron. This is indeed the case since $(\mathcal{Q}_j)_I \wedge \Delta\rho(\boldsymbol{x}'') = 0$ is a face of $(\mathcal{Q}_j)_I$. The runtime is (in the worst case) exponential since computing the integer hull takes exponential time.

Theorem 4.11 ((Ben-Amram and Genaim, 2014))**.** There is a complete exponential-time algorithm for finding a BG-LLRF of minimal depth, if one exists, for a given integer *MLC* loop $I(\mathcal{Q}_1), \ldots, I(\mathcal{Q}_k)$.

The *decision problem* for integer loops is `coNP`-complete, as for LRFs. This result follows from the following characterisation that is related to Observation 4.7:

Theorem 4.12. There is no BG-LLRF for $I(\mathcal{Q}_1), \ldots, I(\mathcal{Q}_k)$, if and only if there is $T \subseteq I(\mathcal{Q}_1) \cup \cdots \cup I(\mathcal{Q}_k)$ for which there is no non-trivial BG-QLRF.

This characterisation facilitates the construction of witnesses against the existence of a BG-LLRF. In fact they are witnesses against the existence of a non-trivial BG-QLRF for a subset of the transitions. The form of such witnesses is similar to what we have shown for LRFs.

The problem of seeking a BG-LLRF when provided a polyhedral set of initial states $\mathcal{S}_0$ is similar to what we have described for the case of LRFs. Namely, we first infer a supporting invariant and add it to the transition relations of the different paths, and then apply the algorithm described above to find the different components of the BG-LLRF. We could also use the template approach to infer a supporting invariant and a BG-QLRF simultaneously. However, in this case, it is important to note that the invariants should always consider the original *MLC* loop, and not just transitions that have not been eliminated so far (Brockschmidt *et al.*, 2013; Larraz *et al.*, 2014). Another obstacle for the template approach is that it is not clear how to select an optimal BG-QLRF since the constraints are non-linear, and thus, unlike the case of LRFs, completeness is not guaranteed even in the case of $\mathbb{R}$. Regarding the complexity of the related decision problems, nothing is known beyond the lower bound results for LRFs.

Inferring BG-LLRFs for CFGs can be done similarly to what we have explained for the *MLC* loop case, where in every iteration we find a BG-QLRF for the transition relations of all remaining edges, and then eliminate transitions that are ranked. As explained with respect to LRFs, we can seek a BG-QLRF where each node is assigned a (possibly) different ρ_ℓ, or seek BG-QLRFs at the level of SCCs. The complexity of the related decision problems, in both approach, and without restricting to an initial state, are the same as the case of *MLC* loops. Handling the case of initial states is done as explained above for *MLC* loops, in particular, the inference of invariants must always consider the original CFG (Brockschmidt *et al.*, 2013; Larraz *et al.*, 2014) and not only the parts that are currently under consideration.

Example 4.14. Consider the CFG depicted in Figure 2.2, and let us demonstrate how to synthesise a BG-LLRF. We first do it for the entire CFG and then at the level of SCCs. In both cases we assume that invariants have been added to the corresponding transition relations.

In a first step, we consider all transition relations of the CFG, where each node is assigned a (template) function $\rho(x, y, z) = \lambda_{\ell,1}x + \lambda_{\ell,2}y + \lambda_{\ell,3}z + \lambda_{\ell,0}$. We find the following optimal BG-QLRF:

$$\begin{array}{lll} \rho_{\ell_0}(x,y,z) = 2x+3 & \rho_{\ell_2}(x,y,z) = 2x+2 & \rho_{\ell_4}(x,y,z) = 2x+2 \\ \rho_{\ell_1}(x,y,z) = 2x+2 & \rho_{\ell_3}(x,y,z) = 2x+1 & \rho_{\ell_5}(x,y,z) = 2x+1 \end{array}$$

This BG-QLRF is decreasing on all transitions of $\mathcal{Q}_0$, $\mathcal{Q}_2$, $\mathcal{Q}_5$, $\mathcal{Q}_7$, and $\mathcal{Q}_8$, and thus it eliminates the corresponding edges. Seeking a BG-QLRF for what is left of the CFG (*i.e.*, $\mathcal{Q}_1$, $\mathcal{Q}_3$, $\mathcal{Q}_4$, and $\mathcal{Q}_6$) we find the following BG-QLRF that is decreasing on remaining transition relations:

$$\begin{array}{ll} \rho_{\ell_1}(x,y,z) = 3y+2 & \rho_{\ell_3}(x,y,z) = z-y \\ \rho_{\ell_2}(x,y,z) = 3y+1 & \rho_{\ell_4}(x,y,z) = 3y+3 \end{array}$$

Now we are left with no edges, and thus we have the following BG-LLRF (a tuple for each node, where those of ℓ_0 and ℓ_5 were complemented with 0 components for clarity):

$$\begin{array}{llllll} \ell_0: & \langle 2x+4, 0\rangle & \ell_2: & \langle 2x+3, 3y+1\rangle & \ell_4: & \langle 2x+3, 3y+3\rangle \\ \ell_1: & \langle 2x+3, 3y+2\rangle & \ell_3: & \langle 2x, z-y\rangle & \ell_5: & \langle 2x+2, 0\rangle \end{array}$$

Let us now consider the approach that works at the level of the SCCs. We start by seeking a BG-QLRF for the single SCC of $\mathcal{Q}_1$, $\mathcal{Q}_2$, $\mathcal{Q}_3$, $\mathcal{Q}_4$, and $\mathcal{Q}_5$. We find the optimal BG-QLRF $\rho_1(x, y, z) = x + 1$ which is decreasing on all transitions of $\mathcal{Q}_5$, and thus eliminates the corresponding edge and splits the SCC into two: the one of $\mathcal{Q}_4$, and the one of $\mathcal{Q}_1, \mathcal{Q}_3$ and $\mathcal{Q}_6$. For the first one we find the BG-QLRF $\rho_2(x, y, z) = z - y$ which eliminates $\mathcal{Q}_4$, and for the second we find the BG-LLRF $\rho_1(x, y, z) = y + 1$ which eliminates $\mathcal{Q}_3$ and leaves us without cycles and thus we proved termination. Note that when seeking BG-QLRFs at the level of SCCs, it is not always needed to use different function for the different nodes, since unlike LRFs, QLRFs are not required to decrease on all transitions.

4.2.2 ADFG-LLRFs

The following definition of an LLRF is due to Alias *et al.* (2010), which is obtained[3] by strengthening the one of BG-LLRF to require all components to be non-negative on all transitions—this is reflected in (4.30) of Definition 4.7 when compared to (4.24) of Definition 4.5.

Definition 4.7. Given an *MLC* loop $\mathcal{Q}_1, \dots, \mathcal{Q}_k \subseteq \mathbb{Q}^{2n}$, we say that $\tau = \langle \rho_1, \dots, \rho_d \rangle$ is an ADFG-LLRF (of depth d) for the loop, if for every $\boldsymbol{x}'' \in \mathcal{Q}_1 \cup \dots \cup \mathcal{Q}_k$ there is an index i such that:

$$\forall j \le d \ . \qquad \rho_j(\boldsymbol{x}) \ge 0 \, , \tag{4.30}$$

$$\forall j < i \ . \ \Delta\rho_j(\boldsymbol{x}'') \ge 0 \, , \tag{4.31}$$

$$\Delta\rho_i(\boldsymbol{x}'') \ge 1 \, . \tag{4.32}$$

We say that $\boldsymbol{x}''$ is *ranked by* ρ_i (for the minimal such i).

BG-LLRFs are more powerful than ADFG-LLRFs.

Example 4.15. Loop (4.27) of Example 4.10 does not have an ADFG-LLRF, while it has a BG-LLRF. The *MLC* of Example 4.9 has an ADFG-LLRF $\tau = \langle x_1, x_2 \rangle$. The *MLC* loop of Example 2.6 does not have an ADFG-LLRF (recall that it does not have a BG-LLRF as well).

Definition 4.8. Let $\mathcal{Q}_1, \dots, \mathcal{Q}_k$ be an *MLC* loop. We say that an affine linear function ρ is an ADFG-QLRF for $\mathcal{Q}'_1 \cup \dots \cup \mathcal{Q}'_k \subseteq \mathbb{Q}^{2n}$, where $\mathcal{Q}'_i \subseteq \mathcal{Q}_i$, if the following holds:

$$\forall \boldsymbol{x}'' \in \mathcal{Q}_1 \cup \dots \cup \mathcal{Q}_k \ . \qquad \rho(\boldsymbol{x}) \ge 0 \, , \tag{4.33}$$

$$\forall \boldsymbol{x}'' \in \mathcal{Q}'_1 \cup \dots \cup \mathcal{Q}'_k \ . \ \Delta\rho(\boldsymbol{x}'') \ge 0 \, , \tag{4.34}$$

We say that it is *non-trivial* if, in addition, $\Delta\rho(\boldsymbol{x}'') > 0$ for at least one $\boldsymbol{x}'' \in \mathcal{Q}'_1 \cup \dots \cup \mathcal{Q}'_k$.

When compared to BG-QLRFs as in Definition 4.6, the difference is that ρ is required to be non-negative on all transitions and not only

[3] Chronologically, the work of Alias *et al.* (2010) was developed before that of Ben-Amram and Genaim (2014), but we present them in a reverse order for the sake of the systematic presentation.

on the transitions under consideration. Note that ADFG-QLRFs also have the property that any nonzero conic combination of ADFG-QLRFs ρ_1 and ρ_2 results in an ADFG-QLRF that ranks all transitions that are ranked by ρ_1 and ρ_2, which means that there exists an optimal ADFG-QLRF.

Remark 4.5. Interestingly, Ben-Amram and Genaim (2015) show that all the results (complexity and algorithmic, both over rationals and integers) that we have discussed in Section 4.2.1 for BG-LLRFs, hold also for ADFG-LLRF. The only (trivial) change required is in the procedure that synthesises the QLRFs, to require the QLRF to be non-negative on all transition instead on those under consideration. However, the algorithmic aspects of ADFG-LLRFs as developed in the original work of Alias *et al.* (2010) are different, and shed light on some properties of such LLRFs. We discuss this in the rest of this section.

Alias *et al.* (2010) provide a complete polynomial-time algorithm for finding a non-trivial ADFG-QLRF $\rho(\boldsymbol{x}) = \vec{\lambda}\boldsymbol{x} + \lambda_0$ for a set of transitions defined by a given *MLC* loop $\mathcal{Q}_1, \ldots, \mathcal{Q}_k$. The algorithm is as follows:

1. Set up an LP problem (using Farkas' Lemma) requiring all paths of the input *MLC* loop to entail $\rho(\boldsymbol{x}) \geq 0$, and each path $\mathcal{Q}_j$ to entail $\Delta\rho(\boldsymbol{x}'') \geq \delta_j$, where $0 \leq \delta_j \leq 1$ is a variable.

2. Solve the LP problem by maximising $\sum_{j=0}^{k} \delta_j$, which fixes values for all variables, including $(\vec{\lambda}, \lambda_0)$.

3. If all δ_j are zero in the solution, the algorithm fails, otherwise ρ ranks all paths $\mathcal{Q}_j$ for which $\delta_j = 1$ (each δ_j can be either 0 or 1, since when $0 < \delta_j < 1$ we can always scale ρ up to obtain $\delta_j = 1$).

The run-time of this algorithm is polynomial since it is based on solving a single LP problem of polynomial size.

When the algorithm above is used within Algorithm 1, once ρ has been found at Line 4, Alias *et al.* (2010) eliminate at Line 5 all paths for which $\delta_j = 1$. This also means that the ADFG-LLRF is not weak. The total run-time of Algorithm 1 in this case is polynomial, since it solves at most k LP problems (in at most k iterations of the while loop) of polynomial size (the bit-size of the *MLC* loop does not increase through the iterations, since we only eliminate paths).

The algorithm for synthesising ADFG-QLRFs that we described above is clearly sound, however, its optimality is not clear. Moreover, at Line 5 of Algorithm 1 we eliminate only paths that are completely ranked by ρ, but there might be transitions in other paths that are ranked by ρ that are no eliminated. Thus, completeness and optimality are not immediate to see (*i.e.*, the reason why Algorithm 1 will find an ADFG-LLRF of minimal depth if one exists). Alias *et al.* (2010) show, in a quite elaborate proof, that Algorithm 1 is complete in this case, and will find an ADFG-LLRF of *minimal depth*, if one exists, *i.e.*, it is equivalent to using a procedure that synthesise an optimal ADFG-QLRF similar to that of BG-QLRFs.

Theorem 4.13 ((Alias *et al.*, 2010)). There is a polynomial-time algorithm for finding an ADFG-LLRF of minimal depth, if one exists, for a given rational *MLC* loop $\mathcal{Q}_1, \dots, \mathcal{Q}_k$.

Example 4.16. Let us demonstrate the algorithm on the *MLC* loops of Example 4.9 using the above algorithm for ADFG-QLRFs. `LLRFSYN` is called with $\mathcal{Q}_1, \mathcal{Q}_2$, and then, in the first iteration of the while loop, at Line 4 it finds the non-trivial ADFG-QLRF $\rho_1(x_1, x_2) = x_1$ that ranks $\mathcal{Q}_1$, which is then eliminated at Line 5. In the next iteration, at Line 4 it finds the non-trivial ADFG-QLRF $\rho_2(x_1, x_2) = x_2$ that ranks $\mathcal{Q}_2$, which is then eliminated at Line 5. Since both paths were eliminated, we exit the while loop and arrive at Line 10 with the ADFG-LLRF $\langle x_1, x_2 \rangle$, which is not weak. Applying `LLRFSYN` to the *MLC* loop of Example 2.6 fails in the first iteration, as in the case of BG-LLRFs, because $\mathcal{Q}_1, \mathcal{Q}_2$ does not have an ADFG-QLRF.

As for the upper bound on the depth of ADFG-LLRFs, Alias *et al.* (2010) show that it is $\min(n, k)$. This means that for *SLC* loops, ADFG-LLRFs have the same power as LRFs since $\min(n, 1) = 1$ is an upper bound on the depth of the ADFG-LLRF in this case.

The problem of deciding existence of an ADFG-LLRF of a given depth is simply solved by bounding the number of iterations of the while-loop in Algorithm 1.

The problem of finding an ADFG-LLRF when starting from a polyhedral set of initial set of state $\mathcal{S}_0$, and that of general CFGs are the same as in the case of BG-LLRF. The difference is only in the kind of QLRF that we infer.

Remark 4.6. Let us change the algorithm of ADFG-LLRF as described above, to require the ADFG-QLRF to be non-negative only on the transitions under considerations instead of all transitions, but still work at the level of paths. We get a new kind of LLRFs that are weaker than BG-LLRFs and stronger than ADFG-LLRFs. The definition would be like BG-LLRFs, but requires each path to be completely ranked by some ρ_i. We believe that this definition of LLRFs was been used by Alias *et al.* (2010), despite of being more intuitive, because they wanted the LLRFs to satisfy additional properties that would allow them to construct a bound on the number of execution steps. This definition has been used by Brockschmidt *et al.* (2016) for inferring complexity bounds.

4.2.3 BMS-LLRFs

The next type of LLRFs is due to Bradley *et al.* (2005a), which is more general than ADFG-LLRFs and not comparable to BG-LLRF (*i.e.*, there are loops that have one kind of LLRF but not the other).

Definition 4.9. Given an *MLC* $\mathcal{Q}_1 \cup \cdots \cup \mathcal{Q}_k \subseteq \mathbb{Q}^{2n}$, we say that $\tau = \langle \rho_1, \ldots, \rho_d \rangle$ is a BMS-LLRF (of depth d) for the loop, if for every $\mathcal{Q}_\ell$ there is $1 \leq i \leq d$ such that the following hold for any $\boldsymbol{x}'' \in \mathcal{Q}_\ell$

$$\forall j < i \,.\, \Delta\rho_j(\boldsymbol{x}'') \geq 0\,, \tag{4.35}$$

$$\rho_i(\boldsymbol{x}) \geq 0\,, \tag{4.36}$$

$$\Delta\rho_i(\boldsymbol{x}'') \geq 1\,. \tag{4.37}$$

We say that $\mathcal{Q}_\ell$ is *ranked by* ρ_i (for the minimal such i).

Note that that it explicitly associates paths to components of the BMS-LLRF. Recall that such association of paths and components was implicit in ADFG-LLRF for *MLC* loops (*i.e.*, it is not explicit in Definition 4.7, but rather implied by the ADFG-QLRF algorithm of Alias *et al.* (2010)).

Example 4.17. Consider an *MLC* loop $\mathcal{Q}_1, \ldots, \mathcal{Q}_4$ where:

$$\begin{array}{lll} \mathcal{Q}_1 = & \{x \geq 0, & x' \leq x-1, y'=y, z'=z\} \\ \mathcal{Q}_2 = & \{x \geq 0, z \geq 0, & x' \leq x-1, y'=y, z' \leq z-1\} \\ \mathcal{Q}_3 = & \{y \geq 0, z \geq 0, & x'=x, y' \leq y-1, z' \leq z-1\} \\ \mathcal{Q}_4 = & \{y \geq 0, & x'=x, y' \leq y-1, z'=z\} \end{array} \tag{4.38}$$

It has the BMS-LLRF $\langle x, y\rangle$, but it has no BG-LLRF, and thus no ADFG-LLRF, due to the simple fact that there is no linear function that is non-negative on all enabled states, and thus we cannot find a corresponding BG-QLRF. On the other hand, the loop of Example 4.10 has a BG-LLRF but not a BMS-LLRF. This shows that these two kinds of LLRFs have different power. The loop of Example 2.6 has the BMS-LLRF $\langle x_1, x_2\rangle$, but not an ADFG-LLRF nor a BG-LLRF. The loop of Example 4.9 has the BMS-LLRF $\langle x_1, x_2\rangle$, which is also an ADFG-LLRF and a BG-LLRF.

Definition 4.10. Let $\mathcal{Q}_1, \cdots, \mathcal{Q}_k$ be an *MLC* loop. We say that an affine linear function ρ is a BMS-QLRF for $\mathcal{Q}'_1 \cup \cdots \cup \mathcal{Q}'_k \subseteq \mathbb{Q}^{2n}$, where $\mathcal{Q}'_i \subseteq \mathcal{Q}_i$, if the following holds for all $\boldsymbol{x}'' \in \mathcal{Q}'_1 \cup \cdots \cup \mathcal{Q}'_k$:

$$\Delta\rho(\boldsymbol{x}'') \geq 0 \tag{4.39}$$

We say that it is *non-trivial* if for at least one $\mathcal{Q}'_\ell$ it is an LRF.

Unlike BG- and ADFG-QLRFs, existence of an optimal BMS-QLRF is not guaranteed, because a nonzero conic combination of BMS-QLRFs ρ_1 and ρ_2 is not guaranteed to rank all paths ranked by ρ_1 and ρ_2.

Example 4.18. Considering all paths of Loop (4.38): $\rho(x, y, z) = x$, $\rho(x, y, z) = y$, and $\rho(x, y, z) = z$ are all BMS-QLRFs. However combinations such as $x+y$, $x+z$ or $x+y+z$ are not, since they do not rank any complete path.

Bradley *et al.* (2005a) provide a complete polynomial-time algorithm for finding a non-trivial BMS-QLRF $\rho(\boldsymbol{x}) = \vec{\lambda}\boldsymbol{x} + \lambda_0$ for a set of transitions defined by a given *MLC* loop $\mathcal{Q}_1, \ldots, \mathcal{Q}_k$ that, in brief, works as follows: it iterates over all paths, and in each iteration checks whether there is a non-trivial BMS-QLRFs that ranks the current path

$\mathcal{Q}_j$. This is done by setting a LP problem (using Farkas' Lemma) requiring all paths to entail $\Delta\rho(\boldsymbol{x}'') \geq 0$, and $\mathcal{Q}_j$ to entails $\Delta\rho_i(\boldsymbol{x}'') \geq 1$ and $\rho_i(\boldsymbol{x}) \geq 0$; any solution to this problem fixes $(\vec{\lambda}, \lambda_0)$, and thus ρ. If no such path is found the algorithm fails. The runtime of the algorithm is polynomial since it solves at most k LP problems of polynomial size wrt. to the size of the input *MLC* loop.

When this algorithm is used within Algorithm 1, once ρ has been found at Line 4, Bradley *et al.* (2005a) eliminate the path $\mathcal{Q}_j$ (*i.e.*, the one that is completely ranked by ρ). This also means that the BMS-LLRF is not weak.

It is easy to see that if Algorithm 1 returns a tuple τ, in this case, then it is a BMS-LLRF, and, moreover, completeness is guaranteed because: (1) it terminates, since in each iteration we eliminate at least one path; and (2) when it returns NONE, then there is indeed no BMS-LLRF for the loop because it has found a subset of transitions for which there is no BMS-QLRF (see Observation 4.7). The overall runtime is still polynomial since we have at most k iterations, and each iteration requires polynomial time to find a non-trivial BMS-QLRF. However, this algorithm is not guaranteed to return a BMS-LLRF of minimal depth, since there is no optimal choice for BMS-QLRFs.

Example 4.19. Consider Loop 4.38. In the first iteration we could use the BMS-QLRF $\rho(x, y, z) = x$ to eliminate the paths $\mathcal{Q}_1$ and $\mathcal{Q}_2$, and in the second iteration we could use the BMS-QLRF $\rho(x, y, z) = y$ to eliminate the remaining paths $\mathcal{Q}_3$ and $\mathcal{Q}_4$. This results in the BMS-LLRF $\langle x, y\rangle$. Note that since there is no optimal BMS-QLRF, this choice will affect the length of the final BMS-LLRF. For example, if in the first iteration we choose the BMS-QLRF $\rho(x, y, z) = z$, we eliminate paths $\mathcal{Q}_2$ and $\mathcal{Q}_3$; but then there is no single BMS-QLRF that eliminates both paths $\mathcal{Q}_1$ and $\mathcal{Q}_4$, so we have to use $\rho(x, y, z) = x$ to eliminate $\mathcal{Q}_1$ and $\rho(x, y, z) = y$ to eliminate $\mathcal{Q}_4$. This results in the BMS-LLRF $\langle z, x, y\rangle$ which has a different length.

Theorem 4.14 ((Bradley *et al.*, 2005a))**.** There is a polynomial-time algorithm for finding a BMS-LLRF, if one exists, for a rational *MLC* loop.

Let us now consider the integer case. First observe that a complete algorithm for synthesising BMS-QLRFs for $I(\mathcal{Q}_1), \ldots, I(\mathcal{Q}_k)$ can be done by applying the one of the rational case on $(\mathcal{Q}_1)_I, \ldots, (\mathcal{Q}_k)_I$. Then, the following observation helps us to adapt the overall algorithm for rational loop to handle integer loops.

OBSERVATION 4.15 ((Ben-Amram and Genaim, 2015))**.** The integer *MLC* loop $I(\mathcal{Q}_1), \ldots, I(\mathcal{Q}_k)$ has a BMS-LLRF of depth d, if and only if $(\mathcal{Q}_1)_I, \ldots, (\mathcal{Q}_k)_I$ has a BMS-LLRF of depth d.

Using this observation, synthesising BMS-QLRFs for the integer *MLC* loop $I(\mathcal{Q}_1), \ldots, I(\mathcal{Q}_k)$ can be done by applying the algorithm of the rational case on $(\mathcal{Q}_1)_I, \ldots, (\mathcal{Q}_k)_I$. Completeness is guaranteed since we eliminate a complete path in each iteration, and thus all paths remain integral through the iterations of the while-loop. The runtime is exponential since computing the integer-hull is exponential.

Theorem 4.16. There is an exponential-time algorithm for finding a BMS-LLRF, if one exists, for an integer *MLC* loop $I(\mathcal{Q}_1), \ldots, I(\mathcal{Q}_k)$.

Ben-Amram and Genaim (2015) show that the corresponding *decision problem* for integer loops is `coNP`-complete, which results from a similar characterisation of Theorem 4.12 for the case of BG-LLRFs and facilitates the construction of witnesses against the existence of a BMS-QLRF for a subset of the transitions.

An upper bound on the depth of BMS-LLRFs is clearly given by k; the number of paths. Moreover, Ben-Amram and Genaim (2015) show that this bound is tight, *i.e.*, there are k-path loops for which we need k components. Moreover, they show that it is possible for a loop to have LLRFs of all variants that we have seen so far, but such that the minimal depths differ.

Example 4.20. Consider an *MLC* loop specified by the following paths

$$\begin{aligned}
\mathcal{Q}_1 &= \left\{\begin{matrix} r \ge 0, & & t \ge 0, & x \ge 0, & & z \ge 0, & w \ge 0, \\ r' < r, & & t' < t, & & & & \end{matrix}\right\} \\
\mathcal{Q}_2 &= \left\{\begin{matrix} r \ge 0, & s \ge 0, & t \ge 0, & x \ge 0, & & z \ge 0, & w \ge 0, \\ r' = r, & s' < s, & t' < t, & & & & \end{matrix}\right\} \\
\mathcal{Q}_3 &= \left\{\begin{matrix} r \ge 0, & s \ge 0, & t' = t & x \ge 0, & & z \ge 0, & w \ge 0, \\ r' = r, & s' = s, & & x' < x, & & & \end{matrix}\right\} \\
\mathcal{Q}_4 &= \left\{\begin{matrix} r \ge 0, & s \ge 0, & t' = t & x \ge 0, & y \ge 0, & z \ge 0, & w \ge 0, \\ r' = r, & s' = s, & & x' = x, & y' < y, & z' < z, & \end{matrix}\right\} \\
\mathcal{Q}_5 &= \left\{\begin{matrix} r \ge 0, & s \ge 0, & t' = t & x \ge 0, & y \ge 0, & z \ge 0, & w \ge 0, \\ r' = r, & s' = s, & & x' = x, & y' < y, & z' = z, & w' < w \end{matrix}\right\}
\end{aligned}$$

where, for readability, we use $<$ for the relation "smaller at least by 1". This loop has the BMS-LLRF $\langle t, x, y \rangle$, which is neither a BG-LLRF or ADFG-LLRF because t is not lower-bounded on all the paths. Its shortest BG-LLRF is of depth 4, *e.g.*, $\langle r, s, x, y \rangle$, which is not an ADFG-LLRF because y is not lower-bounded on all the paths. Its shortest ADFG-LLRF is of depth 5, *e.g.*, $\langle r, s, x, z, w \rangle$. This reasoning is valid for both integer and rational variables.

Since Algorithm 1 does not return a BMS-LLRF of minimal depth, Ben-Amram and Genaim (2015) study the complexity of finding a BMS-LLRF that satisfies a given bound on the depth.

Theorem 4.17 ((Ben-Amram and Genaim, 2014))**.** Deciding whether there is a BMS-LLRF of depth d for a rational loop $\mathcal{Q}_1, \ldots, \mathcal{Q}_k$, is an `NP`-complete problem, and for an integer loop $I(\mathcal{Q}_1), \ldots, I(\mathcal{Q}_k)$, is a Σ_2^P-complete problem.

The problem of finding a BMS-LLRF when starting from a polyhedral set of initial states $\mathcal{S}_0$, and that for general CFGs, could be addressed as in the case of BG-LLRF. The difference is only in the kind of QLRF that we infer.

4.2.4 MΦRFs

An interesting special case of LLRFs is *multiphase-linear ranking functions* (MΦRFs), which is defined as follows.

Definition 4.11 (MΦRF). Given an *MLC* loop $\mathcal{Q}_1, \ldots, \mathcal{Q}_k \subseteq \mathbb{Q}^{2n}$, we say that $\tau = \langle \rho_1, \ldots, \rho_d \rangle$ is an MΦRF (of depth d) for the loop, if for every $\boldsymbol{x}'' \in \mathcal{Q}_1 \cup \cdots \cup \mathcal{Q}_k$ there is an index i such that:

$$\forall j \leq i \ . \ \Delta\rho_j(\boldsymbol{x}'') \geq 1\,, \tag{4.40}$$

$$\rho_i(\boldsymbol{x}) \geq 0 \tag{4.41}$$

We say that $\boldsymbol{x}''$ is *ranked by* ρ_i (for the minimal such i).

When compared to LLRFs as in Definition 4.3, the difference is that all components ρ_j, with $j < i$ are decreasing rather than non-increasing. It is easy to see that this definition, for $d = 1$, means that ρ_1 is an LRF, and for $d > 1$, it implies that ρ_1 is always decreasing; as long as $\rho_1(\boldsymbol{x}) \geq 0$, transition $\boldsymbol{x}''$ must be ranked by ρ_1, and when $\rho_1(\boldsymbol{x}) < 0$, $\langle \rho_2, \ldots, \rho_d \rangle$ becomes an MΦRF for the rest of the execution. This agrees with the intuitive notion of "phases."

Example 4.21. Consider the following loop:

$$\textit{while } (x \geq -z) \textit{ do } x' = x + y,\ y' = y + z,\ z' = z - 1 \tag{4.42}$$

Clearly, the loop goes through three phases — in the first, z descends, while the other variables may increase; in the second (which begins once z becomes negative), y decreases; in the last phase (beginning when y becomes negative), x decreases. Note that since there is no lower bound on y or on z, they cannot be used in an LRF; however, each phase is clearly finite, as it is associated with a value that is non-negative and decreasing during that phase. In other words, each phase is linearly ranked. Formally, this loop has the MΦRF $\langle z + 1, y + 1, x \rangle$.

Example 4.22. Some loops have multiphase behaviour which is not so evident as in the last example. Consider the following loop

$$\textit{while } (x \geq 1,\ y \geq 1,\ x \geq y,\ 4y \geq x) \textit{ do } x' = 2x,\ y' = 3y \tag{4.43}$$

It has the MΦRF $\langle x - 4y, x - 2y, x - y \rangle$.

Definition 4.12. Let $\mathcal{Q}_1, \ldots, \mathcal{Q}_k$ be an *MLC* loop. We say that an affine linear function ρ is an MΦ-QLRF for $\mathcal{Q}'_1 \cup \cdots \cup \mathcal{Q}'_k \subseteq \mathbb{Q}^{2n}$, where $\mathcal{Q}'_i \subseteq \mathcal{Q}_i$, if the following holds for all $\boldsymbol{x}'' \in \mathcal{Q}'_1 \cup \cdots \cup \mathcal{Q}'_k$:

$$\Delta\rho(\boldsymbol{x}'') \geq 1 \tag{4.44}$$

We say that it is *non-trivial* if, in addition, $\rho(\boldsymbol{x}) \geq 0$, for at least one $\boldsymbol{x}'' \in \mathcal{Q}'_1 \cup \cdots \cup \mathcal{Q}'_k$.

Unlike BG- and ADFG-LLRFs, the existence of optimal MΦ-QLRF is not guaranteed because a non-zero conic combination of MΦ-QLRFs ρ_1 and ρ_2 is not guaranteed to rank all transitions ranked by ρ_1 and ρ_2.

A polynomial-time algorithm for synthesising MΦ-QLRFs $\rho(\boldsymbol{x}) = \vec{\lambda}\boldsymbol{x} + \lambda_0$ can be as follows:

1. Set up an LP problem $\mathcal{S}_d$ (resp. $\mathcal{S}_p$), using Farkas' Lemma, requiring all paths to imply $\Delta\rho(\boldsymbol{x}'') \geq 1$ (resp. $\rho(\boldsymbol{x}) \leq 0$); and
2. Choose a point $(\vec{\lambda}, \lambda_0)$ from $\mathcal{S}_d$ that is not in $\mathcal{S}_p$, which can be done by iterating over the inequalities $\vec{a}_i\boldsymbol{x}'' \leq b_i$ of $\mathcal{S}_p$, and picking a point from $\mathcal{S}_d \wedge \vec{a}\boldsymbol{x}'' > b$ if it is not empty.

Incorporating such a procedure at Line 4 of Algorithm 1, and eliminating all transition for which $\rho(x) > 0$ at Line 5, we obtain a sound procedure for synthesising MΦRFs for *MLC* loops, however completeness in not guaranteed since the algorithm might not terminate. Note that the MΦRF we build is not weak.

Unlike other kinds of LLRFs, that we have seen in the previous sections, there are almost no results on complexity and algorithmic aspects of MΦRFs for *MLC* loops. However, when fixing the depth d, Leike and Heizmann (2015) and Li *et al.* (2016) propose complete solutions for MΦRFs over $\mathbb{R}$. Both rely on the template-based approach, that we have described at the beginning of Section 4.2, which turns the requirements of Definition 4.11, *for a fixed* d, into a set of existential constraints – this gives us a `PSPACE` upper bound, since the existential theory of the reals can be decided in polynomial space (Canny, 1988).

For *SLC* loops, Ben-Amram and Genaim (2017) show that the template-based approach, for seeking an MΦRF for a fixed d, can be performed in polynomial time by avoiding the generation of non-linear constraints. This is done by showing that MΦRFs and a further subclass of MΦRFs called *nested ranking functions* (NLRFs), that was introduced by Leike and Heizmann (2015) and can be synthesised in polynomial time, have the same power for *SLC* loops, *i.e.*, an *SLC* loop has an MΦRF of depth d if and only if it has an NLRF of depth d.

Definition 4.13 (NLRF)**.** Given an *SLC* loop $\mathcal{Q} \subseteq \mathbb{Q}^{2n}$, we say that $\tau = \langle \rho_1, \dots, \rho_d \rangle$ is a *nested ranking function* (of depth d) for $\mathcal{Q}$ if the following requirements are satisfied for all $\boldsymbol{x}'' \in \mathcal{Q}$:

$$\rho_d(\boldsymbol{x}) \geq 0 \tag{4.45}$$

$$\Delta\rho_i(\boldsymbol{x}'') + \rho_{i-1}(\boldsymbol{x}) \geq 1 \qquad \text{for all } i = 1, \dots, d. \tag{4.46}$$

where for uniformity we let $\rho_0(\boldsymbol{x}) = 0$.

It is easy to see that an NLRF is an MΦRF. Indeed, ρ_1 is decreasing, and when it becomes negative ρ_2 starts to decrease, *etc*. In addition, the loop must stop by the time that the last component becomes negative, since ρ_d is non-negative on all enabled states. Note that the above definition extends also to *MLC* loops.

Example 4.23. Consider Loop (4.42). It has the MΦRF $\langle z+1, y+1, x \rangle$ which is not nested because, among other things, last component x might be negative, *e.g.*, for the state $x = -1, y = 0, z = 1$. However, it has the NLRF $\langle z+1, y+1, z+x \rangle$.

The above example shows that there are MΦRFs which are not NLRFs, however, for *SLC* loops Ben-Amram and Genaim (2017) provide a procedure to construct an NLRF from a given MΦRF.

Theorem 4.18 ((Ben-Amram and Genaim, 2017))**.** *If a rational SLC loop $\mathcal{Q} \subseteq \mathbb{Q}^{2n}$ has an MΦRF of depth d, then it has an NLRF of depth d.*

This gives us a complete polynomial-time procedure to determine whether a given *SLC* loop $\mathcal{Q}$ has an MΦRF, which is done by synthesising an NLRF $\tau = \langle \rho_1, \dots, \rho_d \rangle$, where $\rho_i(\boldsymbol{x}) = \vec{\lambda}_i \boldsymbol{x} + \lambda_{i,0}$, as follows:

1. Set a LP problem (using Farkas' Lemma) requiring $\mathcal{Q}$ to imply (4.45,4.46), which generates a set of linear constraints over the variables $(\vec{\lambda}_i, \lambda_{i,0})$ and some other variables for the Farkas' coefficients; and

2. Any solution of this LP problem fixes values for $(\vec{\lambda}, \lambda_0)$ and thus define τ. Moreover, if there is no solution then $\mathcal{Q}$ does not have an NLRF.

This give us the following theorem.

Theorem 4.19. There is a polynomial-time algorithm that, given an *SLC* loop $\mathcal{Q}$ and a depth-bound d, determines whether a depth-d MΦRF exists for $\mathcal{Q}$ and finds its coefficients if one exists.

Ben-Amram and Genaim (2017) also show that, for the class of *SLC* loop, NLRFs have the same power as LLRFs of Definition 4.3, and thus for LLRFs, too, we have a complete solution in polynomial time (over the rationals).

Theorem 4.20 ((Ben-Amram and Genaim, 2017))**.** If $\mathcal{Q}$ has an LLRF of depth d, it has an MΦRF of depth d.

We next consider integer loops. The following results are by Ben-Amram and Genaim (2017).

Theorem 4.21. $I(\mathcal{Q})$ has an MΦRF of depth d if and only if $\mathcal{Q}_I$ has an MΦRF of depth d (as a rational loop).

This gives us a solution of exponential time complexity, because computing the integer hull requires exponential time. However, it is polynomial for the cases in which the integer hull can be computed in polynomial time (Ben-Amram and Genaim, 2014, Sect. 4). The next theorem shows that the exponential time complexity is unavoidable for the general case (unless $\mathtt{P} = \mathtt{NP}$).

Theorem 4.22. Existence of an MΦRF of depth d for a given integer *SLC* loop is a $\mathtt{coNP}$-complete problem.

We are not aware of a computable upper bound on the depth of MΦRF, given the loop. Ben-Amram and Genaim (2017) show that such a bound cannot depend only on the number of variables or paths of the loop, but must also take account of the coefficients and the constants used in the inequalities defining the loop.

Example 4.24. For integer $B > 0$, Ben-Amram and Genaim (2014) show that the following *SLC* loop

$$\mathtt{while}\ (x \geq 1,\ y \geq 1,\ x \geq y,\ 2^B y \geq x)\ \mathtt{do}\ x' = 2x,\ y' = 3y$$

needs at least $B+1$ components in any MΦRF, and that this bound $B+1$ is tight and confirmed by the MΦRF $\langle x-2^By, x-2^{B-1}y, x-2^{B-2}y, \ldots, x-y\rangle$.

Ben-Amram and Genaim (2017) also discuss the consequence of existence of MΦRFs on the number of iterations that an *SLC* loop can make, and show that it is actually linear in the input values.

Theorem 4.23. An *SLC* loop that has an MΦRF terminates for an input $\boldsymbol{x}_0$ in a number of iterations bounded by $O(\|\boldsymbol{x}_0\|_\infty)$.

In a subsequent work, Ben-Amram *et al.* (2019) attempted to solve the general MΦRF problem for *SLC* loops, *i.e.*, without a given bound on the depth. Although the problem remains open, this attempt yielded several important observations. They first observe that if an *SLC* loop has an irredundant MΦRF of depth d, then it has one of the same depth in which the last component ρ_d is non-negative over all enabled states of $\mathcal{Q}$. Using this observation they propose an algorithm that builds a MΦRF recursively starting from the last component, which always find an MΦRF if one exists, however, it might not terminate in other cases. The algorithm can also, in some cases, find witnesses for non-termination when it fails to find a MΦRF.

Ben-Amram *et al.* (2021) demonstrate the usefulness of the algorithm described above for studying properties of *SLC* loops, in particular, it is used to characterise kinds of *SLC* loops for which there is always an MΦRF, if the loop is terminating, and thus have linear run-time complexity. This is done for *octagonal relations* and *affine relations with the finite-monoid property*—for both classes, termination has been proven decidable (Bozga *et al.*, 2014). In addition, they provide a bound on the depth of MΦRFs for these classes of *SLC* loops, which can be used to make the above algorithm complete.

The problem of finding an MΦRF when starting from a polyhedral set of initial states $\mathcal{S}_0$, and that for general CFGs, are the same as in the case of BG-LLRF. The difference is only in the kind of QLRF that we infer.

4.2.5 Other Approaches for LLRFs

The earliest work that we know that addressed the generation of LLRFs is by Feautrier (1992b), where they are called *multidimensional schedules*. Colón and Sipma (2002) use LP methods based on the computation of polars. The LLRF is not constructed explicitly but can be inferred from the results of their algorithm. Bradley *et al.* (2005d) introduced the notion of *Polyranking principle* which is based on lexicographic ranking functions where each component is an NLRF of depth at most 2. In another work, Bradley *et al.* (2005c) considered *MLC* loops with polynomial transitions and the synthesis of lexicographic-polynomial ranking functions. All the works by this group actually tackle an even more complex problem, since they also search for *supporting invariants*, based on the transition constraints and on given preconditions.

Urban and Miné (2014) compute lexicographic ranking functions using abstract interpretation. Gonnord *et al.* (2015) compute LLRFs, essentially ADFG-LLRFs, for complete programs, including a computation of invariants. Their method is designed to improve over both the efficiency and the effectiveness of previous methods, such as Alias *et al.* (2010) and Gulwani and Zuleger (2010). Yuan *et al.* (2021) suggest an approach to the problem of bounding the depth of MΦRFs. Zhu *et al.* (2016) consider a type of LLRFs that combines BMS-LLRFs with the idea of “phases”. It is a special case of general LLRF, but one for which we have an (exponential) complete algorithm.

4.3 Other Types of Ranking Functions

Another type of ranking function that may be interesting in the context of linear programs is *piecewise linear ranking functions* (Urban, 2013). We are not aware of complexity results for this type of functions, for linear-constraint loops like the ones we address in this survey. Also beyond the scope of this survey are *polynomial ranking functions* (Neumann *et al.*, 2020; Shen *et al.*, 2013; Chen *et al.*, 2007; Cousot, 2005).

Zhu and Kincaid (2024) develop a complete (in some sense) algorithm for synthesising (lexicographic) polynomial ranking functions for simple loops that may include non-linear constraints in their description (thus, more general than *SLC* loops).

Leike and Heizmann (2015) present a template-based approach to synthesise many types of ranking functions, including ADFG-LLRFs, piecewise-linear ranking functions and others.

Doménech *et al.* (2019) use control-flow refinement to transform programs with complex control-flow into equivalent simpler ones, which makes it possible, for example, to use LRFs instead LLRFs for proving termination. For example, the loop on the left would be translated into the loop on the right:

```
while(x >= 1)
  if (y <= z-1) y++;
  else x--;
```

```
while(x >= 1 && y <= z-1) y++;
while(x >= 1) x--;
```

The one on the left requires the LLRF $\langle z - y, x \rangle$, while the one on the right requires the LRFs $z-y$ and x. There are also examples that do not admit any kind of ranking function (from those discussed in this section), while after the refinement they do admit LLRFs. Borralleras *et al.* (2017) develop a technique for proving (conditional) termination, which is based on incrementally finding conditional LRFs for the different parts of the program.

Polynomial interpretations are used to prove the termination of term rewriting systems, which are out of the scope of this survey. They are polynomials assigned to each function symbol such that they decrease with every derivation. While they may seem similar to ranking functions, their underlying problems are computationally harder. For example, the problem of deciding whether a single rewriting rule admits a linear interpretation is undecidable (Mitterwallner *et al.*, 2024).

5

Transition Invariants and Difference-Bound Constraints

A key challenge of using ranking functions for termination proofs is that it is not always possible to find a function from a tractable class such as LLRFs, that strictly decreases with every single step of a program's execution. Instead of proving a decrease at every step, we can resort to techniques that prove absence of infinite executions by showing that in any infinite trace, there must be a sub-trace that violates a well-foundedness property. These techniques often rely on the use of *Ramsey's theorem*. This application of Ramsey's theorem was first applied by Geser (1990), and was later applied, in various forms, by several other researchers, including Doornbos and Karger (1998), Lee *et al.* (2001), Dershowitz *et al.* (2001), Codish *et al.* (2003), and Podelski and Rybalchenko (2004b). Blass and Gurevich (2008, Page 2) provide a brief history of this use of Ramsey's theorem.

In this section, we will discuss *disjunctive well-founded transition invariants* (DTI), a technique for proving termination that applies Ramsey's theorem. This method has primarily emerged in the context of linear-constraint programs. We further present classes of linear-constraint programs for which DTI provide a complete criterion for termination—specifically, DTI based on LRFs. These classes (such as size-change terminating programs, monotonicity-constraint programs, *etc*) have been studied from different viewpoints, but our presentation here aims to show how they all fall under the DTI approach.

Organisation of this Section. We start with an overview of transition invariants in Section 5.1. We then discuss several classes of programs: δ-size-change-termination (Section 5.3), size-change-termination (Section 5.4), δ-size-change-termination for fan-in free programs (Section 5.5), monotonicity constraints (Section 5.6), and gap constraints (Section 5.7). We also examine the relation to ranking functions (Section 5.8), the relative power of DTIs (Section 5.9), and finally provide an overview of other related works (Section 5.10).

5.1 Transition Invariants

Given a transition relation $T \subseteq S \times S$, we define $T^i = T^{i-1} \circ T$, for $i \geq 1$, where $T^0 \subseteq S \times S$ is the identity relation and $T_1 \circ T_2 = \{(s, s'') \in S \times S \mid (s, s') \in T_1, (s', s'') \in T_2\}$. The transitive closure of a relation T is defined as $T^+ = \cup_{i \geq 1} T^i$.

The relation T^+ provides crucial information about reachability: a computation under T that starts in s reaches s' if and only if $(s, s') \in T^+$. This concept forms the basis for numerous applications in static analysis and model checking, especially in termination analysis. For termination with respect to an initial set of states, as we have done previously, we assume that T has been reduced to the set of reachable states and then study universal termination.

Instead of working directly with T^+ (which is not always computable, or even representable in any useful form), termination tools resort to approximations known as *transition invariants*.

Definition 5.1 ((Podelski and Rybalchenko, 2004b)). We say that $T_I \subseteq S \times S$ is a *transition invariant* (TI) for $T \subseteq S \times S$, if and only if $T^+ \subseteq T_I$.[1]

[1] Podelski and Rybalchenko (2004b) require $T^+ \subseteq T_I \cap (\mathsf{RCH}(T, S_0) \times \mathsf{RCH}(T, S_0))$, because they consider a set S_0 of initial states.

Recall that a binary relation $T \subset S \times S$ is called *well-founded* if there is no infinite sequence $s_0, s_1, \dots$ such that $(s_i, s_{i+1}) \in T$ for all $i \geq 0$, and that if T is the transition relation of a program, well-foundedness of T is equivalent to (universal) termination.

Definition 5.2. Given a transition relation $T \subseteq S \times S$, and sets $\langle T_1, \dots, T_d \rangle$ of transitions such that $T_i \subseteq S \times S$, we say that $\langle T_1, \dots, T_d \rangle$ is a *disjunctively well-founded transition invariant* (DTI) for T if $T^+ \subseteq T_1 \cup \dots \cup T_d$, and for each $1 \leq i \leq d$, T_i is well-founded.

Theorem 5.1 ((Podelski and Rybalchenko, 2004b))**.** If $T \subseteq S \times S$ has a DTI then T is well-founded.

Proof. Assume that T has the DTI $\langle T_1, \dots, T_d \rangle$ and suppose, for a contradiction, that there is an infinite sequence $s_0, s_1, \dots$ such that $(s_i, s_{i+1}) \in T$ for all $i \geq 0$. For every pair (s_i, s_j) with $i < j$ we must have $(s_i, s_j) \in T_k$ for some T_k. Associating one such k to the pair (i, j) we obtain a colouring of the infinite complete graph with d colours; by Ramsey's theorem, there is an infinite monochromatic clique. This constitutes an infinite subsequence $s_{i_0}, s_{i_1}, \dots$ where $(s_{i_j}, s_{i_{j+1}}) \in T^k$ for all $j \geq 0$, contradicting the well-foundedness of T_k. Note that the converse implication is trivial: if T is well-founded then T^+ is a DTI. □

To make DTI a practical tool for proving termination we need:

1. An effective way to show that the disjuncts are well-founded; and

2. An effective way to show that the disjuncts cover the transitive closure of the transition relation.

This clearly depends, among other things, on the state space S and on the way T and each T_i are specified. In what follows we focus on DTIs for CFGs, and thus assume that the transition relation T corresponds to a (linear-constraint) CFG with variables ranging over $R \in \{\mathbb{R}, \mathbb{Q}, \mathbb{Z}\}$.

Remark 5.1. When a transition relation T originates from a CFG, we can relax the requirements of Definition 5.2 such that instead of computing a DTI that over-approximates T^+, we compute one that over-approximates $T^+|_C = \{((\ell, \boldsymbol{x}), (\ell, \boldsymbol{x}')) \in T^+ \mid \ell \in C\}$ where C

is any feedback vertex set (*i.e.*, removing these vertexes results in an acyclic graph). This is true because $T^+|_C$ is transitively closed, and T^+ is well founded if and only if $T^+|_C$ is well founded (we can easily extend a DTI for $T^+|_C$ to a DTI for T). If the CFG originates from a structured program, C could be the set of locations corresponding to loop heads.

In what follows we assume a given CFG $P = (V, R, L, \ell_0, E)$, where $R \in \{\mathbb{R}, \mathbb{Q}, \mathbb{Z}\}$, and use T_P to refer to the corresponding transition relation. In this context, and for transition relations specified by linear constraints in general, it is common to restrict the DTI to a form in which each T_i is a well-founded convex polyhedron, *i.e.*, a terminating *SLC* loop.

Definition 5.3. $\langle T_1, \dots, T_k \rangle$ is a *polyhedral* DTI for T_P if it is a DTI and each T_i is of the form $T_i = \{((\ell, \boldsymbol{x}), (\ell, \boldsymbol{x}')) \mid (\boldsymbol{x}, \boldsymbol{x}') \in \mathcal{Q}\}$, where $\mathcal{Q}$ is a convex polyhedron and $\ell \in L$. We sometimes write T_i as $(\ell, \mathcal{Q}, \ell)$.

Intuitively, a polyhedral DTI is a termination proof that breaks the task of proving termination for a complex program into a set of proofs for *SLC* loops.

There are DTI-based termination analysis tools (Lindenstrauss and Sagiv, 1997; Codish and Taboch, 1999; Albert *et al.*, 2008; Spoto *et al.*, 2010)[2]. They work in two steps: (1) compute a DTI $T_1 \cup \cdots \cup T_d$ that over-approximates T_P^+, where each T_i is polyhedral as in Definition 5.3; and (2) check that for each $T_i = (\ell, \mathcal{Q}, \ell)$, the *SLC* loop $\mathcal{Q}$ is terminating by seeking a corresponding ranking function, *e.g.*, LRF. This implies that $T_1 \cup \ldots \cup T_d$ is a DTI. Cook *et al.* (2006) follows a different approach, and constructs a DTI incrementally where each component is polyhedral, but has a restricted form as in the following definition.

Definition 5.4. A *linear-ranking function based* DTI (LRF-DTI for short), is a polyhedral DTI as in Definition 5.3 where each transition polyhedron $\mathcal{Q}$ has an LRF, specifically it satisfies $\rho_i(\boldsymbol{x}) \geq 0 \wedge \Delta\rho_i(\boldsymbol{x}'') \geq 1$ for some linear function ρ_i. In what follows we use T_{ρ_i} for the transition relation relation $\{((\ell, \boldsymbol{x}), (\ell, \boldsymbol{x}')) \mid \rho_i(\boldsymbol{x}) \geq 0,\ \Delta\rho_i(\boldsymbol{x}'') \geq 1\}$ (the location ℓ is not important, and will always be clear from the context).

[2]They do not call them DTI, but they are conceptually the same.

The work of Cook *et al.* (2006) has several important observations that make computing a DTI practical, and this paper was influential in promoting the concept of DTI and the use of *SLC* loops as components in a termination proof for a possibly complex program, relying (at least in Cook *et al.* (2006)) on LRFs, instead of using more complex termination proofs such as LLRFs. They describe a method, relying on a program transformation, to compute an over-approximation of T_P^+ using off-the-shelf safety checkers (such checkers are used to prove that a set of (error) states in not reachable, and when they fail they usually provide a counter example). Unlike other algorithms in this survey, this method is not complete for the problem in any sense, but we describe it informally due to its historical importance and as an illustration to how DTIs are used in practice. The rest of this subsection describes this method, while the following subsections are independent of it.

Let us assume that during the execution we can non-deterministically record the current state into (extra) program variables $pc_g, \boldsymbol{x}_g$, where pc_g is used to store the location and $\boldsymbol{x}_g$ to store the value of the program variables $\boldsymbol{x}$. Let us also assume that pc_g has a special value $\ell_\perp$ in the initial state. It is easy to see that when reaching a state $(\ell, \boldsymbol{x})$ and $pc_g \neq \ell_\perp$, it is guaranteed that $((pc_g, \boldsymbol{x}_g), (\ell, \boldsymbol{x})) \in T_P^+$. Moreover, since the recording is done non-deterministically, the opposite also holds: if $((pc_g, \boldsymbol{x}_g), (\ell, \boldsymbol{x})) \in T_P^+$ then there is an execution that reaches the state $(\ell, \boldsymbol{x})$ where the recorded state is $(pc_g, \boldsymbol{x}_g)$. This means that state invariants of the program instrumented with this recording mechanism induce transition invariants for the original program, and thus we can use invariant inference tools to over-approximate T_P^+.

At the level of a CFG, this instrumentation can be done as follows. First we add an extra program variable pc, and for each $(\ell_i, \mathcal{Q}, \ell_j) \in E$ we add $pc = i \wedge pc' = j$ to $\mathcal{Q}$, *i.e.*, variable pc simply tracks the location. Next, we introduce a new set of *ghost* variables $pc_g, \boldsymbol{x}_g$ (used to record a state), and split each edge $(\ell_i, \mathcal{Q}, \ell_j) \in E$ into two edges $(\ell_i, \mathcal{Q}_1, \ell_j)$ and $(\ell_i, \mathcal{Q}_2, \ell_j)$ where: (1) $\mathcal{Q}_1 \equiv \mathcal{Q} \wedge pc_g' = pc_g \wedge \boldsymbol{x}_g' = \boldsymbol{x}_g$, and; (2) $\mathcal{Q}_2 \equiv \mathcal{Q} \wedge pc_g' = pc \wedge \boldsymbol{x}_g' = \boldsymbol{x}$. The purpose of $\mathcal{Q}_2$ is to non-deterministically record the current state into $(pc_g, \boldsymbol{x}_g)$.

The other observation of Cook *et al.* (2006) is that inferring a DTI can be done using an off-the-shelf safety checker that is based on *counter* *example-guided abstraction refinement* approach (CEGAR). We describe this in the next example.

Example 5.1. Let us consider the CFG depicted in Figure 2.2, and we start the execution at ℓ_0 with $\mathcal{S}_0 = \{x \geq 0, y \geq 0\}$. Note that $\{\ell_1, \ell_3\}$ is a feedback vertex set (they correspond to the loop heads of the program in Figure 2.2). Let us assume that the CFG has been instrumented with the recording mechanism as described above. Moreover, we add a new node ℓ_{err} that represents an *error location* that is not connected to the CFG yet. We refer to the condition that allows us to move to ℓ_{err} as the *error condition*.

Next we will proceed iteratively, starting from an empty DTI, where in each iteration: (1) we modify the *error condition* (*i.e.*, how ℓ_{err} is connected to the CFG) to take into account the current DTI; (2)) we use a safety checker to try to prove that ℓ_{err} is unreachable; (3) if we succeed, then as further explained below, this means that the current disjunction is indeed a DTI; otherwise, we use the counter example returned by the safety checker to add a new component T_ρ to the disjunction, if possible, and repeat the process.

In the first step, since the current DTI is empty, we modify the CFG such that whenever ℓ_1 (resp. ℓ_3) is reached with $pc = pc_g = 1$ (resp. $pc = pc_g = 3$), the execution can move to ℓ_{err} (*i.e.*, we add corresponding edges with the corresponding condition). There condition simulate a situation where the execution visit location ℓ_1 (resp. ℓ_3) at least twice. Note that if ℓ_{err} is unreachable, it means that there are no loops in the program and thus the empty disjunction is actually a valid DTI because ℓ_1 and ℓ_3 form a feedback vertex set. Applying a safety checker we get as a counter example an execution path that passes through the nodes ℓ_1, ℓ_2, ℓ_4 and then ℓ_1 again. We treat this cycle as an *SLC* loop, namely: $\{pc = 1,\ pc_g = 1,\ x = x_g,\ x_g \geq 0,\ y_g \geq 0,\ y = y_g - 1,\ z_g = z\}$ (some invariants lave been added). Note that in this *SLC* loop, (x_g, y_g, z_g) is the current state and (x, y, z) is the next state, and that it has the LRF $\rho_1(x_g, y_g, z_g) = y_g$. This leads to adding $T_1 = \{pc = 1,\ pc_g = 1,\ y_g \geq 0,\ y_g - 1 \geq y\}$ to the DTI. The idea is that in the next iteration, this counter example, and possibly others, are eliminated due to T_1.

In the second iteration, we refine the error condition to take T_1 into account, *i.e.*, we allow moving from ℓ_1 to ℓ_{err} if, in addition to $pc = pc_g = 1$, we have $y_g < 0$ or $y_g - 1 < y$. Applying a safety checker we get, as a counter example, an execution path that passes through the nodes $\ell_1, \ell_2, \ell_3, \ell_4$ and then ℓ_1 again. The SLC loop that correspond to this path is $\{pc = 1,\ pc_g = 1,\ x = x_g - 1,\ x_g \geq 0,\ y_g \geq 0,\ z_g = z\}$. This leads to adding $T_2 \equiv \{pc = 1,\ pc_g = 1,\ x_g \geq 0,\ x_g - 1 \geq x\}$.

In the third iteration, we refine the error condition to take both T_1 and T_2 into account. This means that we go to ℓ_{err} if, in addition to $pc = pc_g = 1$, we have both ($y_g < 0$ or $y_g - 1 < y$) and ($x_g < 0$ or $x_g - 1 < x$). Applying a safety checker we get the following counter example (at ℓ_3): $\{pc = 3,\ pc_g = 3,\ x = x_g,\ x_g \geq 0,\ y_g \leq z_g,\ y = y_g + 1,\ z_g = z\}$. It corresponds to looping at ℓ_3, and it leads to adding $T_3 \equiv \{pc = 1,\ pc_g = 1,\ z_g - y_g \geq 0,\ z_g - y_g - 1 \geq z - y\}$.

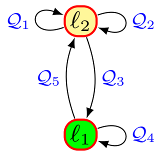
In the forth iteration, we refine the error condition to take T_3 into account similarly to what we have done for T_1 and T_2 (this time at ℓ_3). Now the safety checker succeeds in proving that ℓ_{err} is unreachable, meaning that $T_1 \cup T_2 \cup T_3$ is an invariant for the instrumented CFG (for ℓ_1 and ℓ_3), because otherwise there must be an execution that leads ℓ_{err}, and thus a DTI for the original CFG.

5.2 Wingspan of LRF-DTI

An easy observation is that LRF-DTIs subsume LLRFs. This demonstrates the point that the DTI approach breaks a complex termination proof into simple pieces.

Indeed, suppose that transition relation T has the LLRF $\langle \rho_1, \ldots, \rho_d \rangle$. Let $(s, s') \in T^+$. This means that there is a chain of transitions $(s = s_0, s_1), (s_1, s_2), \ldots, (s_{k-1}, s_k = s')$, all in T. Each such transition is ranked by one of the ρ_j (see Definition 4.3). Let m be the minimal such j. Then we have: $\Delta\rho_m(s_i, s_{i+1}) \geq 0$ for all i, $\Delta\rho_m(s_i, s_{i+1}) \geq 1$ for at least one i, and $\rho_m(s_0) \geq 0$ (since it is non-negative in at least one transition, and is decreasing throughout). Thus ρ_m ranks (as an LRF) the transition (s, s'). It follows that $\{\rho_1, \ldots, \rho_d\}$ constitutes a LRF-DTI for T.

Next, we will describe a few types of programs (i.e., of linear-



$\mathcal{Q}_1$: $\{x \geq 0, y \geq 0, z \geq 0, y' \leq y - 1, x' \leq y + 1, \}$
$\mathcal{Q}_2$: $\{x \geq 0, y \geq 0, z \geq 0, x' \leq x - 1, y' \leq x - 3\}$
$\mathcal{Q}_3$: $\{x \geq 0, y \geq 0, z \geq 0, z' \leq x, z' \leq y, x' \leq x - 1, y' \leq y - 1\}$
$\mathcal{Q}_4$: $\{x \geq 0, y \geq 0, z \geq 0, z' \leq z - 1, x' \leq x, y' \leq y\}$
$\mathcal{Q}_5$: $\{x \geq 0, y \geq 0, z \geq 0, x' \leq x, y' \leq y\}$

Figure 5.1: A CFG with δSCT transition relations.

constraint CFGs) for which LRF-DTIs provide a complete proof method for termination, and (in most of them) makes termination decidable.

5.3 δ-Size-Change-Termination

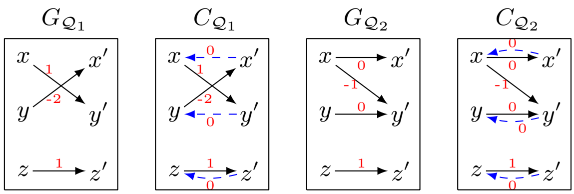
A δ-Size-Change program (or δSCT program) is a CFG where the transition relations include only *bound constraints* of the form $y' \leq x + \delta$, for state variables x, y and $\delta \in \mathbb{Z}$; we interpret such programs over the natural numbers (or assume $\mathbb{Z}$ and say that the constraints include $x \geq 0$, for every $x \in V$). Note that x and y might be the same variable, *e.g.*, $x' \leq x + \delta$, but the one on the left is primed and the other is not. The execution starts at ℓ_0 with any values for the program variables.

Example 5.2. Consider the CFG depicted in Figure 5.1. It is terminating, but it does not have an LLRF of any kind. This is because a QLRF cannot involve x and z due to $\mathcal{Q}_1$, and cannot involve y due to $\mathcal{Q}_2$. This program, however, has an LRF-DTI, as do terminating δSCT programs in general, *e.g.*, $T_x \cup T_y \cup T_z$ (using the notation of Definition 5.4).

Next we state some properties of δSCT. For this, it is useful to view a δSCT transition relation $\mathcal{Q}$ as a weighted bipartite graph $G_{\mathcal{Q}}$.

Definition 5.5. For a δSCT transition relation $\mathcal{Q}$, define the weighted bipartite graph $G_{\mathcal{Q}}$ with nodes $\{x_1, \ldots, x_n\} \cup \{x'_1, \ldots, x'_n\}$ representing the state variables before and after the transition, and arc $x \to y'$ with weight δ whenever $y' \leq x + \delta$ is in the transition constraints. This graph is called *the size-change graph* for $\mathcal{Q}$.

Definition 5.6. For a δSCT transition relation $\mathcal{Q}$, the *circular size-change graph* $C_{\mathcal{Q}}$ is obtained from $G_{\mathcal{Q}}$ by adding a zero-weight arc from



$\mathcal{Q}_1$: $\{x \geq 0, y \geq 0, z \geq 0, y' \leq x + 1, x' \leq y - 2, z' \leq z + 1\}$
$\mathcal{Q}_2$: $\{x \geq 0, y \geq 0, z \geq 0, x' \leq x, y' \leq x - 1, y' \leq y, z' \leq z + 1\}$

Figure 5.2: δSCT transition relations, and their corresponding (circular) size change graphs.

every node x' to the corresponding node x. These are called *backward arcs*.

Example 5.3. Figure 5.2 includes two δSCT transition relations, and their corresponding (circular) size change graphs. Note that $\mathcal{Q}_1$ is terminating and $\mathcal{Q}_2$ is not.

The following theorem combines observations by Codish *et al.* (2005) and Moyen (2009).

Theorem 5.2. For a δSCT transitions relation $\mathcal{Q}$, the following statements are equivalent:

1. $C_{\mathcal{Q}}$ has a negative-weighted simple cycle.

2. $\mathcal{Q}$ has an LRF of the form $\rho(\boldsymbol{x}) = \sum_{i \in S} x_i$ for some $S \subseteq \{1, \ldots, n\}$.

3. The *SLC* loop $\mathcal{Q}$ is terminating.

4. There is no solution to $\mathcal{Q} \wedge (\boldsymbol{x} \leq \boldsymbol{x'}) \wedge (\boldsymbol{x} \geq \boldsymbol{0})$.

Proof. We show that each item implies the next one, and that the last implies the first.

$1 \Rightarrow 2$: A cycle in $C_{\mathcal{Q}}$ must alternate regular (forward) arcs with backward arcs. It is a "zig-zag" cycle (see Figure 5.2). The set S of variables and S' of primed variable participating in this cycle are counterparts, *i.e.*, $x_i \in S \iff x'_i \in S'$. For every $x_i \in S$ there is a single $x'_j \in S'$ such that $x'_j \leq x_i + \delta_i$. This implies

$$\sum_{i \in S} x'_i \leq \sum_{i \in S} (x_i + \delta_i)$$

and since we assume that the total weight of the cycle, which is $\sum_{i \in S} \delta_i$, is negative, we have

$$\sum_{i \in S} x'_i < \sum_{i \in S} x_i$$

Thus we have our LRF $\rho(\boldsymbol{x}) = \sum_{i \in S} x_i$.

$2 \Rightarrow 3$: obvious, since LRFs imply termination of SLC loops.

$3 \Rightarrow 4$: Assume, to the contrary, that there is a solution $\binom{\boldsymbol{x}}{\boldsymbol{x}'}$ to $\mathcal{Q} \wedge (\boldsymbol{x} \leq \boldsymbol{x}') \wedge (\boldsymbol{x} \geq \mathbf{0})$. This solution satisfies every constraint $x'_i \leq x_j + \delta$ of $\mathcal{Q}$, and since $x_i \leq x'_i$, also the constraint $x_i \leq x_j + \delta$ is satisfied. This means that $\binom{\boldsymbol{x}}{\boldsymbol{x}} \in \mathcal{Q}$ and thus the program is not terminating, contradicting 3.

$4 \Rightarrow 1$: Suppose that $C_{\mathcal{Q}}$ has *no* negative-weight cycle. We add an auxiliary node y to $C_{\mathcal{Q}}$ and connect it with zero-weight arcs to all source nodes x_i. We can then compute the weighted distance $\delta(y, \nu)$ for each node ν. Note that: (1) these weights satisfy the constraints of $\mathcal{Q}$, *e.g.*, $x'_j \leq x_i + \delta$, because this is the triangle inequality; and (2) they satisfy $x'_i \geq x_i$, because of the backward arcs. We conclude that there is a solution to $\mathcal{Q} \wedge (\boldsymbol{x} \leq \boldsymbol{x}') \wedge (\boldsymbol{x} \geq \mathbf{0})$. □

Example 5.4. Consider again the δSCT transition relations $\mathcal{Q}_1$ and $\mathcal{Q}_2$ depicted in Figure 5.2. For $\mathcal{Q}_1$, it is easy to see that: $C_{\mathcal{Q}_1}$ includes a negative weighed cycle; it has an LRF $\rho(x, y, z) = x + y$; is terminating; and $\mathcal{Q}_1 \wedge x \leq x' \wedge y \leq y', z \leq z'$ is not satisfiable (since $x + y > x' + y'$). For $\mathcal{Q}_2$, it is easy to see that: $C_{\mathcal{Q}_2}$ does not have a negative weighed cycle; it has no LRF; is not terminating; and $\mathcal{Q}_2 \wedge x \leq x' \wedge y \leq y', z \leq z'$ is satisfied by $x = x' = 3$, $y = y' = 2$, and $z = z' = 0$.

Corollary 5.3. A δSCT CFG terminates if and only if it has an LRF-DTI. Moreover, the form of the ranking functions used is as in Theorem 5.2.

Proof. First note that for δSCT transition relations $\mathcal{Q}_1$ and $\mathcal{Q}_2$, the composition $\mathcal{Q}_1 \circ \mathcal{Q}_2$ is also a δSCT transition relation. Consider any

$((\ell, \boldsymbol{x}), (\ell, \boldsymbol{x}')) \in T^+$, and note that it corresponds to an execution trace where in each step it uses one of the transition relations of the CFG; let us say $\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_k$. The composition of these transition relations is an *SLC* loop with δSCT constraints that must be terminating, because otherwise we could construct an infinite execution for the CFG by repeating this segment. By Theorem 5.2, the composition has an LRF of a specific form (sum of variable), and there are a finite number of such LRFs. This means that the LRF-DTI induced by these LRFs is a DTI for the CFG. □

Thus, the existence of a particular kind of termination witness, namely LRF-DTI, is equivalent to the termination problem for δSCT programs. We conclude that the crux of a termination analysis of (a class of) δSCT programs is to obtain a finite description of all program cycles as *SLC* loops. If such a description is available we can check the *SLC* loops for LRFs. We indeed consider subclasses of δSCT programs, because the whole class is too difficult:

Theorem 5.4 ((Ben-Amram, 2008))**.** The termination problem for δSCT programs is undecidable.

5.4 Size-Change-Termination

A Size-Change program (or SCT program) is the special case of δSCT where the differences δ range over $\{0, -1\}$, or equivalently $(-\infty, 0]$ (the important thing is that there are no relations $y' \geq x + \delta$ with $\delta > 0$), and was developed by Lee *et al.* (2001) before δSCT. Since we compute over the natural numbers, it means that we have two types of constraints: $y' \leq x$ and $y' < x$. Note, for example, that the CFG depicted in Figure 5.1 cannot be expressed using SCT constraints without affecting its termination behaviour, since $x' \leq y + 1$ of $\mathcal{Q}_2$ cannot be exactly modelled using $x' < y$ or $x' \leq y$, and thus would be removed making the CFG non-terminating.

Example 5.5. Consider an *MLC* loop defined by the following paths:

$$\mathcal{Q}_1 = \{x' < y, y' < y\},$$
$$\mathcal{Q}_2 = \{x' < x, y' < x\},$$
$$\mathcal{Q}_3 = \{x' < y, y' \leq x\}.$$

It uses only SCT constraints, and it is terminating. It does not have an LLRF of any kind, because we cannot have a QLRF that involves x (due to $\mathcal{Q}_1$) nor y (due to $\mathcal{Q}_2$), but has an LRF-DTI $T_x \cup T_y \cup T_{x+y}$.

If we express our constraints in this form, a natural way to define the composition operation of two constraint sets $\mathcal{Q}_1$ and $\mathcal{Q}_2$, that we denote by $\mathcal{Q}_1 \bullet \mathcal{Q}_2$, is as follows:

1. $\mathcal{Q}_1 \bullet \mathcal{Q}_2$ includes $y' < x$ if and only if $\mathcal{Q}_1$ includes $z' \bowtie_1 x$ and $\mathcal{Q}_2$ includes $y' \bowtie_2 z$, for some variable z, where at least one of the relations $\bowtie_i$ is $<$;

2. $\mathcal{Q}_1 \bullet \mathcal{Q}_2$ includes $y' \leq x$ if and only if $\mathcal{Q}_1$ includes $z' \leq x$ and $\mathcal{Q}_2$ includes $y' \leq z$, for some variable z, and Case 1 does not apply.

That is, we ignore the fact that differences accumulate and express all the constraints with the vocabulary of $<, \leq$. For CFGs, the composition of two edges $(\ell_i, \mathcal{Q}_1, \ell_j)$ and $(\ell_j, \mathcal{Q}_2, \ell_k)$ is $(\ell_i, \mathcal{Q}_1 \bullet \mathcal{Q}_2, \ell_k)$. Note that the target node of the first edge must be equal to the source node of the second edge. We refer to $\bullet$ by SCT-composition, to distinguish it from the composition $\circ$.

Example 5.6. Consider the *MLC* loop of Example 5.5. We have $\mathcal{Q}_4 = \mathcal{Q}_3 \bullet \mathcal{Q}_3 = \{x' < x, y' < y\}$. All other SCT-compositions yield one of the existing paths, *e.g.*, $\mathcal{Q}_1 \bullet \mathcal{Q}_2 = \mathcal{Q}_1$ and $\mathcal{Q}_2 \bullet \mathcal{Q}_2 = \mathcal{Q}_2$.

Note that $\bullet$ is an over-approximation of $\circ$. For example, $\{x' < x\} \bullet \{x' < x\} = \{x' < x\}$ while $\{x' < x\} \circ \{x' < x\} = \{x' \leq x - 2\}$.

If we start from the set of all edges of the CFG, and compute the transitive closure using $\bullet$, it is guaranteed that the computation terminates since the set of possible constraint sets is finite. Thus T_P^+ can be symbolically over-approximated in finite time. Moreover, this over-approximation does not lose any information that may be necessary for the termination proof, *i.e.*, the CFG is non-terminating if and only if there is $(\ell, \mathcal{Q}, \ell)$ in this transitive closure such that $\mathcal{Q}$ is not well-founded. Thus we have *the closure algorithm* for SCT:

1. Compute the transitive closure, wrt. SCT-composition, of the set of edges of the CFG.

2. For every $(\ell, \mathcal{Q}, \ell)$ in the transitive closure, check that $\mathcal{Q}$ is well-founded which can be done by seeking corresponding LRFs according to Theorem 5.2.

Example 5.7. The transitive closure of the *MLC* loop of Example 5.5 adds only $\mathcal{Q}_4$ of Example 5.6. Then, $\mathcal{Q}_1$ has the LRF $\rho(x,y) = x$, $\mathcal{Q}_2$ has the LRF $\rho(x,y) = y$, $\mathcal{Q}_3$ has the LRF $\rho(x,y) = x+y$, and $\mathcal{Q}_4$ admits any of these functions as an LRF.

Using the above algorithm (in a space-economic version) we obtain:

Theorem 5.5 ((Lee *et al.*, 2001))**.** For CFGs with SCT transition relations, termination is decidable in PSPACE.

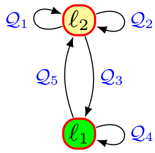
5.5 Fan-in Free δ-Size-Change-Termination

We say that a δSCT transition polyhedron $\mathcal{Q}$ *has fan-in* if there are two constraints $y' \le x + \delta_x$, $y' \le z + \delta_z$ which share the target variable y'. Equivalently, if the corresponding size change graph $G_{\mathcal{Q}}$ has a node with in-degree greater than 1. Fan-in *free* δSCT CFG is a δSCT CFG that does not have any fan-in.

Example 5.8. Consider the δSCT transition relations of Figure 5.2: $\mathcal{Q}_1$ is fan-in free and $\mathcal{Q}_2$ has a fan-in on the target variable y'.

Ben-Amram (2008) studied the class of CFGs with fan-in *free* δSCT transition relations, and showed how to form a finite over-approximation of T_P^+ that does not compromise information that is important to termination. The details are complex, so we will just give the result:

Theorem 5.6. The termination problem for CFG with fan-in δSCT transition relations is decidable in PSPACE.



$\mathcal{Q}_1$: $\{y' < y, x' < y, y \geq z, z' \geq z\}$
$\mathcal{Q}_2$: $\{x' < x, y' < x, x \geq z, z' \geq z\}$
$\mathcal{Q}_3$: $\{x' \leq x, y' \leq y, z' > z, x \geq z, y \geq z\}$
$\mathcal{Q}_4$: $\{x' \leq y, y' \leq x, z' > z, y \geq z\}$
$\mathcal{Q}_5$: $\{x' \leq x, y' \leq y, z' > z, x \geq z, y \geq z\}$

Figure 5.3: A CFG with MC transition relations.

5.6 Monotonicity Constraints

A *monotonicity constraint* (MC) transition relation is a conjunction of order constraints $x \bowtie y$ where $x, y \in \{x_1, \ldots, x_n, x'_1, \ldots, x'_n\}$, and $\bowtie \in \{>, \geq, =\}$. It extends SCT by allowing order constraints between any pair of variables, and, moreover, they are interpreted over $\mathbb{Z}$ instead of $\mathbb{N}$. Note that $x = y$ is just syntactic sugar for $x \leq y \wedge y \leq x$. So actually we have just two types of constraints.

Example 5.9. Consider the MC CFG depicted in Figure 5.3: it is terminating, and does not have an LLRF of any kind. It cannot be modelled with SCT constraints since it includes constraints like $x_2 \geq x_3$ and $x'_3 > x_3$, which are not allowed in SCT, and removing them would make it non-terminating.

Proving termination of CFGs with MC transition relations can be done, as in the case of SCT, by computing the transitive closure of the set of edges, and then check that every $(\ell, \mathcal{Q}, \ell)$ in the closure is well-founded. The closure of two *MC* transition relations $\mathcal{Q}_1$ and $\mathcal{Q}_2$ is computed in a similar way to the case of SCT, but considering all $x, y \in \{x_1, \ldots, x_n, x'_1, \ldots, x'_n\}$ and discarding results that are not satisfiable (which is one of the important differences from SCT). Formally, the composition is defined as

$$\mathcal{Q}_1 \diamond \mathcal{Q}_2 = \{x \bowtie y \mid x, y \in \boldsymbol{x} \cup \boldsymbol{x}', \mathcal{Q}_1[\boldsymbol{x}'/\boldsymbol{z}] \wedge \mathcal{Q}_2[\boldsymbol{x}/\boldsymbol{z}] \vdash x \bowtie y\}$$

where $[\boldsymbol{x}'/\boldsymbol{z}]$ (resp. $[\boldsymbol{x}/\boldsymbol{z}]$) is the renaming of $\boldsymbol{x}'$ (resp. $\boldsymbol{x}$) to $\boldsymbol{z}$. The algorithm is as follows:

1. Compute the transitive closure of the set of edges of the CFG.

2. If for every $(\ell, \mathcal{Q}, \ell)$ in the transitive closure, $\mathcal{Q}$ is well-founded then the CFG is terminating, otherwise it is not.

Like the case of SCT, the transitive closure can be computed in a finite time, and, moreover, the CFG is non-terminating if and only if there is $(\ell, \mathcal{Q}, \ell)$ in the transitive closure such that $\mathcal{Q}$ is not well-founded. Thus, to make the algorithm complete, we have to have find a complete procedure for the well-foundedness check of Point 2 above. Unlike the case of SCT, where LRFs are enough for this check, a complete procedure for MC checks that $\mathcal{Q}$ has an MΦRF of a bounded depth.

Lemma 5.7. An MC transition relation $\mathcal{Q}$ is well-founded if and only if it has an MΦRF of depth at most 5^{2n}.

Proof. It follows from results by Ben-Amram *et al.* (2021) and Ben-Amram and Genaim (2017), in turn using Bozga *et al.*, 2014. These results involve octagonal transition relations, where an octagonal polyhedron is one defined by constraints of either the form $\pm x \geq c$ or $\pm x \pm y \geq c$. Note that MC transition relations are octagonal. The first result shows that an octagonal transition relation, over the rationals, is well-founded if and only if it has an MΦRF of depth bounded by 5^{2n}. The second shows that for *SLC* loops specified by *integral* transition polyhedra, a tuple $\langle \rho_1, \ldots, \rho_d \rangle$ is an MΦRF over the rationals if and only if it is over the integers. Since a set of MC constraints is an octagonal relation and, unlike octagonal relations in general, is also an integral polyhedron, the statement of the lemma follows. □

This algorithm also implies that a terminating CFG with MC transition relations has LRF-DTI, which is defined as a disjunction of the components of the different MΦRFs.

Theorem 5.8. A CFG with MC transition relations is terminating if and only if it has an LRF-DTI.

Example 5.10. Consider the CFG depicted in Figure 5.3. Computing the transitive closure results in 16 transition relations, including the one already in CFG. The following (first column) are those important for termination, *i.e.*, source location equals to target location, and their corresponding ranking functions (second column):

$(\ell_2,\ \mathcal{Q}_1{=}\{y > y', y > x', y \geq z, z' \geq z\}\ ,\ell_2)$	$y - z$
$(\ell_2,\ \mathcal{Q}_2{=}\{x > x', x > y', x \geq z, z' \geq z\}\ ,\ell_2)$	$x - z$
$(\ell_1,\ \mathcal{Q}_4{=}\{x \geq y', y \geq x', y \geq z, z' > z\}\ ,\ell_1)$	$\langle x + y - 2z, y - z\rangle$
$(\ell_2,\ \mathcal{Q}_6{=}\{x > z, x \geq x', y > z, y \geq y', z' > z\},\ell_2)$	$x + y - z$
$(\ell_1,\ \mathcal{Q}_7{=}\{x > z, x \geq x', y \geq z, y \geq y', z' > z\},\ell_1)$	$x + y - z$
$(\ell_1,\ \mathcal{Q}_8{=}\{x \geq z, y > z, y > x', y > y', z' > z\},\ell_1)$	$y - z$
$(\ell_1,\ \mathcal{Q}_9{=}\{x > z, x > x', x > y', y \geq z, z' > z\},\ell_1)$	$x - z$
$(\ell_2,\mathcal{Q}_{10}{=}\{x > z, x \geq y', y > z, y \geq x', z' > z\},\ell_2)$	$x + y - z$

The first three already appear in the CFG, and the others were obtained using the following compositions: $\mathcal{Q}_6 = \mathcal{Q}_3 \diamond \mathcal{Q}_5$, $\mathcal{Q}_7 = \mathcal{Q}_4 \diamond \mathcal{Q}_4$, $\mathcal{Q}_8 = \mathcal{Q}_5 \diamond (\mathcal{Q}_1 \diamond \mathcal{Q}_3)$, $\mathcal{Q}_9 = \mathcal{Q}_3 \diamond (\mathcal{Q}_2 \diamond \mathcal{Q}_3)$, $\mathcal{Q}_{10} = \mathcal{Q}_3 \diamond (\mathcal{Q}_4 \diamond \mathcal{Q}_5)$. Note that all have LRFs, except $\mathcal{Q}_4$ that requires an MΦRF.

As for the case of SCT, using the closure algorithm (in a space-economic version) we obtain:

Theorem 5.9 ((Ben-Amram, 2011))**.** The termination problem for CFGs with MC transition relations is in `PSPACE`.

5.7 Gap Constraints

Gap constraints extend monotonicity constraints in two ways. First, a non-negative "gap" may be added in inequalities, *i.e.*, we have constraints of the form $x \geq y + c$ with $c \in \mathbb{N}$ (note that c cannot be negative as allowed in δSCT constraints). Here, too, x and y range over $\{x_1, \ldots, x_n\} \cup \{x'_1, \ldots, x'_n\}$. In addition, constraints of the form $x \geq a$ or $x \leq a$ are allowed, with $a, b \in \mathbb{Z}$.

Theorem 5.10 ((Bozzelli and Pinchinat, 2014))**.** The termination problem for CFG with gap constraint transition relations is in `PSPACE`.

We can prove termination of gap constraint programs using LRF-DTIs constructed similarly to the method for monotonicity constraints described above. We describe informally how this may be done. Let a_L be the lowest constant that appear in constraints of the form $x \bowtie a$

where $\bowtie \in \{\leq, \geq\}$, and a_H the highest one. Let $c_{\max}$ be the largest value among the "gaps" in the constraints.

We perform *state explosion* and replace every location in the CFG by a set of locations where every one of them is associated with a particular assignment to the variables of values in $\{-\infty, a_L, a_L + 1, \ldots, a_H + c_{max}, +\infty\}$, where $-\infty$ represents any value less than a_L and $+\infty$ represents any value larger than $a_H + c_{max}$. The edges of the original nodes are replicated among these new nodes with the addition of the constraints implied by the assignments that label the nodes. LRFs or MΦRFs are then computed for all cycles. Computing a cycle is done using the MC abstraction (*i.e.*, $x > y + c$ is treated as $x > y$) as long as variables are labelled $-\infty$ or $+\infty$.

5.8 Monotonicity Constraints and Ranking Functions

While LRF-DTIs use a simple form of ranking functions to describe each of the disjuncts in the DTI, it is not clear whether there is a closed form for a *global* ranking function, one that ranks every transition of the program. The case of MC programs is an example where we have such a closed form (Ben-Amram, 2011). This form is more complex, however, than those discussed in Section 4. Briefly, it is a *piecewise lexicographic-linear ranking function*. The form is illustrated by the following example:

$$\rho_\ell(\boldsymbol{x}) = \begin{cases} \langle 1, x_2 - x_4, 1, x_3 - x_4 \rangle & \text{if } x_2 - x_4 > x_2 - x_3 \\ \langle 1, x_2 - x_4, 0, x_3 - x_4 \rangle & \text{if } x_2 - x_4 \leq x_2 - x_3 \end{cases}$$

Note that the function is indexed by the program location it is associated with (see Section 4.1.4). In comparison with LLRFs of Section 4 we note the following differences:

1. The function is *piecewise*—each piece defined by a set of inequalities on differences of two variables.

2. The positions of the lexicographic tuple alternate between constants, and differences of pairs of variables.

Example 5.11. Consider the CFG of Example 5.3. It has the following ranking function (the same for both locations): $\rho(x, y, z) = \max(x, y) - z$ which in this case is just piecewise linear.

This result raises the following open problems:

OPEN PROBLEM 5. Is it decidable whether a general CFG (or, for simplicity, an *MLC* loop) has a piecewise LRF? A piecewise LLRFs? (Here we should allow LLRF positions to include arbitrary linear expressions in the program variables; and similarly for the conditions defining the pieces).

OPEN PROBLEM 6. Is there a closed form for global ranking functions that works for all terminating fan-in free δSCT programs?

5.9 The Power of Transition Invariants

The power of the DTI approach, even when restricted to LRF-DTI, is clear in the context of CFGs, or even *MLC* loops, because they have branching and non-determinism that allows generating traces with different properties. *SLC* loops do not have branching, and have a limited form of non-determinism that originate from the constraints specifying them. Given this, it is natural to ask the following.

OPEN PROBLEM 7. Are there terminating *SLC* loops, deterministic or non-deterministic, whose termination can be shown using LRF-DTI, but not using ranking functions as those of Section 4?

The restriction to LRF-DTI is because the ranking functions of Section 4 are restricted to linear components, moreover, we can focus on MΦRFs since they are the most powerful, among those discussed in Section 4, for *SLC* loop. In what follows we discuss partial answers to this question, and state open problems.

For integer loops, the following deterministic *SLC* loop

$$\texttt{while } (x \geq 0) \texttt{ do } x' = 10 - 2x$$

is terminating over the integers, and non-terminating over the rationals (*e.g.*, for $3\frac{1}{3}$). It has a DTI ($T_x \cup T_{10-x}$) over the integers, and does not have an MΦRF. This provides a positive answer for the above problem, for the integer case, however, we note that this loop has a piecewise LRF:

$$\rho(x) = \begin{cases} x & x > 3 \\ 10 - x & \text{otherwise} \end{cases}$$

This somehow introduces piecewise LRFs (with polyhedral conditions) into this discussion, and thus we can generalise the problem above to the following one about the relative power of these termination arguments.

OPEN PROBLEM 8. What is the relative power of piece-wise LRFs (with polyhedral conditions), LRF-DTIs, and MΦRFs, for *SLC* loops.

To understand the power of DTI for *SLC* loops, for the rational case, one might also study the need for T^+ for this class of loops. In particular, study if the requirement $T^+ \subseteq T_1 \cup \cdots \cup T_k$, where each T_i is well-founded, can be relaxed to $T \subseteq T_1 \cup \cdots \cup T_k$ for *SLC* loops. This is not true for integer *SLC* loops. For example, $\mathcal{Q} = \{x \geq 0, x' = 1 - x\} \subseteq T_x \cup T_{-x}$, but the loop is non-terminating for $x = 0$.

OPEN PROBLEM 9. For an *SLC* loop over the rationals, does $\mathcal{Q} \subseteq T_i \cup \cdots \cup T_k$, where each T_i is well-founded, *i.e.*, a terminating *SLC* loop, implies termination of $\mathcal{Q}$?

For LRF-DTIs we have the following conjuncture, which we know to be true for for $k \leq 3$.

CONJECTURE 5.11. If $\mathcal{Q} \subseteq T_{\rho_1} \cup \cdots \cup T_{\rho_k}$, then $\mathcal{Q}$ has an MΦRF.

Finally, we note that there are terminating *SLC* loops that do not have a polyhedral DTI at all. For example, the following *SLC* loop

$$\textit{while } (x \geq 1,\ y \geq 1,\ x \geq y) \textit{ do } x' = 2x,\ y' = 3y$$

which is terminating, and its termination can be shown using the techniques of Section 3.1, or using non-linear ranking functions such as $\rho(x, y) = \log_2(x) - \log_2(y)$ or $\rho(x, y) = \frac{x}{y}$.

5.10 Other Works Related to Transition Invariants

The practical application of DTI was also promoted by Podelski and Rybalchenko (2007), who proposed a technique to generate transition invariants that are *inductive*, using predicate abstraction. Two subsequent works (Heizmann *et al.*, 2010; Zuleger, 2018) explore the connections of this type of DTI termination proofs to SCT. Zuleger (2015) constructs, for *fan-out free* SCT programs, global ranking functions which are still piecewise-lexicographic, as those mentioned earlier, but are optimal in their depth (which is interesting if the ranking functions are used to estimate execution time, see our discussion of depth in Section 4.2). Chen *et al.* (2015) propose heuristics for discovering DTIs for *SLC* loops. Kroening *et al.* (2010) proposed using *compositional transition invariants*, which are transition invariants T_I that satisfy $T_I \circ T_I \subseteq T_I$. They show a heuristic for finding such DTIs that performs better, empirically, than the method of suggested by Cook *et al.* (2006). Ganty and Genaim (2013) developed conditional termination analysis based on DTIs. Their idea is to use DTIs to isolate the non-terminating part of a given transition relation.

6

Witnesses for Non-Termination

By non-termination we mean the converse of termination, namely the *existence* of an infinite computation. A *non-termination witness* is an object whose existence proves that a given program, or loop, is non-terminating. Note that, in general, we cannot resort to the easy answer "present a non-terminating path", as this is an infinite object. An algorithm that can decide the existence of a non-termination witness of a given kind can serve as a partial solution to the termination problem, and complement partial solutions that can only confirm termination (*e.g.*, ranking functions). In this section we present non-termination witnesses, in particular *recurrent sets* of different forms.

Definition 6.1. Given a transition relation $T \subseteq S \times S$, we say that a non-empty set $G \subseteq S$ is a recurrent set for T if and only if $\forall s \in G. \exists s' \in G.\ (s, s') \in T$.

A recurrent set clearly implies non-termination of T, since we can construct an infinite execution that uses only states from G, but also the inverse holds: if T is non-terminating, then the set of states that participate in (any subset of) its infinite executions is a recurrent set. Thus, recurrent sets constitute a complete criterion for non-termination.

To establish non-termination wrt. a set of initial states $S_0 \subset S$, we seek a recurrent set G such that $S_0 \cap G \neq \emptyset$. This is still a complete criterion for non-termination, wrt. a given set of initial states, because if a recurrent set G is reachable from S_0 only indirectly using an execution path $s_0, s_1, \ldots, s_k$ where $s_0 \in S_0$ and $s_k \in G$, then $G' = G \cup \{s_0, \ldots, s_k\}$ is a recurrent set too and satisfies $S_0 \cap G' \neq \emptyset$ (we could also seek a recurrent set for T_{S_0}; the restriction of T to states reachable from S_0). However, requiring $S_0 \cap G = \emptyset$ might be too restrictive in practice because, for the sake of practicality, we typically seek recurrent sets of a particular form, *e.g.*, polyhedral, and thus instead we require that G is reachable from S_0.

Organisation of this Section. In the rest of this section we will discuss non-termination analysis using polyhedral recurrent sets. Section 6.1 discusses the inference of recurrent sets for *SLC* loops; Section 6.2 discusses the notion of *Geometric Non-Termination Arguments*, and show that it is a special form of recurrent sets; Section 6.3 explains how these notions extend to non-termination of CFGs; Section 6.4 discusses the notion of unbounded executions and its relation to non-termination; and Section 6.5 discusses other approaches to non-termination.

6.1 Recurrent Sets for Single-path Linear-Constraint Loops

In this section we discuss the inference of polyhedral recurrent sets for *SLC* loops, first without any assumption on the input states and then assuming a given polyhedral set of initial states. Moreover, we first assume that variables range over the reals, and then discuss the rational and integer cases. Let us start by defining the notion of a recurrent set in this context, which is equivalent to Definition 6.1 but more adequate for inferring them automatically.

Definition 6.2 ((Gupta *et al.*, 2008)). A polyhedral set $\mathcal{G} \subseteq \mathbb{R}^n$ is recurrent set for an *SLC* loop $\mathcal{Q} \subseteq \mathbb{R}^{2n}$ if and only if:

$$\exists \boldsymbol{x} \in \mathbb{R}^n.\ \mathcal{G}(\boldsymbol{x}) \tag{6.1}$$

$$\forall \boldsymbol{x} \in \mathbb{R}^n\ \exists \boldsymbol{x}' \in \mathbb{R}^n.\ \mathcal{G}(\boldsymbol{x}) \to \mathcal{Q}(\boldsymbol{x}, \boldsymbol{x}') \wedge \mathcal{G}(\boldsymbol{x}'). \tag{6.2}$$

Condition (6.1) forces $\mathcal{G}$ to be non-empty, and Condition (6.2) forces any $\boldsymbol{x} \in \mathcal{G}$ to have a successor $\boldsymbol{x}' \in \mathcal{G}$. The domain of variables is explicitly chosen as $\mathbb{R}$. If we are interested in $\mathbb{Z}$ or $\mathbb{Q}$, we require $\boldsymbol{x}$ and $\boldsymbol{x}'$ to range over the respective domain in (6.1,6.2). This is a subtle issue in automatic inference of recurrent sets, and will be discussed later in detail.

Since $\mathcal{G}$ is polyhedral, *i.e.*, defined by a finite set of inequalities, inferring a recurrent set for $\mathcal{Q}$ can be based on the template-based approach. We start from a template recurrent set $\mathcal{G}$, where the coefficients and constants of its inequalities are parameters, and then find values for these parameters such that (6.1,6.2) hold. However, due to the quantifier alternation $\forall\exists$ in (6.2), we cannot base such inference directly on Farkas' lemma as we have done in Section 4 for LRFs and LLRFs. If we succeed to eliminate $\exists \boldsymbol{x}'$ from (6.2), then we can apply Farkas' lemma since we are left with a $\exists\forall$ formula (the $\exists$ here is over the template parameters of $\mathcal{G}$). This is clearly not possible in general, however, Gupta *et al.* (2008) show that this can be done for some cases of *SLC* loops, in particular affine *SLC* loops as in (2.7).

Let us assume that $\mathcal{Q}$ is given as $A''\binom{\boldsymbol{x}}{\boldsymbol{x}'} \leq \boldsymbol{c}$, and that $\mathcal{G}$ is a template of the form $B\boldsymbol{x} \leq \boldsymbol{b}$, where B and $\boldsymbol{b}$ include template parameters. To eliminate $\exists \boldsymbol{x}'$ of (6.2), Gupta *et al.* (2008) assume that $\mathcal{Q}$ includes (or implies) equations of the form $\boldsymbol{x}' = A\boldsymbol{x} + \boldsymbol{d}$, *i.e.*, the variables are updated deterministically. Then, we eliminate $\exists \boldsymbol{x}'$ by replacing occurrences of $\boldsymbol{x}'$ by $A\boldsymbol{x} + \boldsymbol{d}$. This leaves us with a formula of the form

$$\exists \boldsymbol{x} \in \mathbb{R}^n.\ B\boldsymbol{x} \leq \boldsymbol{b}, \tag{6.3}$$

$$\forall \boldsymbol{x} \in \mathbb{R}^n \in \mathbb{R}^n.\ B\boldsymbol{x} \leq \boldsymbol{b} \rightarrow A''\binom{\boldsymbol{x}}{A\boldsymbol{x}+\boldsymbol{d}} \leq \boldsymbol{c} \wedge B(A\boldsymbol{x} + \boldsymbol{d}) \leq \boldsymbol{b}, \tag{6.4}$$

in which both sides of the implication are linear inequalities with template parameters. Thus, we can use Farkas' lemma to translate (6.4) into a non-linear formula $\Psi_{(6.4)}$ over the template parameters and some other variables representing the Farkas' coefficients (non-linearity is due to the template parameters on the left-hand side of the implication). Solving $\Psi_{(6.4)}$ in conjunction with (6.3) we obtain values for the template parameters, in B and $\boldsymbol{b}$, for which (6.1,6.2) are satisfied, and thus $B\boldsymbol{x} \leq \boldsymbol{b}$ is a recurrent set for $\mathcal{Q}$. Note that if $\mathcal{Q}$ is directly given as a linear loop of the form

$$\textit{while } (G\boldsymbol{x} \leq \boldsymbol{g}) \textit{ do } \boldsymbol{x}' = A\boldsymbol{x} + \boldsymbol{d}$$

then $A''\binom{\boldsymbol{x}}{A\boldsymbol{x}+\boldsymbol{d}} \leq \boldsymbol{c}$ in (6.4) become $G\boldsymbol{x} \leq \boldsymbol{g}$.

Example 6.1. Consider the following *SLC* loop $\mathcal{Q}$ and a corresponding template recurrent set $\mathcal{G}$ ($b_1, \ldots, b_6$ are the parameters):

$$\mathcal{Q} = \{-x_1 + x_2 \leq -1,\ x_1' = -x_1 + x_2,\ x_2' = x_2 - 1\} \tag{6.5}$$

$$\mathcal{G} = \{b_1x_1 + b_2x_2 \leq b_3,\ b_4x_1 + b_5x_2 \leq b_6\} \tag{6.6}$$

Note that x_1 and x_2 are updated as required in (6.4). Rewriting (6.4) using this context we get:

$$\exists \boldsymbol{b} \in \mathbb{R}^6,\ \forall \boldsymbol{x} \in \mathbb{R}^2,$$

$$\boxed{\begin{array}{l} b_1x_1 + b_2x_2 \leq b_3 \wedge \\ b_4x_1 + b_5x_2 \leq b_6 \wedge \end{array}} \rightarrow \begin{array}{l} \boxed{-x_1 + x_2 \leq -1 \wedge} \\ \boxed{\begin{array}{l} -b_1x_1 + (b_1 + b_2)x_2 \leq b_3 + b_2 \wedge \\ -b_4x_1 + (b_4 + b_5)x_2 \leq b_6 + b_5 \end{array}} \end{array} \tag{6.7}$$

The left-hand side is $\mathcal{G}(\boldsymbol{x})$; the first inequality in the right-hand side is $\mathcal{Q}(\boldsymbol{x}, A\boldsymbol{x} + \boldsymbol{d})$; and the rest correspond to $\mathcal{G}(\boldsymbol{x}, A\boldsymbol{x} + \boldsymbol{d})$.

Using Farkas' lemma we can translate (6.7) into the following set of non-linear constraints

$$\left\{ \begin{array}{l} \hline \mu_1b_1 + \mu_2b_4 = -1,\ \mu_1b_2 + \mu_2b_5 = 1, \\ \mu_1b_3 + \mu_2b_6 \leq -1,\ \mu_1 \geq 0,\ \mu_2 \geq 0, \\ \hline \xi_1b_1 + \xi_2b_4 = -b_1,\ \xi_1b_2 + \xi_2b_5 = b_1 + b_2, \\ \xi_1b_3 + \xi_2b_6 \leq b_3 + b_2,\ \xi_1 \geq 0,\ \xi_2 \geq 0, \\ \hline \eta_1b_1 + \eta_2b_4 = -b_4,\ \eta_1b_2 + \eta_2b_5 = b_4 + b_5, \\ \eta_1b_3 + \eta_2b_6 \leq b_6 + b_5,\ \eta_1 \geq 0,\ \eta_2 \geq 0, \\ \hline \end{array} \right\} \tag{6.8}$$

where each block corresponds to translating, using Farkas' lemma, one constraint from the right-hand side of (6.7). Solving (6.8) together with (6.6), to require $\mathcal{G}$ to be non-empty, we get the following possible solution:

$$b_1 \mapsto 1,\ b_2 \mapsto 0,\ b_3 \mapsto 0,\ b_4 \mapsto -1,\ b_5 \mapsto 1,\ b_6 \mapsto -1,$$

which defines the recurrent set $\{x_1 \leq 0, -x_1 + x_2 \leq -1\}$.

Let us now consider the case where the domain of the variables is the integers, *i.e.*, replacing $\mathbb{R}$ by $\mathbb{Z}$ in (6.3,6.4). The use of Farkas' lemma in this case is not immediate because a loop might be non-terminating over $\mathbb{R}$ but terminating over $\mathbb{Z}$. Thus, unlike for the case of LRFs and LLRFs, relaxation of the problem from $\mathbb{Z}$ to $\mathbb{R}$ is not sound. However, such a relaxation is sound if we guarantee that: (1) $\mathcal{G}$ has a least one integer state; and (2) for every integer state in $\mathcal{G}$, there is an integer successor in $\mathcal{G}$. The first condition can be achieved by requiring (6.3) to hold over $\mathbb{Z}$, and the second is guaranteed to hold if we assume the update $\boldsymbol{x}' = A\boldsymbol{x} + \boldsymbol{d}$ has only integer coefficients and constants. Similar argument holds for the case of $\mathbb{Q}^n$.

To summarise this approach, in terms of decidability of the underlying problems, what we have described above is a complete procedure for seeking recurrent sets, matching a given template, for affine *SLC* loops over $\mathbb{R}$ (because non-linear polynomial constraints can be solved in polynomial space (Canny, 1988)). The method is not complete because solving non-linear polynomial constraints is not decidable over $\mathbb{Z}$ and its decidability over $\mathbb{Q}$ is unknown.

Next we present an alternative definition for recurrent sets, which is more restrictive than the general case, but allows using Farkas' lemma smoothly, even for nondeterministic *SLC* loops. This notion was introduced by Chen *et al.* (2014).

Definition 6.3. Let $\mathcal{Q} \subseteq \mathbb{R}^{2n}$ be an *SLC* loop and $\mathcal{B} = \texttt{proj}_{\boldsymbol{x}}(\mathcal{Q}) \subseteq \mathbb{R}^n$ be its set of enabled states. A polyhedral set $\mathcal{G} \subseteq \mathbb{R}^n$ is a *closed* recurrent set for $\mathcal{Q}$ if and only if:

$$\exists \boldsymbol{x} \in \mathbb{R}^n.\ \mathcal{G}(\boldsymbol{x}) \tag{6.9}$$

$$\forall \boldsymbol{x} \in \mathbb{R}^n.\ \mathcal{G}(\boldsymbol{x}) \rightarrow \mathcal{B}(\boldsymbol{x}) \tag{6.10}$$

$$\forall \boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^n.\ \mathcal{G}(\boldsymbol{x}) \wedge \mathcal{Q}(\boldsymbol{x}, \boldsymbol{x}') \rightarrow \mathcal{G}(\boldsymbol{x}'). \tag{6.11}$$

Note that (6.9) is required to guarantee that $\mathcal{G}$ is not empty, and (6.10) is required to guaranties that $\mathcal{G}$ is a subset of the enabled states, and thus for any $\boldsymbol{x} \in \mathcal{G}$ we can make progress.

The advantage of this definition over Definition 6.2 is that it allows using Farkas' lemma directly, however, it is more restrictive in general since it requires all the successors of $\boldsymbol{x} \in \mathcal{G}$ to be also in $\mathcal{G}$. For deterministic SLC loops, this definition is equivalent to Definition 6.2 since in such case each enabled state $\boldsymbol{x}$ has a single successor. Moreover, if a transition relation T that has a recurrent set, then there exists transition relation $T' \subseteq T$ that has a closed recurrent set (Chen *et al.*, 2014).

Example 6.2. The loop of Example 6.1 is deterministic, and thus the recurrent set we inferred there is also closed. The SLC loop $\mathcal{Q}_1 = \{x \geq 0, x' = x - y, y' \leq y\}$ is non-deterministic, and has the closed recurrent set $\mathcal{G}_1 = \{x \geq 0, y \leq 0\}$. It also has the recurrent set $\mathcal{G}_1' = \{x \geq 0, x \geq y\}$ which is not closed because $\left(\begin{smallmatrix}1\\1\end{smallmatrix}\right) \in \mathcal{G}_1'$ has a successor $\left(\begin{smallmatrix}0\\1\end{smallmatrix}\right) \notin \mathcal{G}_1'$. The loop $\mathcal{Q}_2 = \{x \geq 0, x' \leq x - y, y' \leq y\}$ is non-deterministic, and has the recurrent set $\mathcal{G}_2 = \{x \geq 0, y \leq 0\}$ but does not have a closed one.

Let us now consider the case when variables range over the rationals. Requiring the solution (*i.e.*, the coefficients in (6.11) and (6.10) together with (6.9)) to be rational is sound. This is true since if the polyhedron $\mathcal{G}$ uses only rational coefficients in its inequalities, and satisfies (6.9)–(6.11) then it is a recurrent set over the rationals. This, however, is not sound when variables range over the integers, because it is not guaranteed that every integer state $\boldsymbol{x} \in \mathcal{G}$ has an integer successor in $\mathcal{G}$ (the successor might be non-integer).

Example 6.3. The SLC loop $\mathcal{Q} = \{x \geq 2, 2x' = 3x\}$ is non-terminating over the rationals, and is terminating over the integers (because $(\frac{3}{2})^i x$ is eventually non-integer). The set $\mathcal{G} = \{x \geq 2\}$ is a recurrent over the rationals. Over the integers, both (6.9) and (6.10) are satisfied, but the integer state $x = 3$, for example, does not have an integer successor.

This problem can also appear for non-deterministic loops.

Example 6.4. Consider the following (nondeterministic) SLC loop[1] which is terminating over the integers but not over the reals (and rationals):

$$\mathcal{Q} = \left\{ \begin{array}{l} -6x - 6y - 6x' - 6y' \leq -17,\ 4x' - 3y' \leq 1, \\ 70x - 21y + 18x' + 18y' \leq 64,\ -3x' + 4y' \leq 1 \\ -63x + 28y - 24x' - 24y' \leq -55 \end{array} \right\}$$

[1]This loop was constructed by taking the convex-hull of the following transitions: $((\frac{1}{2}, \frac{1}{3}), (1,1)), ((1,1), (\frac{1}{2}, \frac{1}{3})), ((\frac{1}{3}, \frac{1}{2}), (1,1)), ((1,1), (\frac{1}{3}, \frac{1}{2}))$ and $((1,2), (1,1))$.

The only enabled integer states are $(1,1)$ and $(1,2)$, and the transitions involving these states are $((1,2),(1,1))$, $((1,1),(\frac{1}{2},\frac{1}{3}))$, $((\frac{1}{2},\frac{1}{3}),(1,1))$, $((\frac{1}{3},\frac{1}{2}),(1,1))$, and $((1,1),(\frac{1}{3},\frac{1}{2}))$. It is easy to see that these transitions can form an infinite execution over the reals (and rationals), but not over the integers. The following polyhedral set (which is the projection of $\mathcal{Q}$ on x and y)

$$\mathcal{G} = \{-6x + 6y \leq -5, 4x - 3y \leq 1, -3x + 4y \leq 1\}$$

is a closed recurrent set over the reals, however, the state $\binom{1}{1} \in \mathcal{G}$ does not have an integer successor in $\mathcal{G}$ (nor in $\mathcal{Q}$). Note that $\mathcal{G}$ is closed because it is a superset of the projection of $\mathcal{Q}$ on (x', y') which is $\{-6x + 6y \leq -5, 4x - 3y \leq 1, -9x + 4y \leq -1, x \leq 1\}$.

To solve this problem, *i.e.*, make the relaxation to the reals sound, we can add template inequalities of the form $\boldsymbol{x}' = A\boldsymbol{x} + \boldsymbol{d}$ to $\mathcal{Q}$, where A and $\boldsymbol{d}$ are parameters, and synthesise (integer) values for them together with a closed recurrent set. In addition, we have to require

$$\exists \boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^n.\ \mathcal{Q}(\boldsymbol{x}, \boldsymbol{x}') \wedge \boldsymbol{x}' = A\boldsymbol{x} + \boldsymbol{d} \tag{6.12}$$

$$\forall \boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^n.\ \mathcal{Q}(\boldsymbol{x}, \boldsymbol{x}') \wedge \boldsymbol{x}' = A\boldsymbol{x} + \boldsymbol{d} \rightarrow \mathcal{B}(\boldsymbol{x}) \tag{6.13}$$

The first guarantees that the restriction of $\mathcal{Q}$ is not empty, and the second guarantees that the update does not block any of the enabled states. Larraz *et al.* (2014) introduced this technique for analysing the non-termination of CFGs, and we will discuss it later in Section 6.3. This techniques can also be used to make the approach described in (6.3,6.4) applicable for nondeterministic *SLC* loops as well. Note that this technique is also useful for the real-number case.

To summarise this approach, in terms of decidability of the underlying problems, what we have described above is a complete procedure for seeking closed recurrent sets, of a given template, for *SLC* loops over $\mathbb{R}$ (because non-linear polynomial constraints can be solved in polynomial space (Canny, 1988)).

Inferring a recurrent set for an *SLC* loop $\mathcal{Q}$ wrt. a polyhedral set of initial state $\mathcal{S}_0$ can be done by requiring $\mathcal{S}_0(\boldsymbol{x})$ to hold as well in (6.1) and (6.9), *i.e.*, require the recurrent set to include a state from $\mathcal{S}_0$. The decidability of the resulting problems is still the same as we have described above, for both kinds of recurrent sets. We note that the requirement that $\mathcal{S}_0$ intersects the recurrent set is, in some sense, non-restrictive: if the recurrent set $\mathcal{G}$ is reachable using a finite sequence of states $s_0 \in \mathcal{S}_0, s_1, \ldots, s_k \in \mathcal{G}$, then the convex hull of $\mathcal{G}$ and $s_0, \ldots, s_k$ is also a recurrent set. So there is a recurrent set including s_0 (caveat: this recurrent set may have a more complex description than $\mathcal{G}$).

Example 6.5. Let us analyse the non-termination of the *SLC* loop $\mathcal{Q}$ of Example 6.1, wrt. to the initial set of states $\mathcal{S}_0 = \{x_1 \leq -1, x_2 = 0\}$. Solving (6.8) together with (6.6) and $\mathcal{S}_0$ fails, because the loop terminates after one iteration for these initial states. On the other hand, for $\mathcal{S}_0 = \{x_1 \leq -1, x_2 \leq -1\}$ we succeed since it intersects the recurrent set $\mathcal{G} = \{x_1 \leq 0, -x_1 + x_2 \leq -1\}$.

We finish this section with some open problems.

OPEN PROBLEMS 10. Is there an algorithm to decide the existence of a polyhedral recurrent set (Definition 6.2) for (special cases of) *SLC* loops, over $\mathbb{R}$, $\mathbb{Q}$ or $\mathbb{Z}$? Is there an algorithm that decides the existence of a recurrent set matching a given a template?

An intriguing question is whether polyhedral recurrent sets suffice for proving non-termination of *SLC* loops.

OPEN PROBLEMS 11. Does every non-terminating *SLC* loop (perhaps, of a particular form) have a polyhedral recurrence set?

6.2 Geometric Non-Termination Arguments

The concept of Geometric Non-Termination Arguments (GNTA) is due to Leike and Heizmann (2018), and is intended for proving non-termination of *SLC* loops. Although GNTAs are not formulated as recurrent sets by Leike and Heizmann (2018), we show that they directly correspond to polyhedral recurrent sets. We will also see that GNTAs have a clear algorithmic advantage over the approaches described in Section 6.1, in particular for integer loops.

Leike and Heizmann (2018) observed an infinite execution pattern, in which variables have a geometric growth, of the form

$$\boldsymbol{x}_0,\ \boldsymbol{x}_0 + \sum_{i=0}^{0} \boldsymbol{y}\lambda^i,\ \boldsymbol{x}_0 + \sum_{i=0}^{1} \boldsymbol{y}\lambda^i,\ \boldsymbol{x}_0 + \sum_{i=0}^{2} \boldsymbol{y}\lambda^i,\ \dots \tag{6.14}$$

where $\boldsymbol{y} \in \mathbb{R}^n$ is the direction in which the execution moves, and is related to the recession cone of the loop, and $\lambda > 0$ is the speed at which it is moving.

Example 6.6. Consider the *SLC* loop $\mathcal{Q} = \{x_1 + x_2 \geq 3, x_1' = 3x_1 + 1\}$, which has the following infinite execution:

$$\left(\begin{smallmatrix}2\\1\end{smallmatrix}\right), \left(\begin{smallmatrix}7\\1\end{smallmatrix}\right), \left(\begin{smallmatrix}22\\1\end{smallmatrix}\right), \left(\begin{smallmatrix}67\\1\end{smallmatrix}\right), \dots \tag{6.15}$$

It can be generated using (6.14) with $\boldsymbol{x}_0 = \left(\begin{smallmatrix}2\\1\end{smallmatrix}\right)$, $\boldsymbol{y} = \left(\begin{smallmatrix}5\\0\end{smallmatrix}\right)$, and $\lambda = 2$. Note that $\boldsymbol{y} \in \texttt{rec.cone}(\texttt{proj}_{\boldsymbol{x}}(\mathcal{Q}))$.

Leike and Heizmann (2018) generalised (6.14) to handle cases in which variables grow in different directions, and at different speeds, to the following form (it resembles pointwise sum of geometric series)

$$\boldsymbol{x}_0,\ \boldsymbol{x}_0 + \sum_{i=0}^{0} YU^i\mathbf{1},\ \boldsymbol{x}_0 + \sum_{i=0}^{1} YU^i\mathbf{1},\ \boldsymbol{x}_0 + \sum_{i=0}^{2} YU^i\mathbf{1},\ \dots \tag{6.16}$$

where for some $k > 0$, $\mathbf{1} \in \mathbb{R}^k$ is a column vector of 1's, $Y \in \mathbb{R}^{n\times k}$ is a matrix such that its columns $\boldsymbol{y}_1, \dots, \boldsymbol{y}_k$ are the directions in which the execution moves, and are related to the recession cone of $\mathcal{Q}$, and $\boldsymbol{U} \in \mathbb{R}^{k\times k}$ is a matrix

$$U = \begin{pmatrix} \lambda_1 & \mu_1 & 0 & \dots & 0 & 0 \\ 0 & \lambda_2 & \mu_2 & \dots & 0 & 0 \\ \vdots & & & \ddots & & \vdots \\ 0 & 0 & 0 & \dots & \lambda_{k-1} & \mu_{k-1} \\ 0 & 0 & 0 & \dots & 0 & \lambda_k \end{pmatrix}$$

with $\lambda_1, \dots, \lambda_k, \mu_1, \dots, \mu_{k-1} \geq 0$, representing the speed of growth.

Example 6.7. Consider the *SLC* loop $\mathcal{Q} = \{x_1 + x_2 \geq 4, x_1' = 3x_1 + x_2, x_2' = 2x_2\}$, which has the following infinite execution:

$$\left(\begin{smallmatrix}3\\1\end{smallmatrix}\right), \left(\begin{smallmatrix}10\\2\end{smallmatrix}\right), \left(\begin{smallmatrix}32\\4\end{smallmatrix}\right), \left(\begin{smallmatrix}100\\8\end{smallmatrix}\right), \dots \tag{6.17}$$

It can be generated using (6.16) with

$$\boldsymbol{x}_0 = \left(\begin{smallmatrix}3\\1\end{smallmatrix}\right), Y = \begin{pmatrix}4 & 3\\0 & 1\end{pmatrix}, \text{ and } U = \begin{pmatrix}3 & 1\\0 & 2\end{pmatrix}$$

Note that the columns of Y are in $\texttt{rec.cone}(\texttt{proj}_{\boldsymbol{x}}(\mathcal{Q}))$.

A GNTA consists of $\boldsymbol{x}_0$, Y and U that yield an infinite execution as in (6.16). The following definition states how a GNTA is extracted from $\mathcal{Q}$.

Definition 6.4 **((Leike and Heizmann, 2018)).** Let $\mathcal{Q}$ be an *SLC* loop specified by $A''\left(\begin{smallmatrix}\boldsymbol{x}\\\boldsymbol{x}'\end{smallmatrix}\right) \le \boldsymbol{c}$. A tuple $\langle \boldsymbol{x}_0, \boldsymbol{y}_1, \dots, \boldsymbol{y}_k, \lambda_1, \dots, \lambda_k, \mu_1, \dots, \mu_k\rangle$ is a *geometric non-termination argument* (GNTA) of size k for $\mathcal{Q}$ if and only if the following holds

$$\boldsymbol{x}_0, \boldsymbol{y}_1, \dots, \boldsymbol{y}_k \in \mathbb{R}^n, \lambda_1, \dots, \lambda_k, \mu_1, \dots, \mu_k \ge 0 \tag{6.18}$$

$$A''\left(\begin{smallmatrix}\boldsymbol{x}_0\\\boldsymbol{x}_0+\Sigma_i \boldsymbol{y}_i\end{smallmatrix}\right) \le \boldsymbol{c} \tag{6.19}$$

$$A''\left(\begin{smallmatrix}\boldsymbol{y}_1\\\lambda_1\boldsymbol{y}_1\end{smallmatrix}\right) \le \mathbf{0} \text{ and } A''\left(\begin{smallmatrix}\boldsymbol{y}_i\\\lambda_i\boldsymbol{y}_i+\mu_{i-1}\boldsymbol{y}_{i-1}\end{smallmatrix}\right) \le \mathbf{0} \text{ for } 1 < i \le k. \tag{6.20}$$

Note that (6.19) requires a specific transition to be in $\mathcal{Q}$, while (6.20) requires specific rays to be in the recession cone of $\mathcal{Q}$. Condition (6.18) fixes the domain of the elements of a GNTA, and is useful when seeking GNTAs over the integers as we will see later.

Theorem 6.1 **((Leike and Heizmann, 2018)).** If an *SLC* loop has a GNTA $\langle \boldsymbol{x}_0, \boldsymbol{y}_1, \dots, \boldsymbol{y}_k, \vec{\lambda}, \vec{\mu}\rangle$, then there is an infinite execution that starts at state $\boldsymbol{x}_0$.

Proof. The idea is to construct an execution of the form (6.16), and show that every pair of consecutive states is a transition in $\mathcal{Q}$, namely

$$\begin{pmatrix}\boldsymbol{x}_0 + \sum_{j=0}^{i-1} YU^j\mathbf{1}\\ \boldsymbol{x}_0 + \sum_{j=0}^{i} YU^j\mathbf{1}\end{pmatrix} \in Q \text{ for all } i \ge 0. \tag{6.21}$$

This can be done by induction. It holds for $i = 0$ due to (6.19). Assume it holds for $i = t > 0$, then for $i = t + 1$ we can rewrite (6.21) as

$$\begin{pmatrix}\boldsymbol{x}_0 + \sum_{j=0}^{t-1} YU^j\mathbf{1}\\ \boldsymbol{x}_0 + \sum_{j=0}^{t} YU^j\mathbf{1}\end{pmatrix} + \begin{pmatrix}YU^t\mathbf{1}\\ YU^{t+1}\mathbf{1}\end{pmatrix} \tag{6.22}$$

The term on the left is in $\mathcal{Q}$ by the induction hypothesis, and the one on the right is a non-negative combination of the rays defined in (6.20), and thus the sum is in $\mathcal{Q}$. Note that multiplication on the right by $\mathbf{1}$ is equivalent to adding together the columns of the multiplied matrix.

□

OBSERVATION 6.2. GNTAs induce polyhedral recurrent sets.

Proof. Consider the *SLC* loop $\mathcal{Q}' \subseteq \mathbb{R}^{2n}$ built from the points and rays in (6.19,6.20) as follows

$$\mathtt{conv}\{\binom{\boldsymbol{x}_0}{\boldsymbol{x}_0+\Sigma_i \boldsymbol{y}_i}\} + \mathtt{cone}\{\binom{\boldsymbol{y}_1}{\lambda_1 \boldsymbol{y}_1}, \binom{\boldsymbol{y}_2}{\lambda_2 \boldsymbol{y}_2+\mu_1 \boldsymbol{y}_1}, \ldots, \binom{\boldsymbol{y}_k}{\lambda_k \boldsymbol{y}_k+\mu_{k-1} \boldsymbol{y}_{k-1}}\}$$

and note that $\mathcal{Q}' \subseteq \mathcal{Q}$. Clearly $\mathtt{proj}_{\boldsymbol{x}'}(\mathcal{Q}') \subseteq \mathtt{proj}_{\boldsymbol{x}}(\mathcal{Q}')$, which means that $\mathtt{proj}_{\boldsymbol{x}}(\mathcal{Q}')$ is a closed recurrent set for $\mathcal{Q}'$ and thus a recurrent set for $\mathcal{Q}$. □

A complete algorithm for finding a GNTA of size k, in practice, amounts to solving the constraints (6.18)–(6.20); this is a system of quadratic equations, and can be solved in polynomial space (Canny, 1988). Note that bounding the size of the GNTA to k is critical. In general, we do not know a bound on the size of the GNTA that a loop might have. So in practice we have to settle for an incomplete solution and arbitrarily set a bound. However, Leike and Heizmann (2018) also identified special cases for which GNTA is a complete non-termination criterion and such bound exists.

Theorem 6.3 ((Leike and Heizmann, 2018)). If an affine *SLC* loop *while* $(B\boldsymbol{x} \leq \boldsymbol{b})$ *do* $\boldsymbol{x}' = A\boldsymbol{x} + \boldsymbol{c}$, with n variables, is non-terminating, and A has only non-negative real eigenvalues, then there is a GNTA for the loop, of size at most n.

In the discussion above we have considered the case in which variables range over the reals, however, the case in which variables range over the integers (resp. rationals) is similar: we need only to require $\boldsymbol{x}_0, \boldsymbol{y}_i, \lambda_i$, and μ_i in (6.18)) to be integer (resp. rational). This is a clear advantage of the GNTA approach over those we discussed in Section 6.1.

Theorem 6.4. A GNTA where all components are integers (resp. rationals), implies that the corresponding loop has an infinite computation over the integers (resp. rationals).

To handle non-termination wrt. a polyhedral set $\mathcal{S}_0$ of initial states, we only need is to require $\mathcal{S}_0(\boldsymbol{x}_0)$ to hold in Definition 6.4.

OPEN PROBLEMS 12.

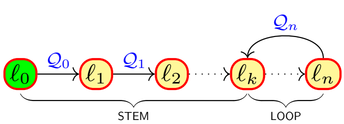
- Is there a more efficient algorithm for finding a GNTA, or deciding its existence?
- Is there a (terminating) algorithm that does not need to be provided with the size of the GNTA?
- Do GNTAs suffice for a larger class of loops?

6.3 Non-Termination of Control-Flow Graphs

In this section we turn our attention to proving non-termination of CFGs. We overview several techniques that are based on different kinds of recurrent sets to detect non-terminating loops, and also different approaches to prove that the loop is actually reachable.

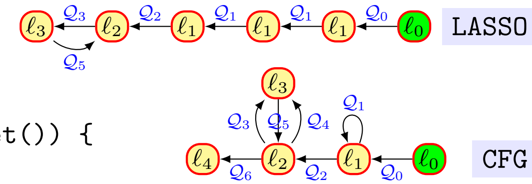
6.3.1 Lasso Loops Techniques

The technique of Gupta *et al.* (2008) is based on enumerating *lasso* loops, which are common in termination and non-termination analysis, from the CFG and then try to prove that they are non-terminating. The work of Velroyen and Rümmer (2008) is based on similar ideas—Gupta *et al.* (2008) mention that it was developed independently at the same time. A *lasso* loop can be viewed as a CFG of the form



and it is typically extracted from the original CFG, in this case, by starting at the initial location ℓ_0 and following some path. The nodes

```
assume(x >= 0 &&
       i >= 1 &&
       y >= 1);
while (i>=0 && nondet()) {
  y=y-1;
  i=i-1;
}
while (x >= 0) {
  if (nondet()) i=i+1;
  x = x-y-1;
}
```



$\mathcal{Q}_0$: $\{x \geq 0, i \geq 1, y \geq 1, x' = x, i' = i, y' = y\}$
$\mathcal{Q}_1$: $\{i \geq 0, x' = x, i' = i - 1, y' = y - 1\}$
$\mathcal{Q}_2$: $\{x' = x, i' = i, y' = y\}$
$\mathcal{Q}_3$: $\{x \geq 0, x' = x, i' = i, y' = y\}$
$\mathcal{Q}_4$: $\{x \geq 0, x' = x, i' = i + 1, y' = y\}$
$\mathcal{Q}_5$: $\{x \geq 0, x' = x - y - 1, i' = i, y' = y\}$
$\mathcal{Q}_6$: $\{x \leq -1, x' = x, i' = i, y' = y\}$

Figure 6.1: A program, its corresponding CFG, and a corresponding lasso loop.

$\ell_0, \cdots, \ell_n$ are not necessarily different (in the original CFG), which allows the STEM and the loop to include unrolling of loops of the original CFG. Clearly, non-termination of a lasso loop implies non-termination of the original CFG.

A lasso loop is basically an *SLC* loop with a polyhedral set of initial states: $\mathcal{S} = \mathcal{Q}_0(\boldsymbol{x}_0, \boldsymbol{x}_1) \wedge \mathcal{Q}_1(\boldsymbol{x}_1, \boldsymbol{x}_2) \wedge \cdots \wedge \mathcal{Q}_{k-1}(\boldsymbol{x}_{k-1}, \boldsymbol{x}_k)$ can be projected onto $\boldsymbol{x}_k$ to obtain a polyhedral set of initial of states, and $\mathcal{P} = \mathcal{Q}_k(\boldsymbol{x}_k, \boldsymbol{x}_{k+1}) \wedge \cdots \wedge \mathcal{Q}_n(\boldsymbol{x}_n, \boldsymbol{x}_{n+1})$ can be projected onto $(\boldsymbol{x}_k, \boldsymbol{x}_{n+1})$ to obtain an *SLC* loop. Thus, the techniques of sections 6.1 and 6.2 can be (indirectly) used for proving non-termination of lasso loops. It is also straightforward, and indeed done in practice, to adapt those techniques to work directly on $\mathcal{P}$ and $\mathcal{S}$ (variables other than ($\boldsymbol{x}_k$ and $\boldsymbol{x}_{n+1}$) are considered existential when using Farkas' lemma).

Example 6.8. Consider the program and the corresponding CFG depicted in Figure 6.1. The first loop is terminating, the second loop does not terminate when y is negative. The initial value of y is at least 1, and the first loop decreases its value at most $i + 1$ times. When the second loop is reached, the value of y can be negative if the first loop is executed at least two iterations (for initial value $y = 1$). To expose this behaviour, Gupta *et al.* (2008) unfold the first loop twice and obtain the lasso loop shown in Figure 6.1 as well. Now we can prove the non-termination of this lasso loop, because it is like proving non-termination of the *SLC* loop $\mathcal{Q} = \{x \geq 0, x' = x' - y - 1, y' = y, i' = i\}$ with the set of initial states $\mathcal{S}_0 = \{y \geq -1, x \geq 0, i \geq -1\}$. Note we can produce several terminating lasso loops before producing the desired one.

6.3.2 Quasi -Invariants Techniques

The approach of Larraz *et al.* (2014) is based on finding a strongly connected sub-graph (SCSG) that is non-terminating when considered separately, and then proving that it is reachable from the initial location. This is done by enumerating all SCSG until finding the desired one. The main advantage over the lasso based approach is that the number of SCSGs is finite, while the number of lassos is infinite. One can also employ various heuristics for reachability analysis (Beyer and Keremoglu, 2011; Asadi *et al.*, 2021).

Proving termination of a given SCSG is based on a concept that Larraz *et al.* (2014) call *quasi*-invariants. These are properties that once hold at the locations of the SCSG, they will continue to hold. This notion can be seen as a generalisation of closed recurrent sets to involve several locations. In what follows, we will present the basic ideas of this approach, but will not strictly follow the definitions as presented by Larraz *et al.* (2014), since much of the details are added to obtain a practical implementation. We also note that Larraz *et al.* (2014) assume that CFGs satisfy some properties, that we mostly skip, which can be easily achieved by simple program transformations, and are useful for practical reasons. The property that is important to our presentation is that we can always make a progress, except from the terminal locations, *i.e.*, there are no blocking states.

Let P' be an SCSG of a CFG P, and let $\ell_{i_1}, \ldots, \ell_{i_k}$ be its locations. We say that $\mathcal{I}_{i_1}, \ldots, \mathcal{I}_{i_1} \subseteq \mathbb{R}^n$ is a (polyhedral) quasi-invariant for P' if the following are satisfied:

$$\exists \boldsymbol{x}, \boldsymbol{x}'.\, \mathcal{I}_{\ell_i}(\boldsymbol{x}) \wedge \mathcal{Q}(\boldsymbol{x}, \boldsymbol{x}') \text{ for all } (\ell_i, \mathcal{Q}, \ell_j) \in P' \tag{6.23}$$

$$\forall \boldsymbol{x}, \boldsymbol{x}'.\, \mathcal{I}_{\ell_i}(\boldsymbol{x}) \wedge \mathcal{Q}(\boldsymbol{x}, \boldsymbol{x}') \rightarrow \mathcal{I}_{\ell_j}(\boldsymbol{x}') \text{ for all } (\ell_i, \mathcal{Q}, \ell_j) \in P' \tag{6.24}$$

$$\forall \boldsymbol{x}, \boldsymbol{x}'.\, \mathcal{I}_{\ell_i}(\boldsymbol{x}) \wedge \mathcal{Q}(\boldsymbol{x}, \boldsymbol{x}') \rightarrow \mathit{false} \text{ for all } \ell_i \in P', (\ell_i, \mathcal{Q}, \ell_j) \notin P' \tag{6.25}$$

Lets us explain the meaning of these formulas: (6.23) guarantees that all components of the quasi-invariant are not empty, and is similar to (6.9) of closed recurrent sets; (6.24) guarantees that when progressing from a state within the quasi-invariant we remain within the quasi-invariant, and is similar to (6.11) of closed recurrent sets; and (6.24) states that executions within the quasi-invariant cannot escape from the SCSG, which is similar to (6.10) of closed recurrent sets. Clearly, P' does not terminate when starting the execution at location $\ell_i \in P'$ with $\boldsymbol{x} \in \mathcal{I}_{\ell_i}$. Moreover, if the state $(\ell_i, \boldsymbol{x})$ is reachable in P, then P is non-terminating.

Example 6.9. For the CFG of Figure 6.1, Larraz *et al.* (2014) consider the SCSG of nodes l_2 and l_3 and all edges that connect them. Then they infer $\mathcal{I}_{l_2} = \mathcal{I}_{l_3} = \{x \geq 0, y \leq -1\}$, and then separately prove that l_2 is reachable with some $\boldsymbol{x} \in \mathcal{I}_{l_2}$.

We have seen in Section 6.1 that non-determinism might prevent *SLC* loops to have a closed recurrent set. This is also true for quasi-invariants. The solution that was suggested in the context of *SLC* loops is to try make $\mathcal{Q}$ deterministic, by adding more constraints, while seeking a closed recurrent set. This solution was actually proposed by Larraz *et al.* (2014) for inferring quasi-invariants. The most common way to do this is by adding parametric constraints of the form $\boldsymbol{x}' = A\boldsymbol{x} + \boldsymbol{c}$, which are also useful for handling the integer case when forcing A and $\boldsymbol{c}$ to be integer (since any integer enabled state will have an integer successor).

Example 6.10. Consider again the CFG of Figure 6.1, and assume $\mathcal{Q}_3$ has $x' \leq x - y - 1$ instead of $x' = x - y - 1$. With this change, it is not possible to infer a quasi-invariant satisfying (6.23)-(6.25) (there is no closed recurrent set). Larraz *et al.* (2014) automatically add $x' = x - y - 1$ to $\mathcal{Q}_3$, which makes it possible to infer the quasi-invariant of Example 6.9.

6.3.3 Loop Acceleration Techniques

Frohn and Giesl (2019) use *loop acceleration* to prove non-termination of integer CFGs. The core idea of this approach is that, instead of unfolding a loop a finite number of times to generate a candidate lasso, we can accelerate the loop which leaves the number of necessary unfoldings as a parameter, k, within the accelerated loop's term. A constraint solver can later determine the value of k needed to prove the reachability of a non-terminating simple loop (the loops they consider are single-path, like affine SLC loops, but the guard can have polynomial inequalities and the update is of the form $x'_i = p(\boldsymbol{x})$ where p is a polynomial). Proving non-termination of a simple loop, however, still relies on the concept of recurrent sets even if inferring such sets is done slightly in a different way. Note that their approach extends beyond linear-constraint CFGs because it allows using polynomial expression in the guard and the update (even when analysing linear-constraint CFGs, it might generate transition relations with non-linear constraints).

Next we briefly describe the basics of the algorithm of Frohn and Giesl (2019), for more precise details the reader is refereed to Frohn and Giesl (2019). The algorithm is based on iteratively repeating a series of operations until some conditions are satisfied:

1. *Prove Non-Termination of Simple Loops*: The algorithm attempts to prove non-termination for each simple loop $(\ell_i, \mathcal{Q}, \ell_i)$. If successful, the loop's edge is replaced by $(\ell_i, \mathcal{Q}, \ell_\omega)$, with ℓ_ω indicating non-termination. Non-termination is proven by a variety of techniques, one of them checks if the guard is a recurrent set (or "simple invariant" in their terms). While a guard may not initially be a recurrent set, a later step strengthens it with additional constraints to achieve this goal. In principle, any technique for proving simple loop non-termination can be used here, as long as the guard is strengthened with conditions that ensure non-termination.

2. *Accelerate Simple Loops*: If certain conditions are met by a simple loop $(\ell_i, \mathcal{Q}, \ell_i)$, it is replaced by its accelerated equivalent. This is done by adding an edge $(\ell_j, \mathcal{Q}' \circ \mathcal{Q}_a, \ell_i)$, for every incoming edge $(\ell_j, \mathcal{Q}', \ell_i)$ with $\ell_j \neq \ell_i$, where $\mathcal{Q}_a$ is the result of the acceleration.

A single transition using these new edges represents the execution of $k > 0$ iterations of the original loop, where k is a new variable in $\mathcal{Q}_a$ that is existentially quantified (Alternatively, we could add k as a program variable, in which case its value would be automatically chosen since it is not assigned). The conditions that must be met for acceleration ensure that if the loop guard holds after k applications of the update, then it also holds for all previous applications. While these conditions may not be initially satisfied, a later step in the algorithm strengthens the guard with additional constraints to make this possible.

3. *Strengthen Guards of Simple Loops*: Special kind of invariants (different from the standard notion of invariants) are added to the guards of simple loops. The purpose is to make acceleration or non-termination proofs possible for these loops.

4. *Chaining*: Consecutive edges, such as $(\ell_i, \mathcal{Q}_1, \ell_j)$ and $(\ell_j, \mathcal{Q}_2, \ell_k)$, are replaced by a single, chained edge $(\ell_i, \mathcal{Q}_1 \circ \mathcal{Q}_2, \ell_k)$. Chaining has multiple purposes, including simplifying complex loops into simple ones.

The process concludes when the CFG is reduced to a set of edges all originating from the initial node ℓ_0, or when no progress in made. Then, if an edge $(\ell_0, \mathcal{Q}, \ell_\omega)$ exists and $\mathcal{Q}$ is satisfiable, the CFG is proven to be non-terminating. Note that while we use the notation $\mathcal{Q}$ for transition relations, in practice, these can include polynomial constraints due to acceleration.

Let us demonstrate some steps of this algorithm on the CFG in Figure 6.1.

Example 6.11. The algorithm starts by trying to prove non-termination of the simple loop $(\ell_1, \mathcal{Q}_1, \ell_1)$ and fails. Then it tries to accelerate it and succeed with $\mathcal{Q}'_1 = \{i' = i - k, x' = x, y' = y - k, i - k + 1 \geq 0, k \geq 1\}$. Note that the acceleration in this case resulted in linear expressions, but it might be polynomial as well. To reflect this acceleration in the CFG, we remove the original edge and add a new edge $(\ell_0, \mathcal{Q}_0 \circ \mathcal{Q}'_1, \ell_1)$. When this edge is taken with $k = n$ it simulates n iterations of the original loop. Note that if we take the edge $(\ell_0, \mathcal{Q}_0, \ell_1)$ then we are not executing the loop, *e.g.*, when the guard is not satisfied right from the beginning.

There are no more simple loops, so the algorithm applies chaining which converts the complex loop at ℓ_2 into an *MLC* loop with two paths (simple loops): $(\ell_2, \mathcal{Q}'_3, \ell_2)$ and $(\ell_2, \mathcal{Q}'_4, \ell_2)$ where $\mathcal{Q}'_3 = \mathcal{Q}_3 \circ \mathcal{Q}_5$ and $\mathcal{Q}'_4 = \mathcal{Q}_4 \circ \mathcal{Q}_5$. In addition, it reduces the paths from ℓ_0 to ℓ_2 by connecting ℓ_0 to ℓ_2, *i.e.*, it generates $(\ell_0, \mathcal{Q}_0 \circ \mathcal{Q}_2, \ell_2)$ and $(\ell_0, \mathcal{Q}_0 \circ \mathcal{Q}'_1 \circ \mathcal{Q}_2, \ell_2)$.

In the next iteration, it attempts to prove non-termination of these loops but fails because their guards are not recurrent sets. Additionally, the loops cannot be accelerated. The process then moves on to strengthen the guards with the constraint $y \leq -1$ (let us assume it is added to $\mathcal{Q}'_3$ and $\mathcal{Q}'_4$). In the subsequent iteration, this allows the algorithm to successfully prove non-termination for both loops, as their strengthened guards are now recurrent sets. As a result, it replaces the corresponding edges with $(\ell_2, \mathcal{Q}'_3, \ell_\omega)$ and $(\ell_2, \mathcal{Q}'_4, \ell_\omega)$.

Chaining now creates, among others, the edge $(\ell_0, \mathcal{Q}_0 \circ \mathcal{Q}'_1 \circ \mathcal{Q}_2 \circ \mathcal{Q}'_3, \ell_\omega)$ whose transition relation is satisfiable for any $k \geq 2$, *i.e.*, execution the first loop at least two iterations, and thus the CFG does not terminates.

In a subsequent work, Frohn and Giesl (2023) generalised this approach to allow for the acceleration of more complex loops, such as those with disjunctions in their transition relations. The details are complex, so we refer the reader to Frohn and Giesl (2023) for more details.

6.3.4 Safety Prover Techniques

Chen *et al.* (2014) present a method for proving non-termination of a CFG by reducing the problem to a series of safety-proving tasks. The approach iteratively refines an under-approximation of the original program using counterexamples from a safety prover. The "never terminates" property is encoded as a safety violation, and this refinement process ultimately produces an under-approximation of the CFG, that also induces a closed recurrent set. Note that under-approximations, in this context, means restricting the input values as well as the values of non-deterministic choices, and select an execution a path from the initial location to the loop under consideration.

The algorithm by Chen *et al.* (2014) is formalised on a slightly different (though equivalent) notion of CFGs. For the sake of simplifying the presentation, will explain the basic idea using C-like programs, like the one in Figure 6.1. For this explanation, we slightly modify the meaning of the instruction `nondet()`. We assume it is of the form `nondet` `(`ψ`)`, where ψ is a boolean condition that involves a variable r that refers to the value returned by the function. For example, the `nondet(`$r < 0$`)` would produce a negative number. The original instruction `nondet()` is syntactic sugar for `nondet(`*true*`)`.

The algorithm by Chen *et al.* (2014) is designed to prove non-termination for a given loop within a given program. To do this, it first *instruments* the program with two instructions: an `assume(`*false*`)` statement immediately after the loop's exit to simulate an error state, and an `assume(`*true*`)` statement at the program's beginning to restrict the set of input values. The core of the approach is to show that the `assume(`*false*`)` statement is unreachable. If this can be proven, the loop is guaranteed to be non-terminating (assuming there are no blocking states). However, proving this for all possible inputs is unlikely, as a loop typically terminates for some inputs but not for others. The algorithm therefore focuses on finding a specific subset of inputs and non-deterministic choices for which non-termination holds.

The process works as follows: (1) The instrumented program is passed to a safety prover; (2) If the prover proves that `assume(`*false*`)` is unreachable, the algorithm succeeds (up to a post-processing step that we discuss below); otherwise (3) The prover returns a counterexample, which is then used to strengthen the `assume` instruction that restricts the input and the choices of `nondet(.)`. This strengthening eliminates the counterexample, and the process is repeated.

At the end of this process, we remain with a restriction of the original program. We then need to prove that the loop is reachable from the initial location, which is done by inserting `assume(`*false*`)` just before the loop and passing it to a safety prover, if it returns a counterexample it means that the loop is reachable, and this counterexample is used as a stem for the loop. Finally, we have to prove that the program that consists of the stem and the loop is non-blocking, *i.e.*, that whenever `nondet(`ψ`)` is reached it is possible to pick a value that satisfies ψ and does not block the execution. If we succeed then non-termination is proven. Moreover, if we consider the transition relation induced by the restricted program, then it has a closed recurrent set since all execution are non-terminating.

Example 6.12. Let us see how to prove non-termination for the second loop in the program of Figure 6.1. We first instrument the program by adding the instruction `assume(`*false*`)` immediately after the loop's exit. We do not need to add a separate `assume(`*true*`)` instruction at the beginning, as we will use the existing one to further restrict the input. When passing the instrumented program to a safety prover, it returns the following counterexample:

```
nondet()<=0; x>=0; x=x-y-1; x<0
```

To eliminate this trace, we can strengthen `nondet()` to `nondet(`$r \geq 1$`)`. In the next iteration, we get the following counterexample:

```
i>=0 && nondet(r ≥ 1)>=0; i=i-1; y=y-1;
i>=0 && nondet(r ≥ 1)>=0; i=i-1; y=y-1;
i<0; x>=0; x=x-y-1; x<0
```

To eliminate this trace, we could add `y<=1` to the `assume` instruction at the beginning. Now the safety prover proves that `assume(`*false*`)` is unreachable, because at the beginning `y` is always 1, `x` at least 0, and `i` at least 1. Thus, we reach the second loop with `y` at most -1 and the second loop does not terminate. Next we have to prove that the second loop is reachable. This is done by adding `assume(`*false*`)` before the second loop, and passing it to a safety prover. The prover returns the following counterexample, confirming the loop's reachability via this trace:

```
i>=0 && nondet(r ≥ 1)>=0; i=i-1; y=y-1;
i>=0 && nondet(r ≥ 1)>=0; i=i-1; y=y-1; i<0
```

The restricted program now consists of this trace as a stem leading to the second loop. This program represents a valid restriction of the original one. Finally, it is easy to check that `nondet(`$r \geq 1$`)` does not block any execution. Thus we have proven non-termination.

6.4 Non-terminating vs. Unbounded States

We say that a transition relation T is unbounded in a state $\boldsymbol{x} \in R^n$, with $R \in \{\mathbb{R}, \mathbb{Q}, \mathbb{Z}\}$, if it is possible to make executions of arbitrary length starting from $\boldsymbol{x}$. We say that T is unbounded if it is unbounded in some state.

Example 6.13. Consider the *MLC* loop (2.6). For any input state (x_1, x_2) with $x_1 = 0$, we can take the first path to reset x_2 to $n \in \mathbb{N}$, and then use the second path to make a terminating execution of length n (in total $n + 1$). Thus, this loop is unbounded in any such input state, despite being terminating.

It seems clear that the situation in the example is due to non-determinism. It is easy to see that a deterministic loop is bounded if and only if it is terminating. For *SLC* loops we have an intriguing open problem.

OPEN PROBLEM 13. Is there a terminating, yet unbounded, *SLC* loop?

6.5 Other Approaches to Non-Termination

Brockschmidt *et al.* (2011) present an approach for detecting non-termination in Java Bytecode programs using termination graphs, which are finite representations of all program executions.

Bozga *et al.* (2014) present a complete method for inferring non-termination preconditions for octagonal *SLC* loops and for affine *SLC* loops whose update matrix generates a finite monoid.

Cook *et al.* (2014) investigate the conditions under which abstractions can be used to prove non-termination. Specifically, they explore when a non-terminating abstract transition relation, T^{α} (an over-approximation of a concrete relation T), guarantees that the concrete relation T is also non-terminating. They introduce a class of abstractions, that they call *live*, for which closed recurrent sets are preserved. This means that if the abstract relation T^{α} has a closed recurrent set, then the concrete relation T is guaranteed to have one as well. This finding simplifies the search for a non-termination proof, as one can seek a closed recurrent set for the abstract relation T^α, which is typically easier to analyse. Surprisingly, many of the linear-constraint abstractions used in termination analysis fall into this category, as intuitively, the only requirement is: if f is a final concrete state, and it is in the concrete states described by an abstract one g, then g is also a final abstract state. The authors demonstrate how these abstractions can be applied to analyse programs with non-linear arithmetic and heap manipulation.

Le *et al.* (2015) propose a unified, modular framework that analyses and proves both termination and non-termination simultaneously. The core of this method involves using second-order termination constraints and accumulating a set of relational assumptions on them via a Hoare-style verification.

Bakhirkin *et al.* (2015) present a method for detecting, using a purely forward abstract interpretation, non-terminating loops in imperative programs. The analysis searches for a recurrent set by building and analysing a graph of abstract states. In a subsequent work, Bakhirkin and Piterman (2016) present an abstract interpretation-based analysis for finding recurrent sets, which combines an approximate backward analysis to identify a candidate recurrent set with an over-approximate forward analysis to check and refine it.

Ben-Amram *et al.* (2019) present a method for inferring *monotonic* recurrent sets for rational *SLC* loops, as part of an algorithm that seeks MΦRFs.

A method for computing a subset of the non-terminating initial states for affine *SLC* loops over the reals was presented by Li (2017). For homogeneous linear loops over the reals with only two program variables, Dai and Xia (2012) provided a complete algorithm to compute the full set of non-terminating initial states.

Leike and Heizmann (2018) proved that if an *SLC* loop over $\mathbb{R}$ has a non-terminating execution in which each state $\boldsymbol{x}_i$ satisfies $\|\boldsymbol{x}_i\| \leq c$, for some norm $\|\cdot\|$ and $c \in \mathbb{R}$, then it has a fixpoint transition $\binom{\boldsymbol{x}}{\boldsymbol{x}}$.

7

Conclusions

Termination analysis has received considerable attention in recent decades, and today several powerful tools exist for the automatic termination analysis of different programming languages and computational models. This practical advancement would not have been possible without corresponding theoretical progress, which aims to explore the limits of proving termination and to provide algorithms for specific proof techniques—*e.g.*, ranking functions—along with corresponding complexity classifications for the underlying problems.

In this survey we provided a comprehensive overview of the state-of-the-art in *termination and non-termination analysis of linear-constraint programs*, a field that has seen significant progress over the last three to four decades and whose results are intensively used in practice. At the core of this research is a trade-off between the expressive power of a technique, *i.e.*, the class of programs it can handle, and the computational complexity of the associated decision problems. The survey systematically explored various research directions, from decidability results for specific program classes to a wide range of termination and non-termination witnesses. Despite the significant volume of work in this field, many challenging problems remain open, some of which we stated explicitly in the body of this survey. The answers to these problems will not only advance the theoretical understanding of program termination but may also impact the development of more powerful and automated termination analysis tools.

Our discussion began with the fundamental problem of deciding termination for different classes of linear-constraint programs, including *SLC* and *MLC* loops. We presented a uniform framework for affine *SLC* loops, showing that termination is decidable for variables over the reals, rationals, and integers, a problem that had proven to be a long-standing challenge. We also highlighted key undecidability results for more general classes, such as *MLC* loops, which underscore the inherent difficulty of the problem in its most general form. There are still several major open problems in this direction: (1) The decidability of termination for general *SLC* loops, whether over real, rational, or integer domains, remains an important open question. (2) The decidability of termination for *SLC* loops wrt. a given set of initial states is also an unsolved problem even for affine *SLC* loop. This latter question is closely related to the well-known, and long-standing, *Positivity Problem* for linear recurrence sequences. A good starting point for tackling the general termination problem for *SLC* loops would be to first address simpler sub-problems. This could involve focusing on deterministic loops that are not necessarily affine or on loops that allow a small, controlled degree of non-determinism.

A major part of this survey was dedicated to ranking functions, a classic and powerful method for proving termination. We covered a spectrum of ranking function types, from simple LRFs to more expressive LLRFs and MΦRFs. For each type, we examined the algorithmic and complexity aspects of their synthesis for the different kinds of programs we consider, distinguishing between rational and integer domains. There are still several major open problems in this direction: (1) Unlike other kinds of LLRFs that we considered, there are no decidability results or complete algorithms for MΦRFs without a given bound on the depth, not even for affine *SLC* loops. (2) The problem of synthesising ranking functions wrt. a given set of initial states has not received much attention, possibly due to its inherent difficulty, apart from partial solutions based on inductive invariants. A good starting point for tackling these problems is by considering simpler sub-problems, such as affine *SLC* loops or even those where the update matrix is diagonalisable.

We also explored the concept of disjunctive well-founded transition invariants, which offers an alternative to ranking functions for proving termination. This approach, which is based on Ramsey's theorem, is particularly effective for programs with complex control flow where a single ranking function, within the classes we consider, might not exist. We showed that several well-known termination analysis methods, such as size-change termination and monotonicity constraints, can be understood as applications of the DTI principle. We provided decidability results for these classes. Note that these classes have been originally studied from different viewpoints, but in this survey we have shown how they all fall under the DTI approach. The link between DTIs and ranking functions was also discussed. A major open problem in this area is to characterise classes of programs for which DTIs are not more powerful than LLRFs. A good starting point for tackling this problem is to consider *SLC* loops, where non-determinism does not arise from branching. One could begin with special cases, such as affine or deterministic *SLC* loops, before moving to the general case.

We have also discussed witnesses for non-termination, such as polyhedral recurrent sets and geometric non-termination arguments. These witnesses provide a concrete object that proves a program will not halt, complementing the techniques for proving termination. We reviewed algorithms for their synthesis and highlighted the challenges, particularly when dealing with non-deterministic or integer-based programs. Unlike other topics in this survey, decidability results and complete algorithms for non-termination proofs are very limited: for *SLC* loops one has to provide a limit on the size of the GNTA, and for affine *SLC* loops one has to provide a template recurrent sets. These also work only over the reals. Addressing these problems is a major challenge, and one could start by characterising subclasses of *SLC* loops for which polyhedral recurrent sets or geometric non-termination arguments are sufficient.

References

Akshay, S., N. Balaji, and N. Vyas. (2017). “Complexity of Restricted Variants of Skolem and Related Problems”. In: *Proceedings of the 42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017*. Ed. by K. G. Larsen, H. L. Bodlaender, and J. Raskin. Vol. 83. *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. 78:1–78:14. DOI: 10.4230/LIPICS.MFCS.2017.78.

Albert, E., P. Arenas, M. Codish, S. Genaim, G. Puebla, and D. Zanardini. (2008). “Termination Analysis of Java Bytecode”. In: *Proceedings of the 10th International Conference on Formal Methods for Open Object-Based Distributed Systems, FMOODS 2008*. Ed. by G. Barthe and F. S. de Boer. Vol. 5051. *Lecture Notes in Computer Science*. Springer. 2–18. DOI: 10.1007/978-3-540-68863-1_2.

Alias, C., A. Darte, P. Feautrier, and L. Gonnord. (2010). “Multi-dimensional Rankings, Program Termination, and Complexity Bounds of Flowchart Programs”. In: *Proceedings of the 17th International Symposium on Static Analysis, SAS 2010*. Ed. by R. Cousot and M. Martel. Vol. 6337. *Lecture Notes in Computer Science*. Springer. 117–133. DOI: 10.1007/978-3-642-15769-1_8.

Almagor, S., B. Chapman, M. Hosseini, J. Ouaknine, and J. Worrell. (2018). "Effective Divergence Analysis for Linear Recurrence Sequences". In: *Proceedings of the 29th International Conference on Concurrency Theory, CONCUR 2018*. Ed. by S. Schewe and L. Zhang. Vol. 118. *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. 42:1–42:15. DOI: 10.4230/LIPICS.CONCUR.2018.42.

Asadi, A., K. Chatterjee, H. Fu, A. K. Goharshady, and M. Mahdavi. (2021). "Polynomial Reachability Witnesses via Stellensätze". In: *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2021:* ed. by S. N. Freund and E. Yahav. ACM. 772–787. DOI: 10.1145/3453483.3454076.

Bagnara, R., F. Mesnard, A. Pescetti, and E. Zaffanella. (2012). "A New Look at the Automatic Synthesis of Linear Ranking Functions". *Inf. Comput.* 215: 47–67. DOI: 10.1016/J.IC.2012.03.003.

Bakhirkin, A., J. Berdine, and N. Piterman. (2015). "A Forward Analysis for Recurrent Sets". In: *Proceedings of the 22nd International Symposium on Static Analysis, SAS 2015*. Ed. by S. Blazy and T. P. Jensen. Vol. 9291. *Lecture Notes in Computer Science*. Springer. 293–311. DOI: 10.1007/978-3-662-48288-9_17.

Bakhirkin, A. and N. Piterman. (2016). "Finding Recurrent Sets with Backward Analysis and Trace Partitioning". In: *Proceedings of the 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2016*. Ed. by M. Chechik and J. Raskin. Vol. 9636. *Lecture Notes in Computer Science*. Springer. 17–35. DOI: 10.1007/978-3-662-49674-9_2.

Banaszczyk, W., A. E. Litvak, A. Pajor, and S. J. Szarek. (1999). "The Flatness Theorem for Nonsymmetric Convex Bodies via the Local Theory of Banach Spaces". *Math. Oper. Res.* 24(3): 728–750. DOI: 10.1287/MOOR.24.3.728.

Ben-Amram, A. M. (2008). "Size-change Termination with Difference Constraints". *ACM Trans. Program. Lang. Syst.* 30(3): 16:1–16:31. DOI: 10.1145/1353445.1353450.

Ben-Amram, A. M. (2011). "Monotonicity Constraints for Termination in the Integer Domain". *Log. Methods Comput. Sci.* 7(3). DOI: 10.2168/LMCS-7(3:4)2011.

Ben-Amram, A. M. (2014). “The Hardness of Finding Linear Ranking Functions for Lasso Programs”. In: *Proceedings 5th International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2014*. Ed. by A. Peron and C. Piazza. Vol. 161. *EPTCS*. 32–45. DOI: 10.4204/EPTCS.161.6.

Ben-Amram, A. M. (2015). “Mortality of Iterated Piecewise Affine Functions Over the Integers: Decidability and Complexity”. *Computability*. 4(1): 19–56. DOI: 10.3233/COM-150032.

Ben-Amram, A. M., J. J. Doménech, and S. Genaim. (2019). “Multiphase-Linear Ranking Functions and Their Relation to Recurrent Sets”. In: *Proceedings of the 26th International Symposium on Static Analysis, SAS 2019*. Ed. by B. E. Chang. Vol. 11822. *Lecture Notes in Computer Science*. Springer. 459–480. DOI: 10.1007/978-3-030-32304-2_22.

Ben-Amram, A. M., J. J. Doménech, and S. Genaim. (2021). “Loops for which Multiphase-Linear Ranking Functions are Sufficient”. In: *Proceedings of the 17th International Workshop on Termination, WST 2021*. Ed. by S. Genaim. 60–74. URL: https://costa.fdi.ucm.es/wst2021/WST2021_proceedings.pdf.

Ben-Amram, A. M. and S. Genaim. (2013). “On the Linear Ranking Problem for Integer Linear-Constraint Loops”. In: *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2013*. Ed. by R. Giacobazzi and R. Cousot. ACM. 51–62. DOI: 10.1145/2429069.2429078.

Ben-Amram, A. M. and S. Genaim. (2014). “Ranking Functions for Linear-Constraint Loops”. *J. ACM*. 61(4): 26:1–26:55. DOI: 10.1145/2629488.

Ben-Amram, A. M. and S. Genaim. (2015). “Complexity of Bradley-Manna-Sipma Lexicographic Ranking Functions”. In: *Proceedings of the 27th International Conference on Computer Aided Verification, CAV 2015*. Ed. by D. Kroening and C. S. Pasareanu. Vol. 9207. *Lecture Notes in Computer Science*. Springer. 304–321. DOI: 10.1007/978-3-319-21668-3_18.

Ben-Amram, A. M. and S. Genaim. (2017). “On Multiphase-Linear Ranking Functions”. In: *Proceedings of the 29th International Conference on Computer Aided Verification, CAV 2017*. Ed. by R. Majumdar and V. Kuncak. Vol. 10427. *Lecture Notes in Computer Science*. Springer. 601–620. DOI: 10.1007/978-3-319-63390-9_32.

Ben-Amram, A. M., S. Genaim, and A. N. Masud. (2012). “On the Termination of Integer Loops”. *ACM Trans. Program. Lang. Syst.* 34(4): 16:1–16:24. DOI: 10.1145/2400676.2400679.

Beyer, D. and M. E. Keremoglu. (2011). “CPAchecker: A Tool for Configurable Software Verification”. In: *Proceedings of the 23rd International Conference on Computer Aided Verification, CAV 2011*. Ed. by G. Gopalakrishnan and S. Qadeer. Vol. 6806. *Lecture Notes in Computer Science*. Springer. 184–190. DOI: 10.1007/978-3-642-22110-1_16.

Blass, A. and Y. Gurevich. (2008). “Program Termination and Well Partial Orderings”. *ACM Trans. Comput. Log.* 9(3): 18:1–18:26. DOI: 10.1145/1352582.1352586.

Blondel, V. D., O. Bournez, P. Koiran, C. H. Papadimitriou, and J. N. Tsitsiklis. (2001). “Deciding Stability and Mortality of Piecewise Affine Dynamical Systems”. *Theor. Comput. Sci.* 255(1-2): 687–696. DOI: 10.1016/S0304-3975(00)00399-6.

Boigelot, B. (2003). “On Iterating Linear Transformations Over Recognizable Sets of Integers”. *Theor. Comput. Sci.* 309(1-3): 413–468. DOI: 10.1016/S0304-3975(03)00314-1.

Borralleras, C., M. Brockschmidt, D. Larraz, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. (2017). “Proving Termination Through Conditional Termination”. In: *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2017*. Ed. by A. Legay and T. Margaria. Vol. 10205. *Lecture Notes in Computer Science*. 99–117. DOI: 10.1007/978-3-662-54577-5_6.

Bozga, M., R. Iosif, and F. Konecný. (2014). “Deciding Conditional Termination”. *Log. Methods Comput. Sci.* 10(3). DOI: 10.2168/LMCS-10(3:8)2014.

Bozzelli, L. and S. Pinchinat. (2014). “Verification of Gap-Order Constraint Abstractions of Counter Systems”. *Theor. Comput. Sci.* 523: 1–36. DOI: 10.1016/J.TCS.2013.12.002.

Bradley, A. R., Z. Manna, and H. B. Sipma. (2005a). “Linear Ranking with Reachability”. In: *Proceedings of the 17th International Conference on Computer Aided Verification, CAV 2005*. Ed. by K. Etessami and S. K. Rajamani. Vol. 3576. *Lecture Notes in Computer Science*. Springer. 491–504. DOI: 10.1007/11513988_48.

Bradley, A. R., Z. Manna, and H. B. Sipma. (2005b). “Termination Analysis of Integer Linear Loops”. In: *Proceedings of the 16th International Conference on Concurrency Theory, CONCUR 2005*. Ed. by M. Abadi and L. de Alfaro. Vol. 3653. *Lecture Notes in Computer Science*. Springer. 488–502. DOI: 10.1007/11539452_37.

Bradley, A. R., Z. Manna, and H. B. Sipma. (2005c). “Termination of Polynomial Programs”. In: *Proceedings of the 6th International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI 2005*. Ed. by R. Cousot. Vol. 3385. *Lecture Notes in Computer Science*. Springer. 113–129. DOI: 10.1007/978-3-540-30579-8_8.

Bradley, A. R., Z. Manna, and H. B. Sipma. (2005d). “The Polyranking Principle”. In: *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming, ICALP 2005*. Ed. by L. Caires, G. F. Italiano, L. Monteiro, and M. Yung. Vol. 3580. *Lecture Notes in Computer Science*. Springer. 1349–1361. DOI: 10.1007/11523468_109.

Braverman, M. (2006). “Termination of Integer Linear Programs”. In: *Proceedings of the 18th International Conference on Computer Aided Verification. CAV 2006*. Ed. by T. Ball and R. B. Jones. Vol. 4144. *Lecture Notes in Computer Science*. Springer. 372–385. DOI: 10.1007/11817963_34.

Brockschmidt, M., B. Cook, and C. Fuhs. (2013). “Better Termination Proving through Cooperation”. In: *Proceedings of the 25th International Conference on Computer Aided Verification, CAV 2013*. Ed. by N. Sharygina and H. Veith. Vol. 8044. *Lecture Notes in Computer Science*. Springer. 413–429. DOI: 10.1007/978-3-642-39799-8_28.

Brockschmidt, M., F. Emmes, S. Falke, C. Fuhs, and J. Giesl. (2016). "Analyzing Runtime and Size Complexity of Integer Programs". *ACM Trans. Program. Lang. Syst.* 38(4): 13. DOI: 10.1145/2866575.

Brockschmidt, M., T. Ströder, C. Otto, and J. Giesl. (2011). "Automated Detection of Non-termination and NullPointerExceptions for Java Bytecode". In: *Formal Verification of Object-Oriented Software, FoVeOOS'11*. Ed. by B. Beckert, F. Damiani, and D. Gurov. Vol. 7421. *Lecture Notes in Computer Science*. Springer. 123–141. DOI: 10.1007/978-3-642-31762-0_9.

Bruynooghe, M., M. Codish, J. P. Gallagher, S. Genaim, and W. Vanhoof. (2007). "Termination Analysis of Logic Programs Through Combination of Type-Based Norms". *ACM Trans. Program. Lang. Syst.* 29(2): 10. DOI: 10.1145/1216374.1216378.

Canny, J. F. (1988). "Some Algebraic and Geometric Computations in PSPACE". In: *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, STOC 1988*. Ed. by J. Simon. ACM. 460–467. DOI: 10.1145/62212.62257.

Charles, P. J., J. M. Howe, and A. King. (2009). "Integer Polyhedra for Program Analysis". In: *Proceedings of the 5th International Conference on Algorithmic Aspects in Information and Management, AAIM 2009*. Ed. by A. V. Goldberg and Y. Zhou. Vol. 5564. *Lecture Notes in Computer Science*. Springer. 85–99. DOI: 10.1007/978-3-642-02158-9_9.

Chen, H. Y., B. Cook, C. Fuhs, K. Nimkar, and P. W. O'Hearn. (2014). "Proving Nontermination via Safety". In: *Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2014*. Ed. by E. Ábrahám and K. Havelund. Vol. 8413. *Lecture Notes in Computer Science*. Springer. 156–171. DOI: 10.1007/978-3-642-54862-8_11.

Chen, H. Y., S. Flur, and S. Mukhopadhyay. (2015). "Termination Proofs for Linear Simple Loops". *Int. J. Softw. Tools Technol. Transf.* 17(1): 47–57. DOI: 10.1007/S10009-013-0288-8.

Chen, Y., B. Xia, L. Yang, N. Zhan, and C. Zhou. (2007). “Discovering Non-linear Ranking Functions by Solving Semi-algebraic Systems”. In: *Proceedings of the 4th International Colloquium on Theoretical Aspects of Computing, ICTAC 2007*. Ed. by C. B. Jones, Z. Liu, and J. Woodcock. Vol. 4711. *Lecture Notes in Computer Science*. Springer. 34–49. DOI: 10.1007/978-3-540-75292-9_3.

Codish, M., S. Genaim, M. Bruynooghe, J. P. Gallagher, and W. Vanhoof. (2003). “One Loop at a Time”. In: *Proceedings of the 6th International Workshop on Termination, WST 2003*. Ed. by A. Rubio. *Technical Report DSIC-II/15/03*. Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia. 1–4.

Codish, M., V. Lagoon, and P. J. Stuckey. (2005). “Testing for Termination with Monotonicity Constraints”. In: *Proceedings of the 21st International Conference on Logic Programming, ICLP 2005*. Ed. by M. Gabbrielli and G. Gupta. Vol. 3668. *Lecture Notes in Computer Science*. Springer. 326–340. DOI: 10.1007/11562931_25.

Codish, M. and C. Taboch. (1999). “A Semantic Basis for the Termination Analysis of Logic Programs”. *J. Log. Program.* 41(1): 103–123. DOI: 10.1016/S0743-1066(99)00006-0.

Colón, M., S. Sankaranarayanan, and H. Sipma. (2003). “Linear Invariant Generation Using Non-linear Constraint Solving”. In: *Proceedings of the 15th International Conference on Computer Aided Verification, CAV 2003*. Ed. by W. A. H. Jr. and F. Somenzi. Vol. 2725. *Lecture Notes in Computer Science*. Springer. 420–432. DOI: 10.1007/978-3-540-45069-6_39.

Colón, M. and H. Sipma. (2001). “Synthesis of Linear Ranking Functions”. In: *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2001*. Ed. by T. Margaria and W. Yi. Vol. 2031. *Lecture Notes in Computer Science*. Springer. 67–81. DOI: 10.1007/3-540-45319-9_6.

Colón, M. and H. Sipma. (2002). “Practical Methods for Proving Program Termination”. In: *Proceedings of the 14th International Conference on Computer Aided Verification, CAV 2002*. Ed. by E. Brinksma and K. G. Larsen. Vol. 2404. *Lecture Notes in Computer Science*. Springer. 442–454. DOI: 10.1007/3-540-45657-0_36.

Combot, T. (2025). “Computing Linear Relations Between Polynomial Roots”. *Mathematics of Computation*. Mar. DOI: 10.1090/mcom/4081.

Cook, B., C. Fuhs, K. Nimkar, and P. W. O’Hearn. (2014). “Disproving Termination with Overapproximation”. In: *Proceedings of the 2014 Conference on Formal Methods in Computer-Aided Design, FMCAD 2014*. IEEE. 67–74. DOI: 10.1109/FMCAD.2014.6987597.

Cook, B., D. Kroening, P. Rümmer, and C. M. Wintersteiger. (2013). “Ranking Function Synthesis for Bit-Vector Relations”. *Formal Methods Syst. Des.* 43(1): 93–120. DOI: 10.1007/S10703-013-0186-4.

Cook, B., A. Podelski, and A. Rybalchenko. (2006). “Termination Proofs for Systems Code”. In: *Proceedings of the 27th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2006*. Ed. by M. I. Schwartzbach and T. Ball. ACM. 415–426. DOI: 10.1145/1133981.1134029.

Cousot, P. (2005). “Proving Program Invariance and Termination by Parametric Abstraction, Lagrangian Relaxation and Semidefinite Programming”. In: *Proceedings of the 6th International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI 2005*. Ed. by R. Cousot. Vol. 3385. *Lecture Notes in Computer Science*. Springer. 1–24. DOI: 10.1007/978-3-540-30579-8_1.

Cousot, P. and N. Halbwachs. (1978). “Automatic Discovery of Linear Restraints Among Variables of a Program”. In: *Proceedings of the 5th Annual ACM Symposium on Principles of Programming Languages, POPL 1978*. Ed. by A. V. Aho, S. N. Zilles, and T. G. Szymanski. ACM Press. 84–96. DOI: 10.1145/512760.512770.

Czerwinski, W. and L. Orlikowski. (2021). “Reachability in Vector Addition Systems is Ackermann-complete”. In: *Proceedings of the 62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*. IEEE. 1229–1240. DOI: 10.1109/FOCS52979.2021.00120.

Dai, L. and B. Xia. (2012). “Non-termination Sets of Simple Linear Loops”. In: *Proceedings of the 9th International Colloquium on Theoretical Aspects of Computing, ICTAC 2012*. Ed. by A. Roychoudhury and M. D’Souza. Vol. 7521. *Lecture Notes in Computer Science*. Springer. 61–73. DOI: 10.1007/978-3-642-32943-2_5.

Dershowitz, N., N. Lindenstrauss, Y. Sagiv, and A. Serebrenik. (2001). "A General Framework for Automatic Termination Analysis of Logic Programs". *Appl. Algebra Eng. Commun. Comput.* 12(1/2): 117–156. DOI: 10.1007/S002000100065.

Din, M. S. E. and L. Zhi. (2010). "Computing Rational Points in Convex Semialgebraic Sets and Sum of Squares Decompositions". *SIAM J. Optim.* 20(6): 2876–2889. DOI: 10.1137/090772459.

Doménech, J. J., J. P. Gallagher, and S. Genaim. (2019). "Control-Flow Refinement by Partial Evaluation, and its Application to Termination and Cost Analysis". *Theory Pract. Log. Program.* 19(5-6): 990–1005. DOI: 10.1017/S1471068419000310.

Doornbos, H. and B. von Karger. (1998). "On the Union of Well-Founded Relations". *Log. J. IGPL.* 6(2): 195–201. DOI: 10.1093/JIGPAL/6.2.195.

Everest, G., A. J. van der Poorten, I. E. Shparlinski, and T. Ward. (2003). *Recurrence Sequences.* Vol. 104. *Mathematical surveys and monographs.* American Mathematical Society. ISBN: 978-0-8218-3387-2. DOI: 10.1090/surv/104.

Feautrier, P. (1992a). "Some Efficient Solutions to the Affine Scheduling Problem. I. One-Dimensional Time". *Int. J. Parallel Program.* 21(5): 313–347. DOI: 10.1007/BF01407835.

Feautrier, P. (1992b). "Some Efficient Solutions to the Affine Scheduling Problem. Part II. Multidimensional Time". *Int. J. Parallel Program.* 21(6): 389–420. DOI: 10.1007/BF01379404.

Floyd, R. W. (1967). "Assigning Meanings to Programs". In: *Proceedings of Symposia in Applied Mathematics.* Ed. by J. T. Schwartz. Vol. XIX. Providence, RI: American Mathematical Society. 19–32.

Frohn, F. and J. Giesl. (2019). "Termination of Triangular Integer Loops is Decidable". In: *Proceedings of the 31st International Conference on Computer Aided Verification, CAV 2019.* Ed. by I. Dillig and S. Tasiran. Vol. 11562. *Lecture Notes in Computer Science.* Springer. 426–444. DOI: 10.1007/978-3-030-25543-5_24.

Frohn, F. and J. Giesl. (2023). "Proving Non-Termination by Acceleration Driven Clause Learning (Short Paper)". In: *Proceedings of the 29th International Conference on Automated Deduction, CADE 2023*. Ed. by B. Pientka and C. Tinelli. Vol. 14132. *Lecture Notes in Computer Science*. Springer. 220–233. DOI: 10.1007/978-3-031-38499-8_13.

Ganty, P. and S. Genaim. (2013). "Proving Termination Starting from the End". In: *Proceedings of the 25th International Conference on Computer Aided Verification, CAV 2013*. Ed. by N. Sharygina and H. Veith. Vol. 8044. *Lecture Notes in Computer Science*. Springer. 397–412. DOI: 10.1007/978-3-642-39799-8_27.

Geser, A. (1990). "Relative Termination". *PhD thesis*. University of Passau, Germany. URL: https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui/Ulmer_Informatik_Berichte/1991/UIB-1991-03.pdf.

Gonnord, L., D. Monniaux, and G. Radanne. (2015). "Synthesis of Ranking Functions Using Extremal Counterexamples". In: *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2015*. Ed. by D. Grove and S. M. Blackburn. ACM. 608–618. DOI: 10.1145/2737924.2737976.

Guilmant, Q., E. Lefaucheux, J. Ouaknine, and J. Worrell. (2024). "The 2-Dimensional Constraint Loop Problem Is Decidable". In: *Proceedings of the 51st International Colloquium on Automata, Languages, and Programming, ICALP 2024*. Ed. by K. Bringmann, M. Grohe, G. Puppis, and O. Svensson. Vol. 297. *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. 140:1–140:21. DOI: 10.4230/LIPICS.ICALP.2024.140.

Gulwani, S. and F. Zuleger. (2010). "The Reachability-Bound Problem". In: *Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2010*. Ed. by B. G. Zorn and A. Aiken. ACM. 292–304. DOI: 10.1145/1806596.1806630.

Gupta, A., T. A. Henzinger, R. Majumdar, A. Rybalchenko, and R. Xu. (2008). "Proving Non-Termination". In: *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008*. Ed. by G. C. Necula and P. Wadler. ACM. 147–158. DOI: 10.1145/1328438.1328459.

Halava, V., T. Harju, M. Hirvensalo, and J. Karhumaki. (2005). “Skolem’s Problem – On the Border between Decidability and Undecidability”. *Tech. rep.* No. 683. Turku Centre for Computer Science. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=12375e47d55f22cf1e256aa6881c6a9e8329d3bc.

Hark, M., F. Frohn, and J. Giesl. (2020). “Polynomial Loops: Beyond Termination”. In: ed. by E. Albert and L. Kovács. Vol. 73. *EPiC Series in Computing*. EasyChair. 279–297. DOI: 10.29007/NXV1.

Hark, M., F. Frohn, and J. Giesl. (2025). “Termination of Triangular Polynomial Loops”. *Formal Methods Syst. Des.* 65(1): 70–132. DOI: 10.1007/S10703-023-00440-Z.

Hartmann, M. E. (1988). “Cutting Planes and the Complexity of the Integer Hull”. *Tech. rep.* No. 819. School of Operations Research and Industrial Engineering, Cornell University. URL: https://hdl.handle.net/1813/8702.

Heizmann, M., N. D. Jones, and A. Podelski. (2010). “Size-Change Termination and Transition Invariants”. In: *Proceedings of the 17th International Symposium on Static Analysis, SAS 2010*. Ed. by R. Cousot and M. Martel. Vol. 6337. *Lecture Notes in Computer Science*. Springer. 22–50. DOI: 10.1007/978-3-642-15769-1_4.

Hosseini, M., J. Ouaknine, and J. Worrell. (2019). “Termination of Linear Loops over the Integers”. In: *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming, ICALP 2019*. Ed. by C. Baier, I. Chatzigiannakis, P. Flocchini, and S. Leonardi. Vol. 132. *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. 118:1–118:13. DOI: 10.4230/LIPICS.ICALP.2019.118.

Jeannet, B., P. Schrammel, and S. Sankaranarayanan. (2014). “Abstract Acceleration of General Linear Loops”. In: *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2014*. Ed. by S. Jagannathan and P. Sewell. ACM. 529–540. DOI: 10.1145/2535838.2535843.

Kenison, G., J. Nieuwveld, J. Ouaknine, and J. Worrell. (2023). "Positivity Problems for Reversible Linear Recurrence Sequences". In: *Proceedings of the 50th International Colloquium on Automata, Languages, and Programming, ICALP 2023*. Ed. by K. Etessami, U. Feige, and G. Puppis. Vol. 261. *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. 130:1–130:17. DOI: 10.4230/LIPICS.ICALP.2023.130.

Khachiyan, L. and L. Porkolab. (1997). "Computing Integral Points in Convex Semi-Algebraic Sets". In: *Proceedings of the 38th Annual Symposium on Foundations of Computer Science, FOCS 1997*. 162–171. DOI: 10.1109/SFCS.1997.646105.

Khinchin, A. Y. (1948). "Dirichlet's Principle in the Theory of Diophantine Approximations". *Uspekhi Matematicheskikh Nauk*. 3(3): 3–28. URL: https://www.mathnet.ru/eng/rm8706.

Kincaid, Z., J. Breck, J. Cyphert, and T. W. Reps. (2019). "Closed Forms for Numerical Loops". *Proc. ACM Program. Lang.* 3(POPL): 55:1–55:29. DOI: 10.1145/3290368.

Kroening, D., N. Sharygina, A. Tsitovich, and C. M. Wintersteiger. (2010). "Termination Analysis with Compositional Transition Invariants". In: *Proceedings of the 22nd International Conference on Computer Aided Verification, CAV 2010*. Ed. by T. Touili, B. Cook, and P. B. Jackson. Vol. 6174. *Lecture Notes in Computer Science*. Springer. 89–103. DOI: 10.1007/978-3-642-14295-6_9.

Larraz, D., K. Nimkar, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. (2014). "Proving Non-termination Using Max-SMT". In: *Proceedings of the 26th International Conference on Computer Aided Verification, CAV 2014*. Ed. by A. Biere and R. Bloem. Vol. 8559. *Lecture Notes in Computer Science*. Springer. 779–796. DOI: 10.1007/978-3-319-08867-9_52.

Larraz, D., A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. (2013). "Proving Termination of Imperative Programs using Max-SMT". In: *Proceedings of the 2013 Conference on Formal Methods in Computer-Aided Design, FMCAD 2013*. IEEE. 218–225. DOI: 10.1109/FMCAD.2013.6679413.

Le, T. C., S. Qin, and W. Chin. (2015). "Termination and Non-Termination Specification Inference". In: *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2015*. Ed. by D. Grove and S. M. Blackburn. ACM. 489–498. DOI: 10.1145/2737924.2737993.

Lee, C. S., N. D. Jones, and A. M. Ben-Amram. (2001). "The Size-Change Principle for Program Termination". In: *Proceedings of the 28th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2001*. Ed. by C. Hankin and D. Schmidt. ACM. 81–92. DOI: 10.1145/360204.360210.

Leike, J. and M. Heizmann. (2015). "Ranking Templates for Linear Loops". *Log. Methods Comput. Sci.* 11(1): 1–27. DOI: 10.2168/LMCS-11(1:16)2015.

Leike, J. and M. Heizmann. (2018). "Geometric Nontermination Arguments". In: *Proceedings of the 24th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2018*. Ed. by D. Beyer and M. Huisman. Vol. 10806. *Lecture Notes in Computer Science*. Springer. 266–283. DOI: 10.1007/978-3-319-89963-3_16.

Leroux, J. (2021). "The Reachability Problem for Petri Nets is Not Primitive Recursive". In: *Proceedings of the 62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*. IEEE. 1241–1252. DOI: 10.1109/FOCS52979.2021.00121.

Li, Y. (2014). "A Recursive Decision Method for Termination of Linear Programs". In: *Proceedings of the 2014 Symposium on Symbolic-Numeric Computation, SNC 2014*. Ed. by L. Zhi and S. M. Watt. ACM. 97–106. DOI: 10.1145/2631948.2631966.

Li, Y. (2017). "Witness to Non-Termination of Linear Programs". *Theor. Comput. Sci.* 681: 75–100. DOI: 10.1016/J.TCS.2017.03.036.

Li, Y., W. Wu, and Y. Feng. (2020). "On Ranking Functions for Single-Path Linear-Constraint Loops". *Int. J. on Software Tools for Technology Transfer.* 22(6): 655–666. DOI: 10.1007/S10009-019-00549-9.

Li, Y., G. Zhu, and Y. Feng. (2016). "The L-Depth Eventual Linear Ranking Functions for Single-Path Linear Constraint Loops". In: *Proceedings of the 10th International Symposium on Theoretical Aspects of Software Engineering, TASE 2016*. IEEE Computer Society. 30–37. DOI: 10.1109/TASE.2016.8.

Lindenstrauss, N. and Y. Sagiv. (1997). "Automatic Termination Analysis of Logic Programs". In: *Proceedings of the 14th International Conference on Logic Programming, ICLP 1997*. Ed. by L. Naish. MIT Press. 63–77. DOI: 10.7551/mitpress/4299.003.0011.

Magill, S., M. Tsai, P. Lee, and Y. Tsay. (2010). "Automatic Numeric Abstractions for Heap-Manipulating Programs". In: *Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010*. Ed. by M. V. Hermenegildo and J. Palsberg. ACM. 211–222. DOI: 10.1145/1706299.1706326.

Masser, D. W. (1988). "Linear Relations on Algebraic Groups". In: *New Advances in Transcendence Theory*. Ed. by A. Baker. Cambridge University Press. 248–262. DOI: 10.1017/CBO9780511897184.016.

Maurica, F., F. Mesnard, and É. Payet. (2016). "On the Linear Ranking Problem for Simple Floating-Point Loops". In: *Proceedings of the 23rd International Symposium on Static Analysis, SAS 2016*. Ed. by X. Rival. Vol. 9837. *Lecture Notes in Computer Science*. Springer. 300–316. DOI: 10.1007/978-3-662-53413-7_15.

Mesnard, F. and A. Serebrenik. (2008). "Recurrence with Affine Level Mappings is P-Time Decidable for CLP(R)". *Theory Pract. Log. Program.* 8(1): 111–119. DOI: 10.1017/S1471068407003122.

Minsky, M. L. (1967). *Computation: Finite and Infinite Machines*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc. URL: https://dl.acm.org/doi/book/10.5555/1095587.

Mitterwallner, F., A. Middeldorp, and R. Thiemann. (2024). "Linear Termination is Undecidable". In: *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2024*. Ed. by P. Sobocinski, U. D. Lago, and J. Esparza. ACM. 57:1–57:12. DOI: 10.1145/3661814.3662081.

Moyen, J. (2009). "Resource Control Graphs". *ACM Trans. Comput. Log.* 10(4): 29:1–29:44. DOI: 10.1145/1555746.1555753.

Neumann, E., J. Ouaknine, and J. Worrell. (2020). “On Ranking Function Synthesis and Termination for Polynomial Programs”. In: *Proceedings of the 31st International Conference on Concurrency Theory, CONCUR 2020*. Ed. by I. Konnov and L. Kovács. Vol. 171. *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. 15:1–15:15. DOI: 10.4230/LIPICS.CONCUR.2020.15.

Ouaknine, J., J. S. Pinto, and J. Worrell. (2015). “On Termination of Integer Linear Loops”. In: *Proceedings of the 2th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*. Ed. by P. Indyk. SIAM. 957–969. DOI: 10.1137/1.9781611973730.65.

Ouaknine, J. and J. Worrell. (2014a). “On the Positivity Problem for Simple Linear Recurrence Sequences,” in: *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming, ICALP 2014*. Ed. by J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias. Vol. 8573. *Lecture Notes in Computer Science*. Springer. 318–329. DOI: 10.1007/978-3-662-43951-7_27.

Ouaknine, J. and J. Worrell. (2014b). “Positivity Problems for Low-Order Linear Recurrence Sequences”. In: *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*. Ed. by C. Chekuri. SIAM. 366–379. DOI: 10.1137/1.9781611973402.27.

Ouaknine, J. and J. Worrell. (2014c). “Ultimate Positivity is Decidable for Simple Linear Recurrence Sequences”. In: *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming, ICALP 2014*. Ed. by J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias. Vol. 8573. *Lecture Notes in Computer Science*. Springer. 330–341. DOI: 10.1007/978-3-662-43951-7_28.

Podelski, A. and A. Rybalchenko. (2004a). “A Complete Method for the Synthesis of Linear Ranking Functions”. In: *Proceedings of the 5th International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI 2004*. Ed. by B. Steffen and G. Levi. Vol. 2937. *Lecture Notes in Computer Science*. Springer. 239–251. DOI: 10.1007/978-3-540-24622-0_20.

Podelski, A. and A. Rybalchenko. (2004b). "Transition Invariants". In: *Proceedings of the 19th IEEE Symposium on Logic in Computer Science, LICS 2004*. IEEE Computer Society. 32–41. DOI: 10.1109/LICS.2004.1319598.

Podelski, A. and A. Rybalchenko. (2007). "Transition Predicate Abstraction and Fair Termination". *ACM Trans. Program. Lang. Syst.* 29(3): 15. DOI: 10.1145/1232420.1232422.

Rozenberg, G. and A. Salomaa. (1994). *Cornerstones of Undecidability*. Prentice Hall, Inc. ISBN: 978-0-13-297425-7. URL: https://dl.acm.org/doi/10.5555/201013.

Salomaa, A. and M. Soittola. (1978). *Automata-Theoretic Aspects of Formal Power Series. Texts and Monographs in Computer Science*. Springer. ISBN: 978-0-387-90282-1. DOI: 10.1007/978-1-4612-6264-0.

Schrijver, A. (1999). *Theory of Linear and Integer Programming. Wiley-Interscience series in discrete mathematics and optimization*. Wiley. ISBN: 978-0-471-98232-6.

Shen, L., M. Wu, Z. Yang, and Z. Zeng. (2013). "Generating Exact Nonlinear Ranking Functions by Symbolic-Numeric Hybrid Method". *J. Syst. Sci. Complex.* 26(2): 291–301. DOI: 10.1007/S11424-013-1004-1.

Sohn, K. and A. V. Gelder. (1991). "Termination Detection in Logic Programs using Argument Sizes". In: *Proceedings of the 10th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 1991*. Ed. by D. J. Rosenkrantz. ACM Press. 216–226. DOI: 10.1145/113413.113433.

Soittola, M. (1975). "On DOL Synthesis Problem". In: *Automata, Languages, Development*. Ed. by A. Lindenmayer and G. Rozenberg. North-Holland Publishing Company. 313–321.

Spoto, F., F. Mesnard, and É. Payet. (2010). "A Termination Analyzer for Java Bytecode Based on Path-Length". *ACM Trans. Program. Lang. Syst.* 32(3): 8:1–8:70. DOI: 10.1145/1709093.1709095.

Tiwari, A. (2004). "Termination of Linear Programs". In: *Proceedings of the 16th International Conference on Computer Aided Verification, CAV 2004*. Ed. by R. Alur and D. A. Peled. Vol. 3114. *Lecture Notes in Computer Science*. Springer. 70–82. DOI: 10.1007/978-3-540-27813-9_6.

Turing, A. M. (1948). "Checking a Large Routine". In: *Report of a Conference on High Speed Automatic Calculating Machines*. 67–69.

Urban, C. (2013). "The Abstract Domain of Segmented Ranking Functions". In: *Proceedings of the 20th International Symposium on Static Analysis, SAS 2013*. Ed. by F. Logozzo and M. Fähndrich. Vol. 7935. *Lecture Notes in Computer Science*. Springer. 43–62. DOI: 10.1007/978-3-642-38856-9_5.

Urban, C. and A. Miné. (2014). "An Abstract Domain to Infer Ordinal-Valued Ranking Functions". In: *Proceedings of the 23rd European Symposium on Programming, ESOP 2014*. Ed. by Z. Shao. Vol. 8410. *Lecture Notes in Computer Science*. Springer. 412–431. DOI: 10.1007/978-3-642-54833-8_22.

Velroyen, H. and P. Rümmer. (2008). "Non-termination Checking for Imperative Programs". In: *Proceedings of the 2nd International Conference on Tests and Proofs, TAP 2008*. Ed. by B. Beckert and R. Hähnle. Vol. 4966. *Lecture Notes in Computer Science*. Springer. 154–170. DOI: 10.1007/978-3-540-79124-9_11.

Xia, B., L. Yang, N. Zhan, and Z. Zhang. (2011). "Symbolic Decision Procedure for Termination of Linear Programs". *Formal Aspects Comput.* 23(2): 171–190. DOI: 10.1007/S00165-009-0144-5.

Yuan, Y., Y. Li, and W. Shi. (2021). "Detecting Multiphase Linear Ranking Functions for Single-Path Linear-Constraint Loops". *Int. J. Softw. Tools Technol. Transf.* 23(1): 55–67. DOI: 10.1007/S10009-019-00527-1.

Zhu, G., Y. Li, and W. Wu. (2016). "Eventual Linear Ranking Functions for Multi-Path Linear Loops". In: *Proceedings of the 2016 IEEE Information Technology, Networking, Electronic and Automation Control Conference, ITNEC 2016*. 331–337. DOI: 10.1109/ITNEC.2016.7560376.

Zhu, S. and Z. Kincaid. (2021). "Reflections on Termination of Linear Loops". In: *Proceedings of the 33rd International Conference on Computer Aided Verification. CAV 2021*. Ed. by A. Silva and K. R. M. Leino. Vol. 12760. *Lecture Notes in Computer Science*. Springer. 51–74. DOI: 10.1007/978-3-030-81688-9_3.

Zhu, S. and Z. Kincaid. (2024). “Breaking the Mold: Nonlinear Ranking Function Synthesis Without Templates”. In: *Proceedings of the 36th International Conference on Computer Aided Verification, CAV 2024*. Ed. by A. Gurfinkel and V. Ganesh. Vol. 14681. *Lecture Notes in Computer Science*. Springer. 431–452. DOI: 10.1007/978-3-031-65627-9_21.

Zuleger, F. (2015). “Asymptotically Precise Ranking Functions for Deterministic Size-Change Systems”. In: *Proceedings of the 10th International Computer Science Symposium in Russia, Theory and Applications, CSR 2015*. Ed. by L. D. Beklemishev and D. V. Musatov. Vol. 9139. *Lecture Notes in Computer Science*. Springer. 426–442. DOI: 10.1007/978-3-319-20297-6_27.

Zuleger, F. (2018). “Inductive Termination Proofs with Transition Invariants and Their Relationship to the Size-Change Abstraction”. In: *Proceedings of the 25th International Symposium on Static Analysis, SAS 2018*. Ed. by A. Podelski. Vol. 11002. *Lecture Notes in Computer Science*. Springer. 423–444. DOI: 10.1007/978-3-319-99725-4_25.