Ranking Functions for Linear-Constraint Loops

Amir M. Ben-Amram

The Academic College of Tel-Aviv Yaffo
amirben@mta.ac.il

Abstract

Ranking functions are a tool successfully used in termination analysis, complexity analysis, and program parallelization. Among the different types of ranking functions and approaches to finding them, this talk will concentrate on functions that are found by linear programming techniques. The setting is that of a loop that has been pre-abstracted so that it is described by linear constraints over a finite set of numeric variables. I will review results (more or less recent) regarding the search for ranking functions which are either linear or lexicographic-linear.

1 Ranking Functions

Consider a program, viewed abstractly as a "single step" relation $\vdash$ mapping a state s to the next state s'. Thus a computation of the program is a chain $s_0 \vdash s_1 \vdash s_2 \vdash \dots$. In termination analysis, we wish to prove that there are no infinite chains. The *ranking function method* is to find a function ρ that maps program states into a well-founded ordered set W, such that $\rho(s) > \rho(s')$ whenever $s \vdash s'$. Well-founded orders have no infinite strictly-descending chains, so all computations must be finite. In practice, the granularity of the "next step" relation may vary. In many applications, the step in question represents a single application of a loop's body.

The uses of a ranking function are several. Termination *per se* is a basic concern in program verification. Secondly, in program transformation, one wants to ensure that generated code does not breach termination properties, and moreover, that the transformation process itself (e.g., a partial evaluation) is terminating. When the co-domain of the ranking function is the non-negative integers, the function's value on entrance to the loop bounds the number of loop iterations. Thus, such ranking functions are useful in time-complexity analysis [1, 20]. More complex ranking functions, in particular, *lexicographic*, can also be used for this purpose [3]. An estimate of the running time of a program (or just one loop) can be used in several program transformation tasks, such as optimization and parallelization. Some of the techniques mentioned below have been developed for loop parallelization, concurrently or even before their adoption in termination analysis [16, 17].

The main sources for more details on results mentioned in this talk are [4] which summarizes several results on linear ranking functions, [3] on lexicographic ranking functions, and, for recent results, my joint work with Samir Genaim [5, 6].

2 Program Representation

Very often, to generate ranking functions, the semantics of the loop is represented (possibly over-approximated) by linear constraints. A *single-path* linear-constraint loop (*SLC* for short) over variables $x_1, \dots, x_n$ has the form

$$\textit{while}\ (B\mathbf{x} \le \mathbf{b})\ \textit{do}\ A\begin{pmatrix}\mathbf{x}\\ \mathbf{x}'\end{pmatrix} \le \mathbf{c} \qquad (1)$$

where $\mathbf{x}$ and $\mathbf{x}'$ are column vectors, and for some $p, q > 0$, $B \in \mathbb{Q}^{p\times n}$, $A \in \mathbb{Q}^{q\times 2n}$, $\mathbf{b} \in \mathbb{Q}^p$, $\mathbf{c} \in \mathbb{Q}^q$. Note that the coefficients are rationals. The loop variables range over the integers or over the rationals[1]. For notational convenience, we let $\mathbf{x}''$ stand for $\binom{\mathbf{x}}{\mathbf{x}'}$. In (1), the constraint $B\mathbf{x} \le \mathbf{b}$ is the *loop guard*, and $a\mathbf{x}'' \le \mathbf{c}$ is the *loop update*. A case of importance is *deterministic linear update*, which can be expressed as $\mathbf{x}' = A'\mathbf{x} + \mathbf{c}'$ for $A', \mathbf{c}'$ of appropriate dimensions.

The transition relation of an *SLC* loop is a convex polyhedron in $\mathbb{Q}^{2n}$, usually denoted by $\mathcal{Q}$ (for definitions of convex polyhedra and their essential properties, see [22]).

Whereas the loop update above is a conjunction of inequalities, a *multipath* linear-constraint loop (*MLC* for short) is described by a disjunction of such conjunctions. It is written as

$$\mathit{loop} \bigvee_{i=1}^{k} \left[B_i\mathbf{x} \le \mathbf{b}_i \;\wedge\; A_i \begin{pmatrix}\mathbf{x}\\ \mathbf{x}'\end{pmatrix} \le \mathbf{c}_i \right] \tag{2}$$

Or (which amounts to the same) as a list $\mathcal{Q}_1, \dots, \mathcal{Q}_k$ of transition polyhedra. The *MLC* form more faithfully represents loops that have a body which is not straight-line but involves branching, or where a non-linear operation has been abstracted to a disjunction of linear constraints.

A third form of program representation, generalizing *MLC* loops, is a control-flow graph (CFG) annotated with linear constraints on its arcs, as used in [3] (and, with somewhat different terminology, in [18, 23]). Generally speaking, one may say that this form is used when a whole program module is translated to a constraint representation before calling the termination algorithm (examples of tools that has this flow are JULIA [24] for Java bytecode programs, Termilog [15] and Terminweb [10] for Prolog). On the other hand, in a tool such as Terminator [14], only *SLC* loops are used: basically, the tool looks for potentially non-terminating cycles, and applies the ranking-function generation algorithm to one at a time. COSTA [2] handles one *MLC* loop at a time. In the next two sections, we concentrate on *SLC* and *MLC* loops.

3 Linear Ranking Functions

A *linear ranking function* (*LRF*) has the form $\rho(\mathbf{x}) = \vec{\lambda} \cdot \mathbf{x} + \lambda_0$, with $(\lambda_0, \vec{\lambda}) \in \mathbb{Q}^{n+1}$, and is required to satisfy: if $\mathbf{x} \vdash \mathbf{x}'$, then $\rho(\mathbf{x}) \ge \rho(\mathbf{x}') + 1$ and $\rho(\mathbf{x}) \ge 0$. Written explicitly, the conditions are:

$$\vec{\lambda} \cdot \mathbf{x} + \lambda_0 \ge 0 \tag{3}$$

$$\vec{\lambda} \cdot (\mathbf{x} - \mathbf{x}') \ge 1 \tag{4}$$

In general, the co-domain of ρ will be $\mathbb{Q}$, but to justify its use as a termination proof, one can convert it to a function with co-domain ω, namely $1 + \max(\rho(\mathbf{x}), 0)$.

The decision problem—does a *LRF* exist for a given linear-constraint loop? Will be denoted by either LINRF($\mathbb{Q}$) or LINRF($\mathbb{Z}$), depending on whether the variables are allowed to assume any rational value satisfying the constraints, or just integer ones. To the problem of computing an explicit representation of the ranking function, if one exists, we refer as the *synthesis* problem.

THEOREM 3.1. LINRF($\mathbb{Q}$) $\in$ *PTIME, and LRF synthesis can be done in polynomial time, for both SLC and MLC loops.*

[1]assuming that the variables range over the reals makes no difference from the rational case in the problems considered here.

This is perhaps the best-known result in this area. This result is based on using Farkas' lemma to transform the search for a *LRF* into a linear programming problem. Such a solution has been found by multiple researchers in different places and times and in some alternative versions: Sohn and van Gelder [23] did it (for *MLC* loops, restricted to non-negative integers) in termination analysis of logic programs; Feautrier [16], for *MLC* loops, for scheduling parallel computations; Podelski and Rybalchenko [21], for *SLC* loops, in termination analysis of imperative programs; finally, Mesnard and Serebrenik [18] extended Sohn and van Gelder's solution to the rationals. A related technique, in [11], is based on similar considerations but is not polynomial-time.

The fact that this solution is incomplete when the true domain of the variables is $\mathbb{Z}$ is illustrated by examples of loops which terminate (and have a *LRF*) over the integers, whereas over the rationals they are not even terminating. One such loop is

$$\begin{array}{l} \textit{while } (x_2 - x_1 \le 0, x_1 + x_2 \ge 1) \textit{ do} \\ \qquad\qquad x_2' = x_2 - 2x_1 + 1, x_1' = x_1 \end{array} \tag{5}$$

This discrepancy has been noted, for example in [9, 13, 16]; the latter two point out that the discrepancy disappears if all polyhedra in the program are integral. An integral polyhedron is equal to the convex hull of its integer points [22]. The reason that this works is, intuitively, *convexity*: a function that satisfies conditions (3,4) for a given set V of points $\mathbf{x}''$ also satisfies them throughout the covex hull of V. Hence, a complete and sound, but unfortunately exponential, algorithm to decide $\text{LinRF}(\mathbb{Z})$ is to compute the *integer hull* of $\mathcal{Q}$ (or each $\mathcal{Q}_i$), and proceed to solve $\text{LinRF}(\mathbb{Q})$. In [6], we undertook a closer study of the complexity of $\text{LinRF}(\mathbb{Z})$. The main results follow.

THEOREM 3.2. $\text{LinRF}(\mathbb{Z}) \in \text{coNP}$ *(referring to MLC loops); this problem is strongly coNP-hard, even for deterministic SLC loops.*

An outline of the proofs. *coNP-hardness* follows from the hardness of the problem: given a polyhedron (in constraint representation), does it contain any integer point? The reduction constructs a loop that has a *LRF* if and only if the given polyhedron has no integer points.

For *inclusion in coNP*, we have to show that *non-existence* of a *LRF* is a property for which there is a polynomially-verifiable witness. For simplicity, consider a *SLC* loop $\mathcal{Q}$. We start by observing that a single transition $\mathbf{x}'' \in \mathcal{Q}$ that does not satisfy (3,4) for a given vector $(\lambda_0,\vec{\lambda})$ suffices to show that $(\lambda_0,\vec{\lambda})$ are not the coefficients of a ranking function. We thus define $W(\mathbf{x}'')$ to be the set of coefficient vectors $(\lambda_0,\vec{\lambda}) \in \mathbb{Q}^{n+1}$ that do not satisfy (3,4); we say that they are *witnessed against* by $\mathbf{x}''$. It immediately follows that there is no *LRF* for $\mathcal{Q}$ if and only if $\bigcup_{\mathbf{x}'' \in \mathcal{Q}} W(\mathbf{x}'') = \mathbb{Q}^{n+1}$. But this is not an effective condition, since there may be infinitely integer points in $\mathcal{Q}$.

The goal now is to obtain a finite (and polynomial) witness set. To this end, we rely on convexity arguments, and on the *generator representation* of a polyhedron $\mathcal{Q}$. The generator representation consists of *vertices*, which are points of the polyhedron, and *rays*, also known as *recession directions*; a ray $\mathbf{y}'' \in \mathbb{Q}^{2n}$ is a vector such that for $\mathbf{x}'' \in \mathcal{Q}$, $\mathbf{x}'' + a\mathbf{y}''$ is in $\mathcal{Q}$ for all $a \ge 0$. The set of rays is denoted by $\mathcal{R}_\mathcal{Q}$. We call such a vector $\mathbf{y}''$ a *homogenous witness* (h-witness for short) against $(\lambda_0,\vec{\lambda})$ if one of the following requirements is not satisfied:

$$\vec{\lambda} \cdot \mathbf{y} \ge 0 \tag{6}$$

$$\vec{\lambda} \cdot (\mathbf{y} - \mathbf{y}') \ge 0 \tag{7}$$

It can be shown that in this case, $(\lambda_0,\vec{\lambda})$ cannot be the coefficient vector of a ranking function. Now, our witness against the existence of a *LRF* is a pair X, Y of sets, such that $X \subseteq \mathcal{Q}, Y \in \mathcal{R}_{\mathcal{Q}}$, $X \neq \emptyset$ and X, Y witness, all together, against every possible coefficient vector. To verify this we only need to verify that the witnesses are kosher (i.e., inclusion in $\mathcal{Q}$ and $\mathcal{R}_{\mathcal{Q}}$, respectively), and that there is no solution to the combined set of requirements (i.e., (3,4) for the members of X and (6,7) for members of Y). This is a polynomial-time procedure, if the size of X and Y (in bits) is polynomial. Fortunately, the existence of a polynomial-size witness set can be guaranteed; basically, we show that a polynomially big subset of the generators constitutes such a witness set.

Synthesis. By computing the generator representation of a polyhedron (if not available), we obtain an algorithm to find ranking functions—namely by finding a coefficient vector that none of the generators witnesses againt. Computation of the generators is, in general, of exponential time and space complexity. The resulting ranking-function coefficients are, however, of polynomial bit-size, a result based on the considerations involved in proving the inclusion of the decision problem in coNP. Interestingly, this sheds a new light on an algorithm proposed in [9], which searches for *LRF* coefficients by a kind of bisection search on the space of coefficients. Bounding the bit-size of the coefficients turns this search (which is unbounded, and hence only a semi-decision procedure) into a decision procedure.

Polynomially solvable cases. If coNP-hardness is considered as bad news, a positive news is that some special cases of interest can be solved in polynomial time. Basically, this happens when the transition polyhedra are either integral to begin with, or of such a kind that their integer hull is easy to compute. In [6] we characterize some benign cases, including loops in which the body is a sequence of linear affine updates with integer coefficients (as in loop (5) above) and the condition is defined by either an extended form of *difference constraints*, a restricted form of *Two Variables Per Inequality* constraints, or a cone (constraints where the free constant is zero). Some cases in which the body involves linear constraints are also presented. A somewhat surprising *hardness* result is that for octagonal guards [19, 7] and deterministic udpates, LinRF($\mathbb{Z}$) is already coNP-hard.

4 Lexicographic-Linear Ranking Functions

A *lexicographic-affine* ranking function (*LLRF*) has the form $\tau(\mathbf{x}) = \langle \rho_1(\mathbf{x}), \dots, \rho_d(\mathbf{x}) \rangle$ where each $\rho_i(\mathbf{x}) = \lambda^{(i)} \cdot \mathbf{x} + \lambda_0^{(i)}$. It has to descend in lexicographic order. The constant d is called the *dimension* of the function.

Lexicographic ranking functions are a very natural concept: the oldest example of a (manual) termination proof using ranking functions [25] uses lexicographic descent. Several automatic methods of finding *LLRF*s have appeared, and we could point out that any method that finds a *LRF* for each loop in a nested set of loops implicitly constructs a *LLRF*. An algorithm in the style of Terminator [14] can also arrive at a lexicographic ranking function, but implicitly and in a roundabout way.

Before discussing algorithms and complexity, there are some things to note, and first the very definition, as there are different variants of the concept. Our definition follows. We use the notation $\Delta\rho(\mathbf{x}'')$ for $\rho(\mathbf{x}) - \rho(\mathbf{x}')$.

Definition 4.1. Let $\tau = \langle \rho_1, \dots, \rho_d \rangle$. We say that τ is a *LLRF* for a set $T \subseteq \mathbb{Q}^{2n}$ of transitions if and only if for every $\mathbf{x}'' \in T$ there exists $i \le d$ for which the following hold:

$$\forall j < i \;.\; \Delta\rho_j(\mathbf{x}'') \ge 0 \tag{8}$$

$$\forall j \le i \;.\; \rho_j(\mathbf{x}) \ge 0 \tag{9}$$

$$\Delta\rho_i(\mathbf{x}'') \ge 1 \tag{10}$$

When this holds, we write $\tau(\mathbf{x}) \succ_{lex} \tau(\mathbf{x}'')$.

As for *LRFs*, the co-domain of the function is not really a well-founded set, but it can be easily converted to a function into ω^d in order to prove termination. We note that the definition of [3] is more restrictive since it requires (9) to hold for all $1 \le j \le d$. In contrast the definition in [8] is more general since it requires (9) to hold only for $j = i$. Examples can be given to show that these three classes of function are really different. In terms of complexity, [3] give a polynomial-time algorithm, based on repeated finding of *LRFs* using linear programming, whereas [8] use a (possibly exponential) combinatorial search. Our work in [5] is inspired by [3], but extends it in certain ways: most importantly, we consider the complexity of the problem when the variables are integer (that is, the set T of transitions is specified as the integer points in a set $\mathcal{Q}_1, \dots, \mathcal{Q}_k$ of transition polyhedra).

In fact, [3] includes a (quite clever) completeness proof, but as for previous *LRF* generation procedures, this completeness is for rational-valued state space. Even over the rationals, their solution is incomplete when referring to our more flexible class of functions. An interesting property of our class is that an *SLC* loop may require a lexicographic ranking function, as the following example demonstrates:

$$\textit{while}(x_1 \ge 0, x_2 \ge 0, x_3 \ge -x_1) \;\textit{do}\; x_2' = x_2 - x_1, x_3' = x_3 + x_1 - 2\,. \tag{11}$$

It has a *LLRF* $\tau_1 = \langle x_2, x_3 \rangle$ as in Definition 4.1 (over both rationals and integers), while according to the more restricted definition in [3], it has no *LLRF* at all.

As for linear ranking functions, we consider both the complexity of the decision problem over the integers (denoted LexLinRF($\mathbb{Z}$)), and the synthesis problem. Our main results are:

THEOREM 4.2. LexLinRF($\mathbb{Z}$) *is coNP-complete.*

THEOREM 4.3. *For synthesis of LLRFs, when variables range over the integers, there is a complete synthesis algorithm that runs in polynomial time,* given integer polyhedra *(otherwise, integer hulls have to be computed, which increases the running time to exponential.)*

An essential component in our proofs is the notion of a *quasi-LRF*. We say that ρ is a quasi-*LRF* for a set T of transitions if for every $\mathbf{x}'' \in T$ the following holds:

$$\rho(\mathbf{x}) \ge 0 \tag{12}$$

$$\Delta\rho(\mathbf{x}'') \ge 0 \tag{13}$$

We say that it is a *non-trivial* if, in addition, $\Delta\rho(\mathbf{x}'') > 0$, for at least one $\mathbf{x}'' \in X$.

Our synthesis algorithm constructs a *LLRF* $\langle \rho_1(\mathbf{x}), \dots, \rho_d(\mathbf{x}) \rangle$ in this way: first a non-trivial quasi-*LRF* ρ_1 is found; then the transitions where ρ_1 strictly descends are eliminated; if some are left, the algorithm is repeated to generate the next components. The search for a quasi-*LRF* uses linear programming, resembling the *LRF* procedures based on the Farkas lemma. The passage to the next iteration is based on the fact that the *set of transitions where ρ_1 does not descend, out of a transition polyhedron $\mathcal{Q}_i$, is either empty or a face of $\mathcal{Q}_i$*. This leads to an efficient algorithm to compute the polyhedra for the next iteration, keeping them integral, and not blowing up the size of the coefficients.

For the *decision problem*, our results rest on the following proposition, which in turn is proved (in part) using the algorithm:

PROPOSITION 4.4. *There is no LLRF for a set of transitions $T \subseteq \mathbb{Z}^{2n}$ if and only if there is $W \subseteq T$ for which there is no non-trivial quasi-LRF.*

Now, inclusion in coNP follows by showing that non-existence of a non-trivial quasi-*LRF* has a polynomially checkable witness. The considerations in proving this claim are similar to those outlines for *LRF*s, involving witnesses which are vertices and h-witnesses which are rays, though slightly more complicated since the witness has to rule out only non-trivial quasi-*LRF*s (a trivial quasi-*LRF*, namely $\rho(\mathbf{x}) = 0$, always exists, and there may be others).

Further observations. Two interesting results that followed from our investigation are these:

- The dimension d of our ranking functions is always at most n, for an *MLC* loop of any number of alternatives.
- If an *SLC* loop has such a ranking function, its number of iterations can be linearly bounded (more precisely, it is linear in the absolute values of the variables in the initial state) even if the dimension of the ranking function is larger than 1.

5 General Control-Flow Graphs

MLC loops can be extended to general CFGs annotated with linear constraints. This is the program abstraction used in [3, 18, 12, 23]. In [3], termination certificates for such programs consists of a lexicographic affine function ρ_ℓ associated with every program location ℓ, such that in a transition $s \vdash s'$ from location ℓ to ℓ', we have $\rho_\ell(s) \succ_{lex} \rho_{\ell'}(s')$. As their algorithm shows (and also some previous works), the generalization from finding a *LLRF* for an *MLC* loop to this model is quite smooth, and our results transfer in the same way.

A predecessor to [3], in a sense, is the algorithm in [12] (extending [11]). It is somewhat similar in structure to that of [3] (and also ours), finding linear quasi-ranking functions one by one, but it does not construct *LLRF*s explicitly, and its space of ranking functions is more restricted because it handles a strongly-connected component S of the program by looking for a ρ which is bounded and non-increasing throughout the component (that is, ρ_ℓ is the same for all $\ell \in S$). However, we may note that the last difference disappears if one considers only *MLC* loops.

References

[1] E. Albert, P. Arenas, S. Genaim, and G. Puebla. Closed-form upper bounds in static cost analysis. *Journal of Automated Reasoning*, 46(2):161–203, 2010.

[2] E. Albert, P. Arenas, S. Genaim, G. Puebla, and D. Zanardini. Costa: Design and implementation of a cost and termination analyzer for java bytecode. In F. S. de Boer, M. M. Bonsangue, S. Graf, and W. P. de Roever, editors, *Formal Methods for Components and Objects, FMCO'07*, volume 5382 of *LNCS*, pages 113–132. Springer, 2007.

[3] C. Alias, A. Darte, P. Feautrier, and L. Gonnord. Multi-dimensional rankings, program termination, and complexity bounds of flowchart programs. In R. Cousot and M. Martel, editors, *Static Analysis Symposium, SAS'10*, volume 6337 of *LNCS*, pages 117–133. Springer, 2010.

[4] R. Bagnara, F. Mesnard, A. Pescetti, and E. Zaffanella. A new look at the automatic synthesis of linear ranking functions. *Inf. Comput.*, 215:47–67, 2012.

[5] A. M. Ben-Amram and S. Genaim. The linear ranking problem for integer linear-constraint loops. Technical report, 2013.

[6] A. M. Ben-Amram and S. Genaim. On the linear ranking problem for integer linear-constraint loops. In *Proc. of the 40th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages, POPL'13*, pages 51–62, NY, USA, 2013. ACM.

[7] M. Bozga, C. Gîrlea, and R. Iosif. Iterating octagons. In S. Kowaleski and A. Philippou, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 5505 of *LNCS*, pages 337–351. Springer, 2009.

[8] A. R. Bradley, Z. Manna, and H. B. Sipma. Linear ranking with reachability. In K. Etessami and S. K. Rajamani, editors, *Computer Aided Verification, CAV'05*, volume 3576 of *LNCS*, pages 491–504. Springer, 2005.

[9] A. R. Bradley, Z. Manna, and H. B. Sipma. Termination analysis of integer linear loops. In M. Abadi and L. de Alfaro, editors, *Concurrency Theory, CONCUR 2005*, volume 3653 of *LNCS*, pages 488–502. Springer, 2005.

[10] M. Codish and C. Taboch. A semantic basis for termination analysis of logic programs. *The Journal of Logic Programming*, 41(1):103–123, 1999. Preliminary (conference) version in LNCS 1298 (1997).

[11] M. Colón and H. Sipma. Synthesis of linear ranking functions. In *7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2031 of *LNCS*, pages 67–81. Springer, 2001.

[12] M. Colón and H. Sipma. Practical methods for proving program termination. In *14th International Conference on Computer Aided Verification (CAV)*, volume 2404 of *LNCS*, pages 442–454. Springer, 2002.

[13] B. Cook, D. Kroening, P. Rümmer, and C. M. Wintersteiger. Ranking function synthesis for bit-vector relations. In J. Esparza and R. Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS'10*, volume 6015 of *LNCS*, pages 236–250. Springer, 2010.

[14] B. Cook, A. Podelski, and A. Rybalchenko. Termination proofs for systems code. In M. I. Schwartzbach and T. Ball, editors, *Programming Language Design and Implementation, PLDI'06*, pages 415–426. ACM, 2006.

[15] N. Dershowitz, N. Lindenstrauss, Y. Sagiv, and A. Serebrenik. A general framework for automatic termination analysis of logic programs. *Applicable Algebra in Engineering, Communication and Computing*, 12(1–2):117–156, 2001.

[16] P. Feautrier. Some efficient solutions to the affine scheduling problem. I. one-dimensional time. *International Journal of Parallel Programming*, 21(5):313–347, 1992.

[17] P. Feautrier. Some efficient solutions to the affine scheduling problem. II. multidimensional time. *International Journal of Parallel Programming*, 21(6):389–420, 1992.

[18] F. Mesnard and A. Serebrenik. Recurrence with affine level mappings is p-time decidable for clp(r). *TPLP*, 8(1):111–119, 2008.

[19] A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, March 2006.

[20] R. Peña and A. D. Delgado-Muñoz. Size invariant and ranking function synthesis in a functional language. In *Functional and Constraint Logic Programming*, volume 6816 of *LNCS*, pages 52–67. Springer, 2011.

[21] A. Podelski and A. Rybalchenko. A complete method for the synthesis of linear ranking functions. In B. Steffen and G. Levi, editors, *Verification, Model Checking, and Abstract Interpretation, VMCAI'04*, volume 2937 of *LNCS*, pages 239–251. Springer, 2004.

[22] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, New York, 1986.

[23] K. Sohn and A. V. Gelder. Termination detection in logic programs using argument sizes. In D. J. Rosenkrantz, editor, *Symposium on Principles of Database Systems*, pages 216–226. ACM Press, 1991.

[24] F. Spoto, F. Mesnard, and E. Payet. A termination analyzer for Java bytecode based on path-length. *ACM Trans. Program. Lang. Syst.*, 32(3):1–70, 2010.

[25] A. M. Turing. Checking a large routine. In *Report of a Conference on High Speed Automatic Calculating Machines*, pages 67–69, 1948. reprinted in: The early British computer conferences, vol. 14 of Charles Babbage Institute Reprint Series For The History Of Computing, MIT Press, 1989.