A Complexity Tradeoff in Ranking-Function Termination Proofs

Amir M. Ben-Amram*

April 25, 2008

Abstract

To prove that a program terminates, we can employ a ranking function argument, where program states are ranked so that every transition decreases the rank. Alternatively, we can use a set of ranking functions with the property that every cycle in the program's flow-chart can be ranked with one of the functions. This "local" approach has gained interest recently on the grounds that local ranking functions would be simpler and easier to find. The current study is aimed at better understanding the tradeoffs involved, in a precise quantitative sense. We concentrate on a convenient setting, Size-Change Termination framework (SCT). In SCT, programs are replaced by an abstraction whose termination is decidable. Moreover, sufficient classes of ranking functions (both global and local) are known. Our results show a tradeoff: either exponentially many local functions of certain simple forms, or an exponentially complex global function may be required for proving termination.

1 Introduction (informal)

The oldest trick in the book of termination proofs for programs (e.g., [19]) is the *ranking function proof*. In this method, we find a function ρ that maps program states into a well-founded ordered set, such that $\rho(s) > \rho(s')$ whenever s' is a state reachable from state s.

Let us call such a ranking function *global*. Recently, there has been growing interest in the use of *local* ranking functions: briefly, instead of a single

*Email: amirben@mta.ac.il Mailing address: School of Computer Science, Tel-Aviv Academic College, 4 Antokolski Str., 64044 Tel-Aviv, Israel. Phone: $+972-3-5211852$ Fax:$+972-3-5211871$

ranking function we have a finite set of functions, such that every possible cycle—a closed path in the control flow graph—decreases *at least one* of them (others may even increase). This idea has already been described and justified more than once, in different contexts [7, 16]. Codish, Lagoon and Stuckey [4] write:

> The main advantage in applying local ranking functions is that they take a simpler form than corresponding global ranking functions and are easy to find automatically.

The "simpler form" that a local ranking function takes turns out to be, in the case of [4], a linear combination of program variables. This is perhaps not so surprising as linear ranking functions for proving termination of loops are quite ubiquitous. The termination checker described in [6] maintains a set of such functions and iterates a procedure that extends it. Briefly, the procedure checks whether all cycles are covered by the current set of functions; otherwise, a cycle is found whose termination is unproven and a new ranking function is constructed for it.

The quotation from [4] above suggests a tradeoff: *complexity* of the global ranking function[1] versus the *quantity* of simple, local ranking functions. The purpose of this work is to investigate this tradeoff, with the goal of better understanding the design space of termination analyses.

The approach taken here to is to conduct a case study in a very simple setting. The setting is a type of abstract programs called *single-function, strict SCT instances* (1SSCT) with the natural numbers $\mathbb{N}$ as data. The SCT abstraction was defined in [12] and proved useful for termination analysis in many contexts ([13, 5, 7, 9, 10, 18, 1] and others)[2]. A particular impetus for this research was the recent discovery by Lee [11] of a way to express SCT termination proofs in the form of a global ranking function argument; before this work, only a connection to the local approach was known, elaborated by Codish, Lagoon and Stuckey [4]. Together these two works suggest SCT programs as a case for investigating the local-vs-global tradeoff.

In the following section, precise definitions are given to make this article rigorous and self-contained.

Section 3 investigates linear ranking functions. It is shown that *within our case study framework* general linear combinations are not more powerful

[1] not computational complexity, but the complexity of expressing the function, and perhaps that of verifying it

[2] The Logic Programming termination analyses cited here predate the introduction of SCT by Lee et al. However the "monotonicity constraints" they used are very much alike the SCT abstraction. [4] examines the subtle difference.

than simple sums of variables (combinations with 0/1 coefficients); and that instances that require an exponential number of linear ranking functions exist.

Section 4 investigates alternative ranking functions: minimum and maximum of a subset of the variables. This is motivated by the observation that both types of functions also provide completeness for 1SSCT programs and, in fact, beat the linear ones on the examples discussed in Section 3.

Finally, Section 5 turns to global ranking functions and discusses the conjecture that such functions for 1SSCT programs may need to be exponentially complex.

2 Introduction (formal)

This section gives the definitions necessary for the rest of the paper. For a fuller exposition and examples please refer to [12] and other publications on SCT.

2.1 SCT instances and their transition systems

The SCT abstraction of a program is a transition system with states of the form $(\mathtt{f}, \vec{v})$, where $\mathtt{f}$ represents a *program point* (e.g., a basic-block label in an imperative program, or a function name in a functional program) and $\vec{v}$ represents the program's store or environment. The notations we use for SCT evolved from the use of a functional language (though this is not an essential choice!) and hence some of the terminology.

Definition 2.1. *A* control flow graph *is a directed multigraph* $(Fun, Calls)$. *The nodes are referred to as* function names *and include an* initial node $\mathtt{f}_0$. *The arcs are known as calls. Every function* $\mathtt{f} \in Fun$ *has an* arity $arity(\mathtt{f}) \geq 0$, *defining the number of its* parameters. *The latter may be denoted by* $x_1, \ldots, x_{arity(\mathtt{f})}$.

Definition 2.2. *Let* $\mathtt{f}, \mathtt{g} \in Fun$. *A* size-change graph *from* $\mathtt{f}$ *to* $\mathtt{g}$, *written* $G : \mathtt{f} \to \mathtt{g}$, *is a bipartite graph with source nodes* $x_1, \ldots, x_{arity(\mathtt{f})}$, *target nodes* $x'_1, \ldots, \ldots, x'_{arity(\mathtt{g})}$ *and arcs labeled with either* $\downarrow$ *or* $\overline{\downarrow}$.

An arc with a $\downarrow$ label is called *strict*. Note that it is common to mark target parameter names with primes (x'_1). For notational convenience, we may also apply the prime to a set of parameters: $\{x_1, x_4\}'$ is $\{x'_1, x'_4\}$.

Definition 2.3. *Let* $\mathrm{CFG} = (Fun, Calls)$ *be a control-flow graph. An* SCT instance $\mathcal{G}$ *over* CFG *is a set of size-change graphs* G_c*, where* c *ranges over* $Calls$*, and* G_c *has source and target functions as specified by* c*.*

Observe that the CFG is implicitly specified by $\mathcal{G}$.

We treat $\mathcal{G}$ as specifying an infinite-state transition system, intended as a conservative abstraction of the subject program (thus, the transitions should be a superset of the transitions possible in actual program computations).

Definition 2.4. *A* state *is a pair* $(\mathtt{f}, \vec{v})$*, where* $\mathtt{f} \in Fun$ *and* $\vec{v} \in \mathbb{N}^{arity(\mathtt{f})}$*. The set of all states is* St*.*

A state transition *is a pair* (s, s') *of states. For a size-change graph* $G : \mathtt{f} \to \mathtt{g}$ *associated with call* c *and states* $s = (\mathtt{f}, \vec{v})$*,* $s' = (\mathtt{g}, \vec{u})$*, we write* $G \models s \mapsto s'$ *if for every arc* $x_i \xrightarrow{r} x'_j$ *in* G*,* $u_j \leq v_i$*, where if* $r = \downarrow$ *then* $u_j < v_i$*.*

The transition system associated with $\mathcal{G}$*,* $T_{\mathcal{G}}$*, has state space* St *and transitions*

$$\{(s, s') \mid G \models s \mapsto s' \text{ for some } G_c \in \mathcal{G}\}.$$

2.2 Termination and the SCT condition

Definition 2.5. *Let* $\mathcal{G}$ *be an SCT instance. A* run *of* $T_{\mathcal{G}}$ *is a (finite or infinite) sequence of states* $s_0, s_1, s_2 \ldots$ *such that for all* i*, there is some* $G_{c_i} \in \mathcal{G}$ *such that* $G_{c_i} \models s_i \mapsto s_{i+1}$*.*

Definition 2.6. *Transition system* $T_{\mathcal{G}}$ *is* terminating *if it has no infinite run beginning at* $\mathtt{f}_0$*.*

The point of SCT is to determine if $T_{\mathcal{G}}$ is terminating by analyzing $\mathcal{G}$.

Definition 2.7. *Let* $\mathcal{G}$ *be an SCT instance.*

1. *A* $\mathcal{G}$-multipath *is a (finite or infinite) sequence* $M = G_1 G_2 \ldots$ *with* $G_i \in \mathcal{G}$*, such that for all* i*, the target function of* G_i *is the source function of* G_{i+1}*. It is convenient to identify a multipath with the (finite or infinite)* layered directed graph *obtained by identifying the target nodes of* G_i *with the source nodes of* G_{i+1}*.*

2. *A* thread *in* M *is a (finite or infinite) directed path in* M *(when viewed as a single graph). A thread is* complete *if it spans the length of* M*.*

3. *A thread is* descending *if it has at least one arc with* $\downarrow$*. It is* infinitely descending *if the set of such arcs it contains is infinite.*

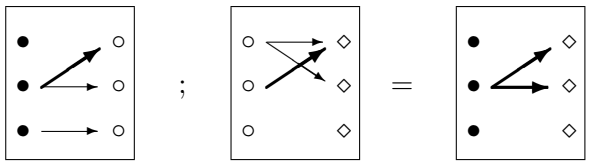

Figure 1: Size-change graph composition. Heavy arcs are strict.

Definition 2.8. *An SCT instance $\mathcal{G}$ is a* positive instance *(or, $\mathcal{G}$ satisfies SCT) if every infinite $\mathcal{G}$-multipath, beginning at $\mathtt{f}_0$, contains an infinitely-descending thread. A single size-change graph G is said to satisfy SCT if $\mathcal{G} = \{G\}$ does.*

THEOREM 2.9 (SCT main theorem). *$T_{\mathcal{G}}$ is terminating if and only if $\mathcal{G}$ satisfies SCT.*

The "if" part of this theorem (soundness of the SCT criterion) is proven in [12]. For the "only if" direction (completeness) see [11].

2.3 The Composition-Closure Algorithm

An important property of SCT is that it can be decided by testing every graph in a finite set—the *composition closure* of $\mathcal{G}$. This decision procedure stems from [17] and we freely mix Sagiv's definitions with newer SCT terminology following [12].

Definition 2.10. *The* composition *of two size-change graphs $G : \mathtt{f} \to \mathtt{g}$ and $G' : \mathtt{g} \to \mathtt{h}$ is $G;G' : \mathtt{f} \to \mathtt{h}$ with arc set E defined as follows. For a pair of parameters $\mathtt{x}, \mathtt{y}$, if multipath GG' has a descending thread from $\mathtt{x}$ to $\mathtt{y}$, then E contains an arc $\mathtt{x} \stackrel{\downarrow}{\to} \mathtt{y}$. Otherwise, if GG' has any thread from $\mathtt{x}$ to $\mathtt{y}$, E contains $\mathtt{x} \stackrel{\Vdots}{\to} \mathtt{y}$. Otherwise, there is no arc from $\mathtt{x}$ to $\mathtt{y}$.*

Composition has the following easy-to-prove property:

LEMMA 2.11. *Let c, c' be two calls such that the target of c is the source of c'. Let G_c, $G_{c'}$ be corresponding size-change graphs. For any states s, s', s'' such that $G_c \models s \mapsto s'$ and $G_{c'} \models s' \mapsto s''$ we have $G_c; G_{c'} \models s \mapsto s''$.*

Composition is illustrated by the diagrams in Figure 1.
We denote the composition closure of $\mathcal{G}$ by $\mathcal{G}^+$.

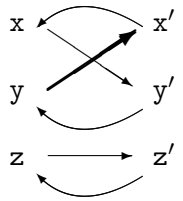

Figure 2: The circular variant of a size-change graph.

Definition 2.12. *For a size-change graph G, the* circular variant *G' of G is a directed graph obtained by adding, for every parameter x_i, a* backward arc *$x'_i \to x_i$ (Figure 2).*

THEOREM 2.13 (after [17]). *$\mathcal{G}$ satisfies SCT if and only if for every $G \in \mathcal{G}^+$, there is a directed cycle in G' that includes a strict arc.*

The closure $\mathcal{G}^+$ can be computed in finite time by a trite procedure. Testing each graph for the property indicated in the theorem is straightforward. It follows that SCT is decidable. This is not a contradiction to its soundness and completeness, since the transition systems described by SCT instances are of a special type and do not represent all programs.

2.4 Ranking functions for SCT programs

Definition 2.14. *Let $P(s, s')$ be any predicate defined over pairs of states. For a size-change graph $G \in \mathcal{G}$ we write $G \models P(s, s')$ if*

$$(G \models s \mapsto s') \Rightarrow P(s, s').$$

Definition 2.15. *Given a size-change graph G, a (local) ranking function for G is a function $\rho : St \to W$, where W is a well-ordered set, such that $G \models \rho(s) > \rho(s')$. Given an SCT instance $\mathcal{G}$, a global ranking function for $\mathcal{G}$ is a function ρ that satisfies $G \models \rho(s) > \rho(s')$ for every $G \in \mathcal{G}$.*

In this paper, the codomain of all global ranking functions will be $\mathbb{N}$ (with the standard order).

Clearly, if a global ranking function exists, then $T_{\mathcal{G}}$ terminates; it is also possible to prove termination using a finite number of local ranking functions (see [16]). It can easily be seen that it suffices to look for local ranking functions for that are "loops," i.e., transition sequences that begin and end at the same program point (function name). Thus the local approach is to look for a finite set of ranking functions that covers all the loops.

3 Linear Ranking functions for SCT instances

As already mentioned, a central claim of [4] is the sufficiency of linear ranking functions of a simple form for all SCT instances. Here is a precise statement.

THEOREM 3.1 (Codish-Lagoon-Stuckey). *Let $\mathcal{G}$ be a positive SCT instance with a single function and parameters $x_1, \ldots, x_n$, and let $G \in \mathcal{G}^+$. Then there is a ranking function for G of the form $\rho(x_1, \ldots, x_n) = \sum a_i x_i$ with all coefficients $a_i \in \{0, 1\}$.*

The theorem implies a certain completeness result for termination checkers based on linear ranking functions (such as [6]). I.e., this strategy is sufficient for handling all SCT programs. It also shows that at most 2^n different ranking functions may ever be needed to handle a single-function SCT program.

Though this theorem is from [4], let us give a proof, as it clarifies the connection between size-change graphs and linear combinations.

Proof of Theorem 3.1. Let $G \in \mathcal{G}^+$. We assume that there is a directed cycle in G', including a strict arc. Now, if there is any cycle, there is a simple one, so we can concentrate on simple cycles. Such a cycle has the form:

$$x_{i_0} \to x'_{i_1} \to x_{i_1} \to x'_{i_2} \to \cdots \to x'_{i_k} \to x_{i_0},$$

where $i_k = i_0$ but $i_1, \ldots, i_k$ are all different. Let $S = \{x_{i_1}, \ldots, x_{i_k}\}$. Thus $S' = \{x'_{i_1}, \ldots, x'_{i_k}\}$. The cycle consists of a perfect matching in G between S and S', augmented with backward arcs (consider Figure 2: the cycle is $\mathtt{x} \to \mathtt{y}' \to \mathtt{y} \to \mathtt{x}' \to \mathtt{x}$. The matching is $\{\mathtt{x} \to \mathtt{y}', \mathtt{y} \to \mathtt{x}'\}$).

It clearly follows that for all j, $G \models x_{i_j} \geq x'_{i_{j+1}}$ (where i_{k+1} means i_0) and since at least one of the forward arcs is strict, for some j we have $G \models x_{i_j} > x'_{i_{j+1}}$. Hence, $G \models \sum_j x_{i_j} > \sum_j x'_{i_j}$. □

A set S for which there exists a perfect matching to S' will be said to be *matched* in G.

We next show that, in some sense, the above is the *only way* in which a size-change graph can imply a linear ranking function. However one has to take into account that a sum of ranking functions is also a ranking function.

Definition 3.2. *Let $G : \mathtt{f} \to \mathtt{f}$ be a size-change graph that satisfies SCT. A function $\rho(x_1, \ldots, x_n) = \sum a_i x_i$, with $a_i \in \mathbb{N}$ is* simple *if $\rho = \sum_{i \in S} x_i$ where S is matched in G.*

Definition 3.3. *Let $G : \mathtt{f} \to \mathtt{f}$ be a size-change graph. A function $\rho(s)$ is a* quasi-ranking function *for G if $G \models \rho(s) \geq \rho(s')$.*

LEMMA 3.4. *Let $G : \mathtt{f} \to \mathtt{f}$ be a size-change graph that satisfies SCT. Assume that the function $\rho(x_1, \ldots, x_n) = \sum a_i x_i$, with $a_i \in \mathbb{N}$, is a ranking function for G; i.e., $G \models \rho(x_1, \ldots, x_n) > \rho(x'_1, \ldots, x'_n)$. Then ρ is the sum of one or more simple quasi-ranking functions, at least one of which is ranking.*

Proof. Consider the digraph $G' = (V, E)$ with $V = X \cup X'$, where $X = \{x_1, \ldots, x_n\}$.

Let us consider this graph as a flow network [15] with unlimited capacities on all arcs in E (but zero capacity elsewhere). We will show that it is possible to assign a flow value (or "weight") $w_{ij} \geq 0$ (always zero if $(x_i, x'_j) \notin E$), obtaining a valid circulation (i.e., flow is conserved at every node), such that

$$\forall i : \quad \sum_{j=1}^{n} w_{ij} = \sum_{j=1}^{n} w_{ji} = a_i. \tag{1}$$

That is, the aggregate flow out of node x_i, as well as into x'_i, is a_i.

To achieve a valid circulation, the flow on a backward arc $x'_i \to x_i$ must clearly be a_i. It rests to assign flows to the forward arcs. To this end, represent each size-change arc $x_i \to x'_j$ as the linear constraint:

$$x_i - x'_j \geq b_{ij}$$

where b_{ij} is 1 if the arc is strict and 0 otherwise. We get a system of linear constraints in $2n$ variables, one for each arc. To simplify notations we may add trivial rows ($0 \geq 0$) for pairs (i, j) where there are no arcs, as to have exactly n^2 constraints. If we further add the constraint:

$$-\sum_{i=1}^{n} a_i x_i + \sum_{j=1}^{n} a_j x'_j \geq 0 \tag{2}$$

we obtain a system of the form $A\mathbf{x} \geq \mathbf{b}$, with $\mathbf{x} = (x_1, \ldots, x_n, x'_1, \ldots, x'_n)$, $\mathbf{b}$ containing 1 for the rows that represent strict arcs, while A contains a $+1$ and a -1 in each row that represents an arc, and the coefficients of (2) in the last row. Here is an example of such a system (with $n = 2$), just to illustrate its form:

$$\begin{bmatrix} 1 & 0 & -1 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -a_1 & -a_2 & a_1 & a_2 \end{bmatrix} \cdot \mathbf{x} \geq \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

The system is feasible if and only if ρ is *not* a ranking function (constraint (2) indicates that ρ does not decrease, while the other constraints ensure a valid transition). Since we assumed that it is, the system must be infeasible, and by Farkas' lemma there is a vector of $n^2 + 1$ weights $\vec{w} \geq 0$, such that

$$A^T \vec{w} = 0 \qquad \text{and} \qquad \mathbf{b}^T \vec{w} > 0.$$

The weight of the last row cannot be zero (for without it, the system is clearly feasible) and by normalizing it to 1 we obtain weights that satisfy (1).

It is also important to note that at least one strict arc must have a nonzero weight. The weights are rational, but not necessarily integral.

Next, we show that it is possible to round the weights to integers while preserving (1). First, we introduce *capacities* for this flow network. The capacities are the current flows, rounded up to integers. Let D be a common denominator for the weights. If they are not all integral, apply the following procedure:

1. Are the weights of strict arcs integral?
   (a) If they are, let e be a (non-strict) arc such that the difference $\lceil w_e \rceil - w_e$ is smallest among the nonzero differences.
   (b) Otherwise choose e in the same fashion among strict arcs.
2. Note that e has a nonzero residual capacity. Suppose that e is $u \to v$; there must be an arc leaving v with a non-integral residual capacity. Continue in this way until a cycle consisting of arcs with non-integral residual capacities is obtained. Now, it is clearly possible to increase the flow on this cycle by $1/D$.

Repeating the procedure sufficiently many times we can get an integral flow which preserves (1) as well as the property that at least one strict arc has nonzero flow.

A procedure similar to the above can be used to decompose the flow into a set of simple cycles of flow 1. These cycles give rise to simple quasi-ranking functions. For those that contain a strict arc, the functions are ranking. □

COROLLARY 3.5. *Whenever a linear ranking function exists for a size-change graph, there is a simple one.*

We can now move to our goal, a lower bound on the number of ranking functions that may be necessary in a termination proof, if we restrict ourselves to assigning a linear ranking function to each control-flow cycle.

The following lower bounds are given by explicit construction of SCT instances. It turns out that they can be obtained by instances of a restricted form; this only strengthens the lower-bound result. The restrictions are two: the control-flow graph shrinks to a single node; and all size-change arcs are strict. Though this may not be obvious, the latter restriction is the more significant; single-node flow-graphs encompass most of the complexity of the SCT problem.

Definition 3.6. *A* 1SSCT *instance is an SCT instance where there is a single function name,* `f` *(which is also the initial function), and all size-change arcs are strict. The name* 1SSCT *also stands for the set of positive* 1SSCT *instances.*

Next, we formulate and prove a lower bound.

Definition 3.7. *A set $\mathcal{R}$ is a* sufficient set of ranking functions *for SCT instance $\mathcal{G}$ if for every $G \in \mathcal{G}^+$ there is $\rho \in \mathcal{R}$ that is a local ranking function for G.*

THEOREM 3.8. *There is a 1SSCT instance $\mathcal{G}$ with n parameters and $n+1$ size-change graphs such that any sufficient set of linear ranking functions for $\mathcal{G}$ contains at least $2^n - 1$ functions.*

Note that the lower bound is tight since $2^n - 1$ is the number of (non-constant) functions of the form of a simple ranking function. Thus, the instance claimed in the theorem requires the use of all possible simple ranking functions!

The theorem is proved by presenting the construction of $\mathcal{G}$ and proving a property that implies the result.

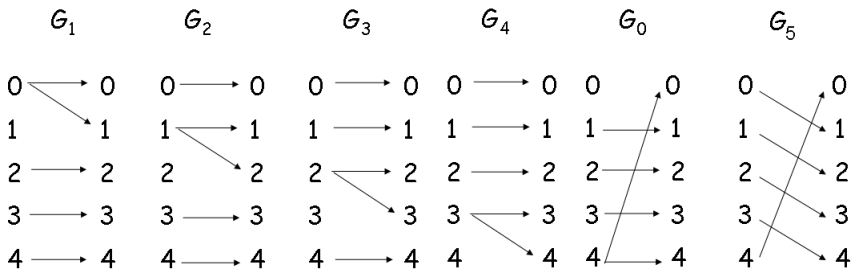

Figure 3: Construction 3.1, with $n = 5$.

Notation. In the constructions, it is convenient to refer to parameters by their indices, e.g., 1 instead of x_1, where context permits. In particular, arcs will be written as $1 \to 2$ instead of $x_1 \to x_2$. Moreover the indices are not necessarily 1 through n, but chosen for convenience of the construction.

Construction 3.1: 1SSCT instance $\mathcal{G}$.

In this construction we index the parameters by the elements of the cyclic group $\mathbb{Z}_n$. For $k = 0, \ldots, n-1$ let

$$G_k = \{i \to i \mid i \neq k\} \cup \{k-1 \to k\}$$

and let

$$G_n = \{i \to i+1 \mid 0 \leq i < n\},$$

where expressions $i+1$, $k-1$ are taken modulo n.

Set $\mathcal{G} = \{G_0, \ldots, G_{n-1}, G_n\}$.

LEMMA 3.9. *For every nonempty set $S = \{s_1, \ldots, s_t\} \subseteq \mathbb{Z}_n$ there is a graph $G_S \in \mathcal{G}^+$ that has S and only S as a matched set.*

Proof. Assume, for simplicity, that $0 \in S$ (there is no loss of generality, since S is nonempty and since the indices can be cyclically permuted for convenience). Let $i_1 < i_2 \cdots < i_s$ be the elements of $\mathbb{Z}_n \setminus S$ in increasing order. Define

$$G_S = G_{i_1}; \cdots; G_{i_s}; G_n$$

Consider the multipath $M_S = G_{i_1} \cdots G_{i_s}$ (an empty multipath if $S = \mathbb{Z}_n$). The reader may verify that for every $1 \leq j \leq s$, there is no complete thread

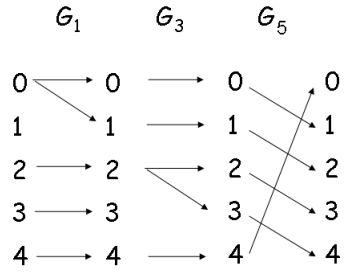

Figure 4: The multipath $M_S; G_n$ for $n = 5$ and $S = \{0, 2, 4\}$.

in M_S beginning at i_j. Thus, G_S has no arc from i_j, which shows that if T is a matched set for G_S, then $T \subseteq S$. Next, for an arbitrary $s \in S$, it is not hard to see that G_S has just a single arc from s into S, namely $s \to \mathrm{succ}(s)'$ where $\mathrm{succ}(s)$ is the first element of S after s (in cyclic order of $\mathbb{Z}_n$). It follows that T must be equal to S. Figure 4 gives an example. □

Proof of Theorem 3.8. $\mathcal{G}$ above satisfies SCT; this is easy to verify. By the last lemma, for each of the $2^n - 1$ different nonempty subsets S of the parameters, there is a size-change graph $G_S \in \mathcal{G}^+$ that has S and only S as a matched set; therefore (by Lemma 3.4), G_S has only $\sum_{x_i \in S} x_i$ (up to a constant factor) as a linear ranking function. The conclusion follows. □

Remark: It is easy to see (and important for the next section) that another example of the same complexity is $\mathcal{G}^t$, the transposition of $\mathcal{G}$ (i.e., exchange the source and target sides of all graphs).

4 Max and Min as ranking functions

The example in the above proof is revealing in that, as the alert reader may have discovered, there is actually a simple ranking function that fits all graphs in $\mathcal{G}^+$, namely: $\max(x_0, \ldots, x_{n-1})$. Similarly, for $\mathcal{G}^t$ we can use $\min(x_0, \ldots, x_{n-1})$.

Conclusion: Even if the restriction to linear ranking functions is sufficient to handle a program, it may well be rewarding (in efficiency) to extend

the ranking function component of a termination analyzer to allow functions like min and max.

In fact, each of these types of ranking function suffices on its own for handling all strict SCT instances—just repeat the proof of Theorem 3.1. Thus, our complexity question applies to such functions as well: how many max (or min) ranking functions may be necessary, in the worst case, for handling a strict SCT instance? We next answer this question.

LEMMA 4.1. *Let $G : \mathtt{f} \to \mathtt{f}$ be a size-change graph that satisfies SCT. Assume that the function $\rho(x_1, \ldots, x_n) = \max_{x_i \in S} x_i$ is a ranking function for G. Then S is self-surjective in G, i.e., $\forall y \in S. \exists x \in S. (x \to y) \in G$.*

Proof. Assume the contrary. Thus, there is $y \in S$ such that no $x \in S$ has arc $x \to y$ in G. It follows that in a state transition modeled by G it is possible to assign to y a value not bounded by any of the values previously in S, so that $\max S$ rises. □

A similar result holds for set minima:

LEMMA 4.2. *Let $G : \mathtt{f} \to \mathtt{f}$ be a size-change graph that satisfies SCT. Assume that the function $\rho(x_1, \ldots, x_n) = \min_{x_i \in S} x_i$ is a ranking function for G. Then S is thread-preserving in G, i.e., $\forall x \in S. \exists y \in S. (x \to y) \in G$.*

We now move to a lower bound.

THEOREM 4.3. *There is a 1SSCT instance $\mathcal{G}$ with n parameters and $n+1$ size-change graphs such that any sufficient set of* **max** *ranking functions for $\mathcal{G}$ contains at least $2^n - 1$ functions. A similar result holds for* **min**.

Proof. The example for **max** is exactly $\mathcal{G}^t$ with $\mathcal{G}$ as in Section 3. For every graph G_S described there, S is the one and only self-surjective nonempty set. For **min**, the example is naturally $\mathcal{G}$ itself. For every graph G_S, S is the one and only thread-preserving nonempty set. □

It is natural to ask what happens if we allow all three types—sums, maxima and minima of sets of parameters—to be mixed in a single termination proof (different functions for different loops). The next construction proves that an exponential lower bound still holds.

Construction 4.1: 1SSCT instance $\mathcal{H}$.

For any set of parameters P let I_P be the size-change graph consisting of arcs $x \to x$ for all $x \in P$. For notational simplicity, I_n means I_P with $P = \mathbb{Z}_n$.

Let G_A, G_B be size-change graphs over parameter sets A and B, respectively. We define $G_A \otimes G_B$ to be a graph over parameter set $A \times B$ that has arc: $(i,j) \to (i',j')$ whenever $(i \to i') \in G_A$ and $(j \to j') \in G_B$.

For sets of graphs $\mathcal{G}_A$ and $\mathcal{G}_B$, we define

$$\mathcal{G}_A \otimes \mathcal{G}_B = \{G \otimes I_B \mid G \in \mathcal{G}_A\} \cup \{I_A \otimes G \mid G \in \mathcal{G}_B\}.$$

For sets $C \subseteq A \times B$, let $\pi_1 C$ (respectively $\pi_2 C$) denote the first (second) projection of C.

Let $\mathcal{H} = \mathcal{G} \otimes \mathcal{G}^t$, where $\mathcal{G}$ is given by Construction 3.1. Note that the parameter names in this instance are the elements of $P = \mathbb{Z}_n \times \mathbb{Z}_n$.

LEMMA 4.4. *For every pair of nonempty sets $S, T \subseteq \mathbb{Z}_n$ there is a graph $H_{ST} \in \mathcal{H}^+$ such that:*
(i) if $A \subseteq P$ is nonempty and thread-preserving in H_{ST} then $\pi_1 A = S$.
(ii) if $B \subseteq P$ is nonempty and self-surjective in H_{ST} then $\pi_2 B = T$.

Proof. Assume w.l.o.g. that $0 \in S, T$; let $i_1 < i_2 \cdots < i_s$ be the elements of $\mathbb{Z}_n \setminus S$, and define

$$H_S = (G_{i_1} \otimes I_n); \cdots ; (G_{i_s} \otimes I_n); (G_n \otimes I_n).$$

Observe that $H_S = G_S \otimes I_n$ with G_S as in Lemma 3.9. In a similar way we obtain $H_T = I_n \otimes G_T^t$. We let $H_{ST} = H_S; H_T$.

Suppose that $A \neq \emptyset$ is a thread preserver in H_{ST}. Let $(i,j) \in A$. Then there is an arc $(i,j) \to (i',j')$ in H_{ST} with $(i',j') \in A$. If there is an arc of H_T into (i',j'), it has the form $(i',j'') \to (i',j')$ for some j'' (determined by G_T). It follows that H_S must have an arc $(i,j) \to (i',j'')$, which by the construction of H_S entails $(i \to i') \in G_S$. Also, $i' \in \pi_1 A$; we conclude that $\pi_1 A$ is thread preserving in G_S, and this implies $\pi_1 A = S$, concluding the proof of (i). Claim (ii) is clearly symmetric. □

THEOREM 4.5. *There is a 1SSCT instance $\mathcal{H}$ with n^2 parameters and $2n+2$ size-change graphs such that any sufficient set of ranking functions for $\mathcal{H}$, where each function is either linear, or is a maximum or minimum over a set of parameters, contains at least $2^n - 1$ functions.*

Proof. Consider the $(2^n-1)^2$ pairs of nonempty sets (S,T) and the graphs H_{ST} provided by the above lemma. Every function of the type $\min_{(i,j)\in A} x_{(i,j)}$ can rank at most $2^n - 1$ such size-change graphs, namely those where $S = \pi_1 A$. Similarly, every function of the type $\max_{(i,j)\in B} x_{(i,j)}$ can rank at most $2^n - 1$ such size-change graphs, namely those where $T = \pi_2 B$. Every linear function can also rank at most $2^n - 1$ graphs, which can be argued

in the manner of Theorem 3.8. Thus, at least $2^n - 1$ different functions are necessary to rank all loops. □

Multiset orderings. In previous work on termination, it has been discovered that a useful operator for constructing ranking functions is the *multiset-forming operator*. Applying this operator to parameters $x_1, \ldots, x_k$ yields the multiset of values of the parameters. For use in ranking functions, multisets are endowed with an ordering; [8] introduced *multiset ordering* and [3] introduced *the multiset ordering dual*. In the context of our case-study, these additions are not significant because when an *all-strict* size-change graph implies descent in multiset ordering, then the maximum of the parameters' values descends, and descent in the dual ordering implies descent of the minimum. Therefore we do not elaborate further on this topic.

5 The complexity of global ranking functions

In the previous sections we studied the local type of ranking functions, and saw that exponentially many functions may be necessary. We now want to confirm the intuition that a single, global ranking function may have to be exponentially complex. The case of strict SCT instances is once again a convenient case-study since a simple form of functions is known to suffice. Lee [11] shows how to construct a global ranking function of the form

$$\min(\max_{i \in S_1} x_i, \ \ldots, \max_{i \in S_m} x_i)$$

for some sets $S_1, \ldots, S_m$. The worst-case complexity of this construction is exponential, i.e., the number of sets S_i may be exponential[3]. One may try to overcome this by allowing more flexibility in the construction of the ranking function. It is natural to allow any formula involving min and max operators, not necessarily combined as above. There are, further, other operators that may be useful. For example, multiset ordering [8], dual multiset ordering [3] and lexicographic ordering are all useful for expressing size-change based termination proofs (see [18, 3]). These can be incorporated into the ranking-function expression by defining operators that form multisets and tuples, associated with the appropriate order relations.

For generality, instead of fixing the set of possible operators, we shall accept any definition that allows the ranking function values to be computed

[3] it is interesting to notice that the hard examples that we gave for the local approach happen to be easy in this setting, that is, require only a linear-sized expression; however, worst-case examples that cause exponential blowup are not hard to design.

in polynomial time (for example, multiset comparison can be done efficiently by sorting). In addition, we consider ranking relations instead of ranking functions. We further fix our domain to the natural numbers, which allows domain-specific operations (such as addition) to be used. Having allowed for so broad a class of ranking functions, it seems intractable to give a concrete hard example as in the previous sections. A more tractable goal, which we achieve in this section, is to show that a polynomial class of ranking functions (we formalize this notion below) would contradict widely-held conjectures in Complexity Theory. Our result is based on the PSPACE-hardness of the SCT problem (to this end, this paper includes a new hardness proof that applies to 1SSCT). At first it may seem that this hardness result (plus the customary hypothesis PSPACE $\neq$ P) implies immediately that ranking functions for SCT instances have to be super-polynomially large. But the implication is far from immediate, because it may be the case that it just takes exponential time to *find* the ranking function. The rest of this section shows how to derive a conclusion on the complexity of the ranking functions, given suitable definitions and assumptions.

The approach is to consider a ranking relation, or more precisely, a formula that represents such a relation, as a witness for termination. We consider the complexity of verifying such a witness, and prove that it suffices to test the witness over a small domain (namely numbers of polynomially many bits). If we also assume that computing the relation is polynomial-time, we find that the existence of a witness is a Σ_2^P problem, a type of problem that under standard complexity-theoretic assumptions cannot be PSPACE-complete.

Let us denote by $\mathcal{F}$ any set of formulae (technically, strings) over variables $X = \{x_1, x_2, \dots\}$ (there should be infinitely many potential variables); an element of $\mathcal{F}$ may be denoted by f. We denote by s, s', etc. states, that is, finite lists of integers that are interpreted as valuations of a corresponding initial segment of X.

The set $\mathcal{F}$ is supposed to be provided with a *decision procedure*, which is just a program $\mathtt{p}$ in a conventional computational model. For every $f \in \mathcal{F}$, the decision procedure defines a binary relation

$$R_f = \{(s, s') \mid [\![\mathtt{p}]\!]\langle f, s, s'\rangle = \mathtt{true}\}.$$

For example, let $\mathcal{F}$ include formulae of the form "$x_i > x_i'$". Given such f, the program $\mathtt{p}$ should report whether x_i does decrease from state s to state s'. Thus, $R_{\text{“}x_i > x_i'\text{”}}$ is the set of transitions in which x_i descends.

Definition 5.1. *We say that $\mathcal{F}$ is a* polynomially-computable class *if the*

decision procedure for $\mathcal{F}$ has running time polynomial in the input length.

Definition 5.2. *Let $\mathcal{G}$ be an SCT instance. We say that a relation R is a ranking relation for $\mathcal{G}$ if for every state transition $s \mapsto s'$ in $T_{\mathcal{G}}$ we have $(s, s') \in R$, and in addition, R is well founded.*

THEOREM 5.3. *A ranking relation for $\mathcal{G}$ exists if and only if $\mathcal{G}$ is a positive instance (satisfying SCT).*

Proof. It is well known that a ranking relation exists if and only if the transition system terminates. Thus this theorem just restates the completeness of SCT. □

Definition 5.4. *We say that $\mathcal{F}$ is a polynomial class of ranking formulae for (a class of) SCT instances if (i) $\mathcal{F}$ is polynomially computable and (ii) for every positive instance $\mathcal{G}$ (in this class), there is $f \in \mathcal{F}$ such that R_f is a ranking relation for $\mathcal{G}$. Moreover, $|f|$ (the string length of f) is polynomially bounded in $|\mathcal{G}|$ (the string length of $\mathcal{G}$).*

Note that the definition involves the time to decide the relation as well as the size of the formula. However, for "natural" operators, the computation time is bounded polynomially in the size of the formula and the inputs, so once we conclude that a polynomial class of ranking functions is unlikely to exist, this will imply that the formulas themselves have to be large.

In the sequel we use the notation $[m]$ for $\{0, \ldots, m-1\}$.

LEMMA 5.5. *Let $\mathcal{G}$ be a 1SSCT instance with n parameters. If $\mathcal{G}$* does not *satisfy SCT, there is a sequence $G_1, \ldots, G_N \in \mathcal{G}$ with $0 < N < 2^{n^2}$, and states $s_0, s_1, \ldots, s_{N-1} \in [Nn]^n$ such that $G_i \models s_i \mapsto s_{i+1 \pmod N}$ for $0 \leq i < N$.*

Informally, the lemma states that if $T_{\mathcal{G}}$ does not terminate, there is a finite cycle in the state space (note that with general programs, nontermination does not necessarily take the form of a finite cycle). Moreover, the lemma gives bounds on the length of the cycle and the size of the states.

Proof. If $\mathcal{G}$ *does not* satisfy SCT, there is a sequence $G_1, \ldots, G_N \in \mathcal{G}$ such that $G = G_1; \cdots; G_N$ fails the test indicated in Theorem 2.13; that is, there is no directed cycle in the circular variant G'. Suppose that we choose such a sequence of minimal length; the bound $N < 2^{n^2}$ follows by a simple counting argument.

We can consider a strict size-change graph as a set of constraints $x'_j < x_i$ over variables $x_1, \ldots, x_n$. The absence of a cycle in such a set is equivalent to

the consistency of the constraints. The multipath $G_1 \dots G_N$ can be viewed as a directed graph D with nodes $x_i^{(j)}$ for $i = 1, \dots, n$ and $j = 0, \dots, N$. The arcs of G_j lead from parameters $x_i^{(j-1)}$ to parameters $x_i^{(j)}$. If G' has no directed cycle, then there is no cycle in the digraph D' obtained from D by identifying nodes $x_i^{(N)}$ with $x_i^{(0)}$, so D' is a consistent set of constraints over the Nn variables. As the constraints only involve the order relation, if they are consistent, they have a model where the values form some permutation of $[Nn]$. The conclusion of the lemma is immediate. □

LEMMA 5.6. *Let $\mathcal{G}$ be a 1SSCT instance with n parameters, and let $R \subset \mathbb{N}^n \times \mathbb{N}^n$ be a well-founded relation. If $\mathcal{G}$* does not *satisfy SCT, then there is $G \in \mathcal{G}$ and states $s, s' \in [n \cdot 2^{n^2}]^n$ such that $G \models s \mapsto s'$ and $\neg R(s, s')$.*

Proof. Let the sequence $G_1, \dots, G_N$ and states $s_0, s_1, \dots, s_{N-1} \in [Nn]^n$ be as in the last lemma. Since R is well founded, we cannot have $R(s_i, s_{i+1 \bmod N})$ for all i. Thus a pair s, s' as claimed exists. □

Recall that Σ_2^P (also known as $\mathrm{NP}^{\mathrm{NP}}$) is the complexity class consisting of sets that can be decided in nondeterministic polynomial time using an NP oracle. Alternatively, it can be characterized as the class of predicates $A(x)$ expressible in the form $\exists_p y \forall_p z B(x, y, z)$ where $\exists_p$, $\forall_p$ quantify over polynomially-long strings and B is polynomial-time decidable. It is commonly presumed [14] that Σ_2^P is a strict subclass of PSPACE; a claim that implies $\mathrm{PSPACE} = \Sigma_2^P$ is considered very unlikely to be true.

THEOREM 5.7. *If there is a polynomial class of ranking formulae for 1SSCT instances, then* $\mathrm{PSPACE} = \Sigma_2^P$.

Proof. Suppose that a polynomial class of ranking formulae $\mathcal{F}$ is sufficient for 1SSCT instances. Then $\mathcal{G}$ satisfies SCT if and only if there is an $f \in \mathcal{F}$ with $|f| < p(|\mathcal{G}|)$ (for an appropriate polynomial p) such that for all $G \in \mathcal{G}$ and all states s, s' it holds that

$$G \models R_f(s, s').$$

By Lemma 5.6, we can replace "for all states" by "for all $s, s' \in [n \cdot 2^{n^2}]^n$." Thus the lengths of the representations of s, s' (with numbers in binary notation) are polynomial in n. Hence, the 1SSCT problem lies in Σ_2^P. Since it is hard for PSPACE (for a proof, see the following section), we obtain $\mathrm{PSPACE} = \Sigma_2^P$. □

6 PSPACE-hardness of 1SSCT

PSPACE-hardness of SCT is proved in [12], but not for the subproblem 1SSCT, a gap that we close in this section. The proof is by reduction from a well-known PSPACE-complete problem.

Definition 6.1 (NFA Universality.). *INSTANCE: A nondeterministic finite automaton $A = (Q, \Sigma, \delta, Q_0, F)$, where as usual $\delta \subseteq Q \times \Sigma \times Q$ is the set of transitions, Q_0 the set of initial states and F the set of accepting states.*
QUESTION: Is $L(A) = \Sigma^*$?

We begin by simplifying the problem somewhat into the following

Definition 6.2 (Simplified NFA Universality.). *INSTANCE: A nondeterministic finite automaton $A = (Q, \Sigma, \delta, Q, Q)$. That is, an automaton in which every state is both initial and accepting.*
QUESTION: Is $L(A) = \Sigma^*$?

LEMMA 6.3. *NFA Universality can be polynomially reduced to Simplified NFA Universality.*

Proof. Given any NFA A whose universality we wish to test, let $\#$ be a symbol not in Σ. Let $A' = (Q, \Sigma \cup \{\#\}, \delta', Q, Q)$ where

$$\delta' = \delta \cup (F \times \{\#\} \times Q_0).$$

Clearly, A' is a Simplified NFA instance and constructed in polynomial time. It is easy to verify that $L(A') = (\Sigma \cup \{\#\})^*$ if and only if $L(A) = \Sigma^*$. □

We now reduce Simplified NFA Universality to 1SSCT.

Let $A = (Q, \Sigma, \delta, Q, Q)$. We construct an SCT instance $\mathcal{G}$ with parameter set Q and a call c for each $c \in \Sigma$. The size-change graph G_c contains an arc $p \to q'$ whenever $(p, c, q) \in \delta$.

It is easy to see that A accepts a word $w = c_1 \dots c_t$ if and only if the multipath $G_{c_1} \dots G_{c_t}$ has a complete thread.

CLAIM 6.4. $L(A) = \Sigma^* \iff \mathcal{G} \in \text{SCT}$.

Proof. Assume that $L(A) = \Sigma^*$. Thus every $\mathcal{G}$-multipath has a complete thread. Let $M = G_1 G_2 \dots$ be an infinite multipath. Let $i > 0$ and consider the finite multipaths $G_i \dots G_j$ for all $j > i$. Each such multipath has a complete thread τ_{ij}; thus there is at least one parameter $q \in Q$ such that

infinitely many such threads begin at q. Let $Q_i \subseteq Q$ be the set of such parameters.

Now, if $q \in Q_i$, then G_i must have an arc from q into Q_{i+1}; otherwise a contradiction would arise since the number of threads from q would be finite. It follows that an infinite thread can be traced by beginning at Q_0 and always following arcs that lead from Q_i to Q_{i+1}. Thus, SCT is satisfied.

Now, assume that SCT is satisfied by $\mathcal{G}$; then every infinite multipath contains an infinite thread. Let M be any finite multipath and consider M^ω, i.e., the concatenation of infinitely many copies of M. From an infinite thread in M^ω one can clearly "cut out" a complete thread in one of the copies of M. Thus, every finite multipath has a complete thread, that is, $L(A) = \Sigma^*$. □

THEOREM 6.5. *The 1SSCT problem is PSPACE-complete.*

Proof. The PSPACE-hardness of NFA Universality, plus the two reductions, proves that 1SSCT is PSPACE-hard. In fact, it is PSPACE-complete, since SCT (and therefore 1SSCT) is decidable in polynomial space [12]. □

7 Conclusion

We have investigated the complexity of ranking functions for SCT programs, in the restricted case where all size-change arcs are strict (the restriction to a single-function instance is not crucial). For this simple case, exponentially many simple, local ranking functions suffice for the termination proof, as is a single, exponentially complex global ranking function. The main point of the article was to provide lower bounds, showing that this tradeoff is inherent. The investigation led to a clarification (at least in the author's mind) of the way certain types of ranking functions (linear, max, min, and others) relate to SCT analysis.

A few research questions arise. (1) Practically, include ranking functions like **max min**, etc. in a ranking-function based termination analyzer. Perhaps there are other functions to be considered? (2) Replace Theorem 5.7 by an explicit lower bound (i.e., construction of a hard instance) for an interesting class of expressions. (3) Look for a polynomial familty of global ranking functions that covers a significant variety of real-life programs.

Recent work on the last point is reported in [2].

Acknowledgement I am grateful to the Acta Informatica referees for their help in improving this paper.

References

[1] James Avery. Size-change termination and bound analysis. In M. Hagiya and P. Wadler, editors, *Functional and Logic Programming: 8th International Symposium, FLOPS 2006*, volume 3945 of *Lecture Notes in Computer Science*. Springer, 2006.

[2] Amir M. Ben-Amram and Michael Codish. A SAT-based approach to size change termination with global ranking functions. In C.R. Ramakrishnan and Jakob Rehof, editors, *14th Intl. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 5028 of *LNCS*, pages 46–55. Springer, 2008.

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

[4] Michael Codish, Vitaly Lagoon, and Peter J. Stuckey. Testing for termination with monotonicity constraints. In Maurizio Gabbrielli and Gopal Gupta, editors, *Logic Programming, 21st International Conference, ICLP 2005*, volume 3668 of *Lecture Notes in Computer Science*, pages 326–340. Springer, 2005.

[5] Michael Codish and Cohavit 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).

[6] Byron Cook, Andreas Podelski, and Andrey Rybalchenko. Termination proofs for systems code. In Michael I. Schwartzbach and Thomas Ball, editors, *Proceedings of the ACM SIGPLAN 2006 Conference on Programming Language Design and Implementation (PLDI), Ottawa, Canada, June 2006*, pages 415–426. ACM, 2006.

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

[8] Nachum Dershowitz and Zohar Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, august 1979.

[9] N. D. Jones and N. Bohr. Termination analysis of the untyped lambda calculus. In *Proceedings of the 15th International Conf. on Rewriting Techniques and Applications, RTA'04*, volume 3091 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2004.

[10] N.D. Jones and A. Glenstrup. Termination analysis and specialization-point insertion in off-line partial evaluation. Technical Report D-498, DIKU, University of Copenhagen, Denmark, 2004.

[11] Chin Soon Lee. Ranking functions for size-change termination, 2008. *ACM Transactions on Programming Languages and Systems*, to appear.

[12] Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. The size-change principle for program termination. In *Proceedings of the Twenty-Eigth ACM Symposium on Principles of Programming Languages, January 2001*, volume 28, pages 81–92. ACM press, January 2001.

[13] Naomi Lindenstrauss and Yehoshua Sagiv. Automatic termination analysis of Prolog programs. In Lee Naish, editor, *Proceedings of the Fourteenth International Conference on Logic Programming*, pages 64–77, Leuven, Belgium, Jul 1997. MIT Press.

[14] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, New York, 1994.

[15] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.

[16] Andreas Podelski and Andrey Rybalchenko. Transition invariants. In Harald Ganzinger, editor, *LICS'04: Logic in Computer Science*, pages 32–41. IEEE Computer Society, 2004.

[17] Yehoshua Sagiv. A termination test for logic programs. In Vijay Saraswat and Kazunori Ueda, editors, *Logic Programming, Proceedings of the 1991 International Symposium, San Diego, California, USA*, pages 518–532. MIT Press, 1991.

[18] René Thiemann and Jürgen Giesl. The size-change principle and dependency pairs for termination of term rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 16(4):229–270, September 2005.

[19] Alan 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.