A SAT-Based Approach to Size Change Termination with Global Ranking Functions

Amir M. Ben-Amram[1] and Michael Codish[2]

[1] School of Computer Science, Tel-Aviv Academic College, Israel.
[2] Department of Computer Science, Ben-Gurion University, Israel
amirben@mta.ac.il mcodish@cs.bgu.ac.il

Abstract. We describe a new approach to proving termination with size change graphs. This is the first decision procedure for size change termination (SCT) which makes direct use of global ranking functions. It handles a well-defined and significant subset of SCT instances, designed to be amenable to a SAT-based solution. We have implemented the approach using a state-of-the-art Boolean satisfaction solver. Experimentation indicates that the approach is a viable alternative to the complete SCT decision procedure based on closure computation and local ranking functions. Our approach has the extra benefit of producing an explicit witness to prove termination in the form of a global ranking function.

1 Introduction

Program termination is a cornerstone problem of program verification, as well as being the quintessential example for undecidability. In practice, however, there is a growing conviction that automated termination analysis is viable. This can be explained by the hypothesis that in realistic, correct programs, termination usually follows from reasons that are not very complex. The challenge of termination analysis is to design a useful program abstraction that captures the properties needed to prove termination as often as possible, while making termination checking decidable.

The Size-Change Termination method (SCT) is such an approach. Formally introduced in [17], SCT is a program abstraction where termination is decidable. Briefly, an abstract program is a directed *control-flow graph* (CFG), where each arc is an abstract transition, specified by its source and target locations and annotated by a *size-change graph*. The latter describes how the sizes of program data are affected by the transition. The abstract program terminates if and only if every infinite CFG path implies that a value decscends infinitely. We assume that the values described by the size-change graphs are well-founded, so infinite descent is impossible.

The size-change termination method has been successfully applied in a variety of different application areas [18, 8, 15, 24, 23, 1, 19]. A significant factor in the success of the method is that, in line with other recent work [5, 10, 11, 13], it departs from the classic approach of seeking a termination proof in the form

of a *global ranking function*—a function that ranks program states so that the rank decreases on every transition. Instead [17] gave an algorithm that takes a local approach, covering all possible CFG cycles and proving termination of each. In [6], the SCT condition is expressed in terms of assigning a *local ranking function* to every possible cycle. That paper shows that it suffices to restrict local ranking functions to a very simple form—namely sums of subsets of the (abstract) program variables. But local ranking functions cannot serve as a witness to termination that can be checked against any transition, in the way a global ranking function can be used.

It is this difference that motivates our interest in global ranking functions: the ranking expression (i.e., the formula that represents the ranking function) is useful as a *certificate*, which can be used to verify the claim that a program terminates. As pointed out in [16], such a setup allows a theorem prover to certify the termination claim while allowing the tool that searches for the termination proof to stay outside the trusted (formally verified) code base. One can also consider applications to proof-carrying code, where again the desire is for the proof to be given as a certificate that is easier to check than to find.

Thus reconciling the SCT method with the global ranking function approach is a theoretical challenge with a practical motivation. Initially, it had not been clear whether there is any constructive way to characterize an SCT termination proof in terms of a global ranking function. A break-through was achieved by Chin Soon Lee [12], who characterized a class of expressions that are sufficient for globally-ranking any terminating SCT program, and showed that the ranking expression can be effectively constructed from the size-change graphs. While this class of expressions is syntactically simple, the ranking expressions themselves can be exponentially large (the upper bound in [12] is triple-exponential). This makes its usage as a certificate difficult.

As Krauss [16] points out, there is a complexity-theoretic argument that precludes the existence of short and simple certificates for size-change termination: the SCT decision problem is PSPACE-complete, and such problems do not have polynomially-verifiable certificates. Ben-Amram [2] gives a more detailed proof, that shows that the assumption "polynomially verifiable" can be replaced with the even humbler "polynomially computable."

Our proposition, presented in this paper, for solving this difficulty is to define a *subset* of SCT instances that is rich enough for practical usage *and* has concise (polynomial) global ranking expressions. Such a set would naturally be in NP, and thus our proposition also gives an answer to a natural theoretical curiosity—namely is there an interesting subset of SCT at the NP level. If a set is in NP, it is amenable for solution strategies not available for PSPACE-complete problems; in particular, it is reducible to SAT, and given the performance of state-of-the art SAT solvers, this is practically significant.

A subset of SCT that is decidable in polynomial time, and hence called SCP, has previously been presented in [4]. This subset is, however, defined implicitly, by giving an algorithm, also called SCP. This algorithm can be seen as a heuristic to recognize programs in our class, called, analogically, SCNP. Thus, SCNP is

the natural SCT subset that encompasses the instances handled by SCP. The arguments and examples given in [4] to explain the usefulness of SCP provide the initial assurance that SCNP is rich enough. Now, we can also support this claim by ample experimental evidence.

Contributions of this work. We identify a class of expressions that are useful for globally ranking size-change terminating programs: the expressions are concise, they are constructed so that proving the descent in every transition is relatively simple, and they are expressive enough for practical usage of SCT. We define SCNP as the class of SCT instances for which this proof strategy works. We turn this characterization into an effective algorithm using SAT and constraint-solving technology. We have thus created the first tool that not only verifies that an SCT instance terminates, but also produces a ranking expression. To further improve its usefulness for certification, our tool also outputs the *justification*, that is, the argument that links the ranking function to the size-change graphs.

Here are some observations related to comparison between the different SCT algorithms. The standard algorithm for deciding SCT in full is based, as mentioned above, on the local approach and composition closure computation. The SCP algorithm of [4] already handles some examples of the following kinds: (1) instances which need exponentially many local ranking-functions (see [2]), therefore driving the closure algorithm to exponential behavior; (2) instances where any ranking expression of the form used in [12] must be exponentially big [3]. Thus, our method also handles such examples. We also illustrate where our method outperforms SCP.

The next two sections contain fuller definitions and background fact on SCT and ranking functions. Section 4 describes our class of ranking functions and some related theory, and Sections 5–6 describe the implementation and experiments performed so far.

2 Size Change Termination

This section introduces the SCT abstraction, and reviews the major facts about SCT decision procedure(s). Terminology is not uniform across the related publications and we have made an arbitrary choice. For instance we use the term *program point* where some references use, e.g., *flow-chart point*, *program location* and *function*, the latter obviously in a functional-programming context.

An abstract program is just a set of *abstract transitions*. An abstract transition is a relation on abstract program states. In this work we only care about abstract states and transitions: therefore, it is programming-language independent. It assumes a front-end to analyze concrete programs and create the abstraction for our analysis. Appropriate front ends already exist for various programming systems [8, 24, 23, 1, 19, 16].

Definition 1 (program). *An (abstract) program consists of a finite set P of* program points*, where every program point p is associated with a fixed list $Arg(p)$*

of argument positions*; and of a finite set of* size-change graphs*, defined below, representing possible transitions. The number of argument positions for p is called the arity of p. We sometimes write p/n to denote that p is of arity n.*

Definition 2 (program state). *Let Val be a fixed, infinite well-ordered set. An (abstract) program state is given by associating a value from Val to each argument position of a program point p. The set of all states is St.*

In this paper we write a program state down as a term $p(u_1, \ldots, u_n)$, where n is understood to be the arity of p. The argument positions may represent actual data in the program (of type consistent with *Val*), but quite often they represent some "size measures" or "norms" associated with the actual data.

Definition 3 (size-change graph). *A size-change graph is formally a triple $g = (p, q, A)$ where p, q are program points and $A \subset Arg(p) \times SizeLabel \times Arg(q)$ is a set of* size-change arcs *with $SizeLabel = \{\downarrow, \overline{\downarrow}\}$ indicating strict or non-strict descent between an argument of p and and argument of q.*

As an alternative notation, we represent a size-change graph as a constraint logic programming clause $p(\bar{x}) :- \pi; q(\bar{y})$ with $\bar{x} = x_1, \ldots, x_n$, $\bar{y} = y_1, \ldots, y_m$, and π a conjunction of constraints of the form $(x_i > y_j)$ or $(x_i \geq y_j)$. We also write $\pi \models \phi$ to indicate that proposition ϕ (involving values $\bar{x}$ and $\bar{y}$) holds under the assumptions π.

Since the size-change graphs implicitly reveal the set of (relevant) program points, we identify an abstract program with a set $\mathcal{G}$ of size change graphs. The *control-flow graph* (CFG) of the program is the directed (multi-)graph over P underlying $\mathcal{G}$ (namely, every size-change graph corresponds to an arc).

Definition 4 (transitions). *A* state transition *is a pair (s, s') of states. Let $g = p(\bar{x}) :- \pi; q(\bar{y})$ be a size-change graph. The associated set of transitions is $T_g = \{(p(\bar{x}), q(\bar{y})) \mid \pi\}$. The transition system associated with a set of size-change graphs, $\mathcal{G}$, is $\mathcal{T}_\mathcal{G} = \bigcup_{g \in \mathcal{G}} T_g$.*

Definition 5 (termination). *Let $\mathcal{G}$ be an SCT instance. A* run *of $\mathcal{T}_\mathcal{G}$ is a (finite or infinite) sequence of states $s_0, s_1, s_2 \ldots$ such that for all i, $(s_i, s_{i+1}) \in \mathcal{T}_G$. Transition system $\mathcal{T}_\mathcal{G}$ is* uniformly terminating *if it has no infinite run.*

An SCT instance $\mathcal{G}$ is positive (or, *satisfies SCT*) if $\mathcal{T}_\mathcal{G}$ is uniformly terminating Fortunately, this is not dependent on the specific choice of *Val*, which justifies considering it as a property of $\mathcal{G}$. This "semantic" definition is not the one given in [17]; indeed, in that paper a "combinatorial" definition is given, in terms of the graphs, and it is a significant result that the given property is equivalent to uniform termination. However, since they are equivalent, we can forego the combinatorial definition, as its details are not used in this work.

It was proved in [17] that the set of positive SCT instances is decidable. Also its complexity class was determined: it is PSPACE-complete. Decidability is proven in two ways, one of which is direct, i.e., an algorithm that specifically solves this problem. It is based on computing the composition-closure of $\mathcal{G}$, a

technique already used by other termination analyzers [18, 8]. Such an algorithm is obviously exponential-time. Most current implementations of SCT use this algorithm, but an obvious concern regarding its complexity prompted Lee and Ben-Amram to look for a polynomial-time decidable subset of SCT. In [4], they present a polynomial-time algorithm (called SCP, for Size-Change termination in Ptime) that decides such a subset. In experiments, it performed very well on the benchmark used in that work—a collection of example programs obtained from researchers working on Prolog termination. An interesting part of [4] is an attempt to explicate the capabilities of this algorithm, since it is heuristic and it is not *a priori* obvious why or when it should be successful. The explanation given links the algorithm to specific "size-change patterns"—including lexicographic descent, multiset descent, and dual-multiset descent (the last two terms are defined later in this paper). These observations formed the starting point of the current work, along with another theoretical discovery, discussed in the next section.

3 Ranking functions

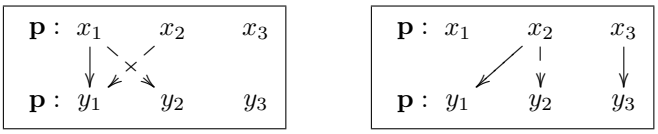
In this section we introduce ranking functions and describe some known facts regarding SCT and ranking functions.

Definition 6 (quasi-order). *A* quasi-order *over a set D is a binary relation $\succsim \subseteq D \times D$ which is transitive and reflexive. Its* strict part *$\succ$ is defined by $x \succ y \iff x \succsim y \wedge y \not\succsim x$. A well-quasi-order means a well-founded one.*

Definition 7 (ranking function). *Let $\mathcal{G}$ be a set of size change graphs. A function ρ which maps program states to a well-quasi-order $(D, \succsim)$ is a (global) ranking function for $\mathcal{G}$ if for every graph $g = p(\bar{x}) :- \pi; q(\bar{y}) \in \mathcal{G}$, it holds that $\pi \models \rho(p(\bar{x})) \succ \rho(q(\bar{y}))$.*

The qualifier *global* is used to distinguish this notion of a ranking function from another (local) one (which is not used in this work). When depicting graphs we use solid and dashed lines to respectively indicate strict and nonstrict edges.

Example 1. Consider the following two size-change graphs:



A ranking function for this set is $\rho(x_1, x_2, x_3) = \max\{\langle x_1, 1, x_3\rangle, \langle x_2, 0, x_3\rangle\}$ where tuples are ordered lexicographically.

We remark that lexicographic ordering is used throught this work, and we rely on the well-known fact that if D is well-founded then so is the set of tuples over D of a fixed length, or of a bounded length (when comparing tuples of differing lengths, if one is a prefix of the other, it is considered smaller).

It is obvious that the existence of a ranking function for $\mathcal{G}$ implies termination. This is, in fact, an equivalence (for the other direction, just take $\mathcal{T}_{\mathcal{G}}$ as a quasi-order). The ranking technique has been known for a long time and seems to be the natural way to prove termination of transition systems. Even if we know that a system terminates, an explicit ranking function conveys interesting information about the way that the computation progresses towards its finish line.

Another view is that a ranking function (given explicitly as some kind of expression) constitutes a *witness*, or *certificate*, to termination. When presented with the expression, one only has to verify that it does decrease on every transition This is conceptually easier than establishing a termination proof from scratch, essentially because it only requires arguing about a single transition at a time. Thus, the argument needed to prove termination, given a ranking function, is logically simpler than the argument necessary for a termination proof using only the SCT graphs, even though this is possible. This simplification is an advantage for users who wish to establish the termination of a program within a theorem-prover [16]. Depending on the form of the ranking function, verifying a certificate may also be easier in a computational-complexity sense (and perhaps a programming sense) than proving termination from the graphs. These considerations are important in "proof-carrying code" scenarios [21].

The SCT criterion was proved in [17] to be decidable, but the algorithms given do not construct a ranking function. The question of whether it is at all possible to obtain an explicit expression for the ranking function (we know that a function exists, once termination is proved!) has only been settled (in the positive) a few years later, in a paper by Chin Soon Lee [12]. The size of the ranking expression in this work is triply-exponential in the worst-case. Recent progress [3] reduces this to a single exponent, but obviously this is still a deterring complexity for practical usage (such as for certification).

However, these works provide an important theoretic underpinning to working with ranking functions for SCT, as they exhibit a class of functions within which all SCT instances can be ranked.

Theorem 1 (Lee 2007). *If $\mathcal{G}$ is SCT, it has a ranking function, effectively constructible from the size-change graphs, of the form: $\rho(s) = \min\{M_1, M_2, \dots\}$ where M_i is $\max\{t_1^i, t_2^i, \dots\}$ and each t_j^i is a tuple of arguments (of the state s) and constants.*

The ranking function in Example 1 has precisely this form (degenerate in that the min operator is unnecessary).

Lee's result indicates a small and yet sufficient set of operators for constructing ranking functions, and also shows that a very limited way of combining them is sufficient. The fact that the expressions are very limited makes the theoretical result even more remarkable. However, it is natural to expect that by narrowing down the class of expressions, we make it more likely that the representation of the ranking function will be big.

In general, we cannot expect *any* reasonable class of expressions to beat the exponential upper bound. This follows from complexity-theoretic considerations,

as explained in [2]. However, it is easy to find *instances* where the use of additional operators or expression structures can allow for more concise expressions.

Example 2. The function $\rho(x_1, x_2, \ldots, x_n) = \langle \max\{x_1, x_2\}, \ldots, \max\{x_{n-1}, x_n\} \rangle$ is of linear size. Expressing it in the form max-of-tuples leads to an expression with $2^{n/2}$ tuples. Thus, simply changing the nesting structure suffices for an exponential improvement in size.

In this work we use several expression constructors that yield concise expressions where, in some cases, the simple forms would lead to exponential size. On the other hand, our class of expressions is not universal. Its specific design is a crucial ingredient in our work and is unveiled in the next section.

4 SCNP: Size-change Termination NP Subset

This section introduces a class SCNP of size change termination problems. Its definition is derived from a specific form of ranking functions. We *first* define a class of functions; then we define SCNP to include an SCT instance if and only if it has a ranking function of our class. Thus, this definition is *based* on ranking functions and the fact that the resulting subset is NP is an immediate consequence of showing that our ranking functions have polynomial size and that the problem of checking a function against a set of graphs is also in NP.

The building blocks for our construction are four types of well-quasi-orders and certain *level mappings* which are functions mapping program states into these orders. These are defined next.

Definition 8 (multiset extensions). *Let $(D, \succsim)$ be a total order and $(D, \succ)$ its strict part. Let $\wp(D)$ denote the set of multisets of elements of D of at most n elements, where n is fixed by context*[1]*. The μ-order extension of $(D, \succsim)$, for $\mu \in \{ max, min, ms, dms \}$, is the quasi-order $(\wp(D), \succsim^{\mu})$ where:*

1. (max order) *$S \succsim^{max} T$ holds iff $\max(S) \succsim \max(T)$, or T is empty; $S \succ^{max} T$ holds iff $\max(S) \succ \max(T)$, or T is empty while S is not.*
2. (min order) *$S \succsim^{min} T$ holds iff $\min(S) \succsim \min(T)$, or S is empty; $S \succ^{min} T$ holds iff $\min(S) \succ \min(T)$, or S is empty while T is not.*
3. (multiset order [14]) *$S \succ^{ms} T$ holds iff T is obtained by replacing a non-empty sub-multiset $U \subseteq T$ by a (possibly empty) multiset V such that $U \succ^{max} V$; The weak relation $S \succsim^{ms} T$ holds iff $S \succ^{ms} T$ or $S = T$.*
4. (dual multiset order [4]) *$S \succ^{dms} T$ holds iff T is obtained by replacing a sub-multiset $U \subseteq S$ by a non-empty multiset V with $U \succ^{min} V$; The weak relation $S \succsim^{dms} T$ holds iff $S \succ^{dms} T$ or $S = T$.*

Example 3. Let $S = \{ 4, 3, 3, 0 \}$ and $T = \{ 4, 3, 2, 1, 1 \}$. We have $T >^{min} S$, $S \geq^{max} T$ and $T \geq^{max} S$. We have $S >^{ms} T$ because $U = \{3, 0\}$ is replaced by $V = \{2, 1, 1\}$, where all elements are smaller than $\max(U)$. We don't have $S >^{dms} T$, but $T >^{dms} S$.

[1] Given an SCT instance $\mathcal{G}$, n is the maximum arity in $\mathcal{G}$.

We give the following well-known facts without proof.

Lemma 1. *When the underlying order is total, so are all the multiset extensions. If $(D, \succ)$ is well-founded, then so is each of the extensions $(D, \succ^\mu)$.*

Note however that the min/max orders are not partial orders, but rather quasi orders as anti-symmetry does not hold. *Why these four orders?* The motivation lies in previous work with SCT, which identified these as being useful in termination arguments that can be proved with size-change graphs.

Definition 9 (level mappings). *Let $\mathcal{G}$ be a set of size change graphs involving N program points; and let M be the sum of arities of all program points. A* level mapping *is a function f from St to a certain (quasi) ordered set. In this work, level mappings are one of the following:*

numeric: *f maps each program state $s = p(\bar{u})$ to a natural number $0 \leq f(s) < N$, such that $f(s)$ only depends on the program-point.*

plain: *f maps a program state $p(\bar{u})$ to a multiset $\{\!| v_1, \ldots, v_k |\!\} \in \wp(Val)$ where $v_1, \ldots, v_k$ are arguments in $\bar{u}$; the selection of argument positions only depends on the program point.*

tagged: *f maps a program state $p(\bar{u})$ to a multiset $\{(v_1, n_1), \ldots, (v_k, n_k)\} \in \wp(D \times \mathbb{N})$ where $v_1, \ldots, v_k$ are elements of $\bar{u}$ and $n_1, \ldots, n_k$ are natural numbers less than M (called tags). The selection of argument positions as well as the tags is determined by the program point.*

We use a subscripted annotation on f to indicate the order associated with its range and write f_μ with $\mu \in \{\!| max, min, ms, dms |\!\}$ for the multiset orders (both over Val and over $Val \times \mathbb{N}$). We write f_ω for the numeric level mapping, where the order $\succsim^\omega$ is the natural order $\geq$ on integers.

Example 4. The following are plain and tagged level mappings (respectively), assuming a program with abstract states $p/2$ and $q/3$:

$$f(s) = \begin{cases} \{u, v\} & \text{if } s = p(u, v) \\ \{u\} & \text{if } s = q(u, v, w) \\ \emptyset & \text{otherwise} \end{cases} \qquad f'(s) = \begin{cases} \{\langle u, 0\rangle\} & \text{if } s = p(u, v) \\ \{\langle u,1\rangle, \langle w,0\rangle\} & \text{if } s = q(u, v, w) \\ \emptyset & \text{otherwise} \end{cases}$$

Definition 10 (SCNP). *A set of size change graphs is in SCNP if it has a ranking function which is a tuple of level mappings.*

Example 5. Consider the size change graphs below

$$\mathcal{G} = \left\{ \begin{array}{l} p(x_1, x_2) :- \;\; x_1 > y_1,\, x_2 \geq y_2,\, x_1 \geq y_3; \quad q(y_1, y_2, y_3), \\ q(x_1, x_2, x_3) :- \;\; x_1 \geq y_1; \quad p(y_1, y_2) \end{array} \right\}$$

With max-ordering $\rho(s) = \langle f'(s), f(s)\rangle$ is a ranking function for $\mathcal{G}$; the reader may find it interesting to figure out the justification before reading further.

Definition 11 (orienting a graph). *We say that a level mapping f_μ* orients *a size-change graph $g = p(\bar{x}) :- \;\pi; q(\bar{y})$ if $\pi \models f(p(\bar{x})) \succsim^\mu f(q(\bar{y}))$; it orients g* strictly *if $\pi \models f(p(\bar{x})) \succ^\mu f(q(\bar{y}))$; and it orients a set $\mathcal{G}$ of size-change graphs if it orients every graph of $\mathcal{G}$, and at least one of them strictly.*

The next lemma is immediate from the definitions.

Lemma 2. *Let $f^1, \ldots, f^m$ be level mappings. The function*

$$\rho(s) = \langle f^1(s), f^2(s), \ldots, f^m(s) \rangle$$

is a (lexicographic) ranking function for $\mathcal{G}$ if and only if for every $g \in \mathcal{G}$, there in an $i \leq m$ such that g is oriented by $f^1, \ldots, f^i$ and strictly oriented by f^i.

Definition 12. *Function $\rho(s) = \langle f^1(s), f^2(s), \ldots, f^m(s) \rangle$ is* irredundant *if for all $i \leq m$, f^i orients all graphs that are not strictly oriented by f^j for some $j < i$, and strictly orients at least one of these graphs.*

It follows that an irredundant function is a tuple of length at most $|\mathcal{G}|$. It also shows that verifying such a ranking function reduces to testing whether a graph is (strictly) oriented by a given level mapping. We elaborate on this test for each kind of level mapping. We assume that a level mapping f is given explicitly, by listing the set of argument positions and/or natural number associated with every program point. For the case of numeric level mappings the test is trivial.

Plain level mappings Let f_μ be a plain level mapping and $g = p(\bar{x}) :- \pi; q(\bar{y})$. It is convenient to view g as a graph (in the way of Definition 3). Thus $\pi \models x \geq y$ is equivalent to g having an arc $x \to y$, while $\pi \models x > y$ if the arc is strict. By g^t we denote the transpose of g (obtained by inverting all the arcs). Let $S \subseteq \bar{x}$ be the set of argument positions selected by $f_\mu(p(\bar{x}))$ and similarly T for $f_\mu(q(\bar{y}))$. The definition of $\succsim^\mu$ indicates precisely what π has to satisfy. We ellaborate, assuming that S, T are nonempty.

1. *max order:* for non-strict descent, every $y \in T$ must be "covered" by an $x \in S$ such that $\pi \models x \geq y$. Strict max-descent requires $S \neq \emptyset$ and $x > y$.
2. *min order:* Same conditions but on g^t (thus, now T covers S).
3. *multiset order:* for non-strict descent, every $y \in T$ must be "covered" by an $x \in S$ such that $\pi \models x \geq y$. Furthermore each $x \in S$ either covers each related y strictly ($x > y$) or covers at most a single y. Descent is strict if there is some x of the strict kind.
4. *dual multiset order:* Same conditions but on g^t (thus, now T covers S).

Example 6. Consider again $\mathcal{G}$ from Example 5, and $f(s)$ from Exmple 4. f orients the first SCG (transition from p to q) strictly under *max* and *ms* orders, but not at all under their duals. The second SCG is oriented weakly under *min* and not under any other of the orders.

Tagged level mappings These are just like plain level mappings except that the underlying order is modified by the presence of tags. So to decide whether $\pi \models f(p(\bar{x})) \succ^\mu f(q(\bar{y}))$ where f is a set of tagged arguments, we use the rules given above for the multiset-extension μ, plus the following facts:

$$\pi \models (x, i) \succ (y, j) \iff (\pi \models x \succsim y) \wedge (\pi \models x \succ y \vee i > j)$$
$$\pi \models (x, i) \succsim (y, j) \iff (\pi \models x \succ y) \vee (\pi \models x \succsim y \wedge i \geq j)$$

As an example the reader may want to verify that the function f', defined in Example 4, does orient the graphs of Example 5 under *max* ordering, where the second one oriented strictly. This is, of course, used in arguing the correctness of the ranking function in that example.

We are now in position to state the following Theorem.

Theorem 2. *SCNP is in NP.*

Proof. If a ranking function of the desired form exists, then there is an irredundant one of polynomial size. Checking the condition in Lemma 2, according to the rules given above for the different orderings, is clearly polynomial-time.

Since the problem is in NP it is known that it can be reduced to SAT. In fact it is possible to transform a complete problem instance into one big Boolean formula whose satisfying assignment will encode the sought solution. To find the assignment we can make use of an off-the-shelf SAT solver. However, it is far more efficient to make use of the structure of the problem and call the SAT solver several times, on smaller SAT instances.

Our algorithm has the following top-level structure: as long as $\mathcal{G}$ is not empty, find a level mapping f that orients $\mathcal{G}$. Remove the graphs oriented strictly by f, and repeat.

Basically, the instruction "find a level mapping" is performed by trying each of the level-mapping types in turn, so that the smaller NP problems that we reduce to SAT are of the form: given $\mathcal{G}$ and μ, find out if there is a level mapping f_μ that orients it. Numeric level mappings have a special role in this algorithm. Since such a level mapping ignores the argument values we write it as $f(p(\_))$.

Claim. If there is a numeric level mapping f that orients $\mathcal{G}$ and program point q is reachable from p then $f(p(\_)) \geq f(q(\_))$.

This (obvious) property implies that f is constant in every strongly connected component (SCC) of $\mathcal{G}$ (considered as a graph with abstract transitions as arcs). Only inter-component transitions can be strictly oriented by f and in fact all of them can, by assigning f values according to reverse topological order of the SCCs. Now, after deleting the strictly-oriented transitions, SCCs will become disconnected from each other which allows them to be processed separately. We obtain the following revised algorithm.

1. Initialize ρ to the empty tuple.
2. Perform the following steps as long as $\mathcal{G}$ is not empty:
3. Compute the decomposition of $\mathcal{G}$ to strongly connected components (SCC's). If $\mathcal{G}$ has inter-component transitions, define a numeric level mapping f by reverse topological ordering of the SCC's. Extend ρ by f and remove the inter-component transitions.
4. If $\mathcal{G}$ has a non-trivial SCC[2], perform the following for each such component $\mathcal{C}$ in turn: (a) Apply SAT solving, as described in Section 5, to find a level

[2] That is, an SCC that contains at least one arc.

mapping f that orients $\mathcal{C}$. If no level mappings exists, exit with a failure message; and (b) Define the value of f as $\emptyset$ for all program points not in $\mathcal{C}$. Append f to ρ and remove the transitions strictly oriented by f.

We conclude this section by discussing the relations of SCNP, SCT and SCP. Viewing all three as decision problems (sets of instances), we have:

$$\text{SCP} \subset \text{SCNP} \subset \text{SCT}.$$

Arguments for these relations are as follows:

1. SCNP $\subseteq$ SCT because SCNP is a sound termination condition. The inclusion is strict because there are terminating instances not in SCNP. Here is such an example (from [4]).

   Example 7. $\left\{ \begin{array}{l} p(x,y,z,w) :- \ x \geq x', x \geq y', w > w'; \ \ p(x',y',z',w'), \\ p(x,y,z,w) :- \ y > x', y \geq y', z > z'; \ \ p(x',y',z',w') \end{array} \right\}$

2. SCNP handles some examples not handled by SCP; for instance, Example 1 on page 5 for which $\rho(x_1, x_2, x_3) = \langle \max\{(x_1, 1), (x_2, 0)\}, \max\{x_3\} \rangle$ is a ranking function.
3. Finally, the claim SCP $\subseteq$ SCNP follows from an analysis of the SCP algorithm that cannot be included in this conference paper, however we refer the reader to [4, Section 5] where some ideas of this analysis are already given.

On the point of complexity, it is interesting to observe that the hard cases for the SCT algorithms based on the local-ranking approach are not necessarily hard for our approach based on global ranking functions. In [2], an SCT instance is described, having $n+1$ arguments and n size-change graphs, that requires 2^n different local ranking functions of the kind discussed in [6]. This means that the closure-based SCT algorithms are driven to exponential behavior. But this instance is handled by SCNP and even SCP.

Finally, we prove that SCNP is complete for NP. Thus solving it with SAT is not an overkill in a complexity-theoretic sense.

Theorem 3. *SCNP is NP-complete.*

Proof. We will prove NP-hardness of a simplified problem—given $\mathcal{G}$ is there any level mapping that orients $\mathcal{G}$? We give a reduction from the well-known *Set Covering* problem SC, defined as follows:

INSTANCE: $\langle n, [S_1, \ldots, S_m], k \rangle$, where each $S_i \subseteq \{1, \ldots, n\}$, and $k \leq n$.
CONDITION: $\{1, \ldots, n\}$ can be covered by k of the sets S_i.

Given an instance of SC, construct size-change graphs as follows. Let $\bar{x} = \langle x_1, \ldots, x_n, x_{n+1}, \ldots, x_{2n-k} \rangle$. We have a single program point $p(\bar{x})$ and graphs $g_i = p(\bar{x}) :- \ \pi_i; q(\bar{x}')$ for $i = 1, 2$ with:

$$\pi_1 = \{x_i > x'_j \mid j \in S_i\} \cup \{x_i \geq x'_j \mid i \leq n,\ j > n\}$$
$$\pi_2 = \{x_i \geq x_{i+1} \mid i < 2n - k\} \cup \{x_{2n-k} \geq x_1\}.$$

Observe that the graph is strongly connected, so a numeric level mapping is ruled out. Graph g_2 is easily seen to defeat any multiset based level mapping that does not include all the arguments. Assuming $k < n$, graph g_1 defeats *min* and *dms* ordering, because x_{n+1} is not covered in g_1^t. It is easy to see that g_1 is oriented by *max*, but only weakly (the reader is invited to verify that tagging would not help). So, to orient g_1 strictly we need the *ms* ordering, and the set has to include all argument positions. Now, arguments x_{n+j} for $j = 1, \ldots, n-k$ can only be covered non-strictly, and so $n-k$ different source arguments are needed to cover them (recall that a source argument can only cover one target argument if it covers it non-strictly). The remaining k arguments among $x_1, \ldots, x_n$ have to cover $x'_1, \ldots, x'_n$ which clearly implies a solution to the Set Covering instance.

This reduction is interesting in that it shows that even if we know the level mapping (in this case $f_{ms}(p(\bar{x}) = \{x_1, \ldots, x_{2n-k}\}$) it is still NP-hard to verify it, just because of not knowing which arcs of the size-change graph to use. This observation motivates us to report not only the level mapping but also its justification as part of the output.

5 A SAT based implementation

This section is dedicated to our solution of the subproblem—finding an orienting level mapping—with the aid of a SAT solver. We use an approach described in [9], where the problem to be solved is encoded into Boolean and partial order constraints. The latter are propositional statements which contain both propositional variables and atoms of the form $(f > g)$ or $(f = g)$ where f and g are partial order symbols. A satisfying assignment assigns Boolean values to the propositional variables and integer values to the partial order symbols.

Let $\mathcal{G}$ be a set of size change graphs and $\mu \in \{ max, min, ms, dms \}$. We construct a propositional formula (with partial order constraints) to determine if there exists a tagged level mapping which μ-orients $\mathcal{G}$. We remark that a plain mapping need not be handled separately: it is a special case of the tagged mapping where all tags are the same. The proposition has the form $\Phi_\mu^\mathcal{G} = \varphi^\mathcal{G} \wedge \varphi_\mu^\mathcal{G}$ where $\varphi^\mathcal{G}$ is a representation of $\mathcal{G}$ and $\varphi_\mu^\mathcal{G}$ is specific for μ.

Encoding a set of size change graphs. Let $g = p(x_1, \ldots, x_n) :\!- \pi; q(y_1, \ldots, y_m)$. We create propositional variables e_c^g with c of the form $(p_i > q_j)$ or $(p_i \geq q_j)$. Intuitively, e_c^g represents the fact $\pi \models (x_i, tag_p^i) > (y_j, tag_q^j)$ (or $\geq$). The encoding introduces partial order constraints on the tag values tag_p^i and tag_q^j. We define $\varphi^\mathcal{G} = \bigwedge_{g \in \mathcal{G}} \varphi^\mathcal{G}$. The following formula encodes graph g. The propositions $\pi \models x_i > y_j$ (or $x_i \geq y_j$) that appear in it are replaced by `true` or `false` according to the size-change graph.

$$\varphi^g = \bigwedge_{\substack{1 \leq i \leq n \\ 1 \leq j \leq m}} \left(\begin{array}{r} (e^g_{x_i > y_j} \leftrightarrow (\pi \models x_i > y_j) \vee (\pi \models x_i \geq y_j) \wedge (tag_p^i > tag_q^j)) \\ \wedge\ (e^g_{x_i \geq y_j} \leftrightarrow (\pi \models x_i > y_j) \vee (\pi \models x_i \geq y_j) \wedge (tag_p^i \geq tag_q^j)) \end{array} \right)$$

Example 8. Let $g = p(x_1, x_2, x_3) \coloneq x_1 > y_1, x_1 \geq y_2; q(y_1, y_2, y_3)$. The encoding is a conjunction of subformulae. Let us consider several of them: The constraint $x_1{>}y_1$ contributes the conjuncts $e^g_{x_1>y_1}$ and $e^g_{x_1\geq y_2}$; The constraint $x_1{\geq}y_2$ contributes the conjuncts $(e^g_{x_1>y_2} \leftrightarrow tag^1_p > tag^2_p)$, and $(e^g_{x_1\geq y_1} \leftrightarrow tag^1_p \geq tag^2_p)$. The absence of a constraint between x_1 and y_3 contributes $\neg e^g_{x_1>y_3}$ and $\neg e^g_{x_1\geq y_3}$.

For the μ specific part of the encoding, the propositional variables $weak^g_\mu$ and $strict^g_\mu$ are interpreted as specifying that graph g is weakly (resp. strictly) oriented by μ respectively. Hence the encoding takes the form:

$$\varphi^{\mathcal{G}}_\mu = \left(\bigwedge_{g\in\mathcal{G}} weak^g_\mu\right) \wedge \left(\bigvee_{g\in\mathcal{G}} strict^g_\mu\right) \wedge \psi^{\mathcal{G}}_\mu$$

The first two conjuncts specify that there exists an f_μ which orients $\mathcal{G}$. The third conjunct $\psi^{\mathcal{G}}_\mu$ constrains variables $weak^g_\mu$ and $strict^g_\mu$ so that they are true exactly when the corresponding graphs are weakly (resp. strictly) oriented by μ. The propositional variables $p_1, \ldots, p_n$ and $q_1, \ldots, q_m$ indicate the argument positions of p and q selected for the level mapping.

Encoding the max set ordering: The following formula encodes the conditions described in Section 4.

$$\psi^{\mathcal{G}}_{max} = \bigwedge_{g=p(\bar{x}) \coloneq \pi;q(\bar{y})} \left(\begin{array}{l} weak^g_{max} \leftrightarrow \bigwedge\limits_{1\leq j\leq m} \left(q_j \rightarrow \bigvee\limits_{1\leq i\leq n} e^g_{x_i\geq y_j} \right) \wedge \\ \\ strict^g_{max} \leftrightarrow \bigwedge\limits_{1\leq j\leq m} \left(q_j \rightarrow \bigvee\limits_{1\leq i\leq n} e^g_{x_i> y_j} \right) \wedge \bigvee\limits_{1\leq i\leq n} p_i \end{array} \right)$$

Encoding the multiset ordering: We follow the encoding for the multiset ordering described in [22]. The operator $\oplus$ specifies that exactly one of a set of propositional variables is true. For each graph g, propositional variables $\gamma^g_{i,j}$ specify that in size change graph $g = p(\bar{x}) \coloneq \pi; q(\bar{y})$ the i^{th} argument of p/n covers the j^{th} argument of q/m. The propositional variables ε^g_i specify if whatever the i^{th} argument of p/n covers is weak (then it may cover only one) or strict (then it may cover several).

The conjunct for graph g has four parts. The first subformula encodes the conditions for weakly orientation by multiset ordering (see Section 4). The second subformula expresses a strict covering. The third subformula specifies that $\gamma^g_{i,j}$ and ε^g_i agree with their intended meaning. The fourth subformula states that if p_i is selected and ε^g_i indicates weak cover, then position i covers exactly one position j.

$$\psi^{\mathcal{G}}_{ms} = \bigwedge_{g=p(\bar{x}) \coloneq \pi;q(\bar{y})} \left(\begin{array}{l} weak^g_{ms} \leftrightarrow \bigwedge\limits_{1\leq j\leq m} \left(q_j \rightarrow \bigvee\limits_{1\leq i\leq n} \gamma^g_{i,j} \right) \quad \wedge \\ \\ strict^g_{ms} \leftrightarrow \bigvee\limits_{1\leq i\leq n} (p_i \wedge \neg\varepsilon^g_i) \quad \wedge \\ \\ \bigwedge\limits_{\substack{1\leq i\leq n \\ 1\leq j\leq m}} \gamma^g_{i,j} \rightarrow p_i \wedge q_j \wedge e^g_{x_i\geq y_j} \wedge (\neg\varepsilon^g_i \leftrightarrow e^g_{x_i>y_j}) \quad \wedge \\ \\ \bigwedge\limits_{1\leq i\leq n} p_i \rightarrow \varepsilon^g_i \rightarrow \oplus \left\{ \gamma^g_{i,j} \,\middle|\, 1 \leq j \leq m \right\} \end{array} \right)$$

Encoding the min and dual-multiset orderings: The encodings are obtained through the respective dualities with the max and multiset orderings.

$$\psi^{\mathcal{G}}_{min} = \bigwedge_{g=p(\bar{x}) \;:\!-\; \pi;q(\bar{y})} \begin{pmatrix} weak^{g}_{min} \leftrightarrow weak^{g^t}_{max} & \wedge \\ strict^{g}_{min} \leftrightarrow strict^{g^t}_{max} & \end{pmatrix}$$

$$\psi^{\mathcal{G}}_{dms} = \bigwedge_{g=p(\bar{x}) \;:\!-\; \pi;q(\bar{y})} \begin{pmatrix} weak^{g}_{dms} \leftrightarrow weak^{g^t}_{ms} & \wedge \\ strict^{g}_{dms} \leftrightarrow strict^{g^t}_{ms} & \end{pmatrix}$$

We have implemented the algorithm in Prolog. After creating the partial order constraints, our program calls the solver described in [9]. This solver transforms the partial order constraint to a CNF which is passed on to the MiniSAT solver for Boolean satisfaction [20] through its Prolog interface described in [7].

6 Experimentation

We have tested our implementation on two benchmark suites. The first is described in [4]. It originates from a logic programming test suite for termination analysis and consists of 123 examples (abstract programs). The second suite originates in a benchmark suite for termination of term rewriting systems (TPDP version 4.0 [25]). It consists of 4062 abstract programs generated when AProVE, a tool to automatically prove termination of term rewriting systems, applied the SCT method based on the embedding order, as described in [24]. The first suite can be obtained via Amir Ben-Amram's web page. The second can be found at `http://www.cs.bgu.ac.il/~mcodish/Software/trs_suite_for_sct.zip`

For the first suite, 84 of the 123 examples are SCT positive. All of these are also in SCP and, with no surprises, also in SCNP. The SAT based implementation is much slower than the implementations for SCP and SCT described in [4]. However, analysis times are reasonable (under 3 seconds for the entire suite) and we have the benefit that ranking functions are provided for the SCNP instances.

For the second suite, 3820 of the 4062 examples are in SCT. There is only one example which is SCT but not SCP. In fact, this is Example 7 which was designed precisely to defeat SCP. Here too SCNP agreed with SCP. Again ranking functions are provided for all the verified instances and the entire suite is analyzed in approximately 20 seconds. Our implementation is not optimized and analysis times are only reported to give an idea of their magnitude and show that the use of SAT solving is not prohibitive.

Acknowledgment

We thank Samir Genaim for help with the benchmarking and Peter Schneider-Kamp for harvesting the collection of 4062 sets of size-change graphs.

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 *LNCS*. Springer, 2006.

2. Amir M. Ben-Amram. A complexity tradeoff in ranking-function termination proofs. submitted for publication, 2007.
3. Amir M. Ben-Amram and Chin Soon Lee. Ranking functions for size-change termination II. 9th International Workshop on Termination (WST 2007), July 2007.
4. Amir M. Ben-Amram and Chin Soon Lee. Size-change analysis in polynomial time. *ACM Transactions on Programming Languages and Systems*, 29(1), 2007.
5. M. Bruynooghe, M. Codish, J.P. Gallagher, S. Genaim, and W. Vanhoof. Termination analysis of logic programs through combination of type-based norms. *ACM TOPLAS*, 29(2), 2007.
6. M. Codish, V. Lagoon, and P. J. Stuckey. Testing for termination with monotonicity constraints. In Maurizio Gabbrielli and Gopal Gupta, editors, *ICLP 2005*, volume 3668 of *LNCS*, pages 326–340. Springer, 2005.
7. Michael Codish, Vitaly Lagoon, and Peter J. Stuckey. Logic programming with satisfiability. *Theory and Practice of Logic Programming*, 2007. (To appear).
8. Michael Codish and Cohavit Taboch. A semantic basis for termination analysis of logic programs. *The Journal of Logic Programming*, 41(1):103–123, 1999.
9. M. Codish, V. Lagoon, and P. J. Stuckey. Solving partial order constraints for LPO termination. In *Proc. RTA '06*, volume 4098 of *LNCS*, pages 4–18, 2006.
10. B. Cook, A. Podelski, and A. Rybalchenko. Abstraction refinement for termination. In *Proc. SAS '05*, pages 87–101, 2005.
11. B. Cook, A. Podelski, and A. Rybalchenko. Termination proofs for systems code. In M. Schwartzbach and T. Ball, editors, *Proc. PLDI*, pages 415–426. ACM, 2006.
12. Lee C.S. ranking functions for size-change termination. submitted, 2007.
13. 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.
14. Nachum Dershowitz and Zohar Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979.
15. N. D. Jones and N. Bohr. Termination analysis of the untyped lambda calculus. In *Proc. RTA'04*, volume 3091 of *LNCS*, pages 1–23. Springer, 2004.
16. A. Krauss. Certified size-change termination. In Frank Pfenning, editor, *Proc. CADE*, volume 4603 of *LNAI*, pages 460–475. Springer-Verlag, July 2007.
17. C.S. Lee, N.D. Jones, and A.M. Ben-Amram. The size-change principle for program termination. In *Proc. POPL'01*, volume 28, pages 81–92. ACM press, Jan. 2001.
18. N. Lindenstrauss, Y. Sagiv, and A. Serebrenik. Termilog: A system for checking termination of queries to logic programs. In Orna Grumberg, editor, *Proc. CAV '97*, volume 1254 of *LNCS*, pages 444–447. Springer, 1997.
19. P. Manolios and D. Vroon. Termination analysis with calling context graphs. In *Proc. CAV '06*, volume 4144 of *LNCS*, pages 401–414. Springer-Verlag, 2006.
20. MiniSAT solver. `http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat`. Viewed December 2005.
21. G.C. Necula. Proof-carrying code. In *Proc. POPL*, pages 106–119. ACM, 1997.
22. P. Schneider-Kamp, R. Thiemann, E. Annov, M. Codish, and J. Giesl. Proving termination using recursive path orders and sat solving. In B. Konev and F. Wolter, editors, *FroCos*, volume 4720 of *LNCS*, pages 267–282. Springer, 2007.
23. D. Sereni and N. Jones. Termination analysis of higher-order functional programs. In *Proc. APLAS*, volume 3780 of *LNCS*, pages 281–297. Springer-Verlag, 2005.
24. R. Thiemann and J. 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.
25. The termination problem data base. `http://www.lri.fr/~marche/tpdb/`.