Size-Change Termination, Monotonicity Constraints and Ranking Functions

Amir M. Ben-Amram

Academic College of Tel-Aviv Yaffo

Abstract. Size-change termination involves deducing program termination based on the impossibility of infinite descent. To this end we may use a program abstraction in which transitions are described by *monotonicity constraints* over (abstract) variables. When only constraints of the form $x > y'$ and $x \geq y'$ are allowed, we have size-change graphs, for which both theory and practice are now more evolved then for general monotonicity constraints. This work shows that it is possible to transfer some theory from the domain of size-change graphs to the general case, complementing and extending previous work on monotonicity constraints. Significantly, we provide a procedure to construct explicit global ranking functions from monotonicity constraints in singly-exponential time, which is better than what has been published so far even for size-change graphs. We also consider the integer domain, where general monotonicity constraints are essential because the domain is not well-founded.

1 Introduction

This paper is concerned with termination analysis. This is a fundamental and much-studied problem of software verification, certification and transformation. A subproblem of termination analysis is the construction of *global ranking functions*. Such a function is required to decrease in each step of a program (for "step" read basic block, function call, etc, as appropriate); an explicitly presented ranking function whose descent is (relatively) easy to verify is a useful *certificate* for termination and may have other uses, such as running-time analysis.

A structured approach is to break the termination problem for programs into two stages, an *abstraction* of the program and analysis of the abstract program. One benefit of this approach is that the abstract programs may be rather independent of the concrete programming language. Another one is that the termination problem for the abstract programs may be decidable.

Size-change termination (SCT [9]) is such an approach. It views a program as a transition system with states. The abstraction consists in forming a *control-flow graph* for the program, identifying a set of *state variables*, and forming a finite set of *size-change graphs* that are abstractions of the transitions of the program. In essence, a size-change graph is a set of inequalities between variables of the source state and the target state. Thus, the SCT abstraction is an example of a transition system defined by constraints of a particular type. The technique

A. Bouajjani and O. Maler (Eds.): CAV 2009, LNCS 5643, pp. 109–123, 2009.
© Springer-Verlag Berlin Heidelberg 2009

concentrates on well-founded domains and on the impossibility of infinite descent. Thus, only two types of inequalities were admitted into the constraints in [9]: $x > y'$ (old value of x greater than new value of y) and $x \geq y'$.

Size-change graphs lend themselves to a very natural generalization: Monotonicity Constraints. Here, a transition may be described by any combination of order relations, including equalities as well as strict and non-strict inqualities, and involving any pair of variables from the source state and target state. Thus, it can express a relation among source variables, that applies to states in which the transition may be taken; a relation among the target variables, which applies to states which the transition may produce; and, as in SCT, relations involving a source variable and a target variable, but here equalities can be used, as well as relations like $x \leq x'$, that is, an increase.

The *Monotonicity Constraint Systems* treated in this paper will include another convenience, *state invariants* associated with a point in the control-flow graph. These too are conjunctions of order constraints.

Monotonicity constraint systems generalize the SCT abstraction and are clearly more expressive. It may happen that analysis of a program yields monotonicity constraints which are not size-change graphs; in such a case, simply approximating the constraints by a size-change graph may end up missing the termination proof. Specific examples appear in the next section. It is not surprising, perhaps, that Monotonicity Constraints actually predated the SCT framework—consider the Prolog termination analyses in [5, 10]. But as often happens in science, concentrating on a simplified system that is sufficiently interesting was conductive to research, and thus the formulation of the SCT framework led to a series of interesting discoveries. To pick up some of the salient points:

- The SCT abstraction has a simple semantics, in terms of transition systems.
- It has an appealing combinatorial representation as a set of graphs.
- A termination criterion has been formulated in terms of these graphs (the existence of an infinite descending thread in every infinite multipath [9]).
- This criterion is equivalent to the termination of every model (transition system)—in logical terms, this condition is sound and complete [8, 9].
- Termination of a set of size-change graphs can be effectively decided; while having exponential worst-case time, the method is often usable.
- It has been established that a global ranking function can also be effectively constructed from the size-change graphs. Lee [8] gave the first proof, where the size of the resulting ranking expression is up to triply-exponential. This left open the challenging problem of improving this upper bound. Progress regarding certain special cases is reported in [2].

Which of the desirable features are preserved if we move to the stronger framework of monotonicity constraints? The first contribution of this paper is an answer: *in essence, all of them.*

The second contribution of this paper is an algorithm to verify termination of a monotonicity constraint system while constructing a global ranking function, all in singly-exponential time. Thus, we solve the open problem from [8], and, surprisingly, by tackling a super-problem.

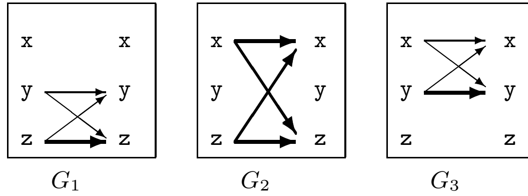

Fig. 1. SCT example with a complex ranking function. There is a single flow-point. Heavy arcs represent strict descent.

To illustrate this result, Figure 1 shows an SCT instance not falling into the classes handled in [2]. A ranking function for this instance, of the kind that we derive in this paper, follows. Its codomain is triples, and it descends lexicographically in every transition.

$$\rho(x, y, z) = \begin{cases} \langle x, y, z\rangle & \text{if} \quad z < x \leq y \vee y < x \leq z \\ \langle y, y, z\rangle & \text{if} \quad z < y < x \vee x \leq y \leq z \\ \langle z, y, z\rangle & \text{if} \quad x \leq z < y \vee y \leq z < x \end{cases}$$

Related Work. An important precursor of this work is [4]. It concentrated on local ranking functions, and made the intriguing observation that the termination test used in [10, 5] is sound and complete for SCT, but incomplete for general monotonicity constraints. It also presented the correct test (see Section 4.3). Even earlier, [7] presented a termination analysis for logic programs in which the correct decision algorithm for monotonicity constraint systems is embedded.

There are many works that target the discovery of relations among variables in a program. Classic examples include [3] for logic programs and [6] for imperative programs. All these techniques can be used to discover monotonicity constraints—none are inherently restricted to size-change graphs.

2 Basic Definitions and Examples

This section introduces monotonicity constraint systems (MCS) and their semantics, and formally relates them to SCT.

A monotonicity constraint system is an abstract program. An abstract program is, essentially, a set of *abstract transitions*. An abstract transition is a relation on (abstract) program states.

When describing program transitions, it is customary to mark the variables in the *resulting* state with primes (e.g., x'). We use S' to denote the primed versions of all variables in a set S. For simplicity, we will name the variables $x_1, \ldots, x_n$ (regardless of what program point we are referring to).

Definition 2.1 (MCS). *A* monotonicity constraint system*, or MCS, is an abstract program representation that consists of a control-flow graph (CFG), monotonicity constraints and state invariants, all defined below.*

- *A control-flow graph is a directed multigraph over the set F of* flow points*.*
- *A* monotonicity constraint *or MC is a conjunction of order constraints of the form $x \bowtie y$ where $x, y \in \{x_1, \ldots, x_n, x'_1, \ldots, x'_n\}$, and $\bowtie \in \{>, \geq, =\}$.*
- *Every CFG arc $f \to g$ is associated with a monotonicity constraint G. We write $G : f \to g$.*
- *For each $f \in F$, there is an* invariant I_f*, which is a conjunction of order constraints among the variables.*

The terms "abstract program", "constraint system" and "MCS instance" are used interchangeably, when context permits. The letter $\mathcal{A}$ is usually used to denote such a program; F^A will be its flow-point set. When notions of connectivity are applied to $\mathcal{A}$ (such as, "$\mathcal{A}$ is strongly connected"), they concern the underlying CFG.

Definition 2.2 (constraints as graphs). *The graph representation of $G : f \to g$ is a labeled graph $(X \cup Y, E)$ with the nodes $X = \{x_1, \ldots, x_n\}$ and $Y = X'$ and E includes a labeled arc for each constraint:*

– For a constraint $x > y$ (respectively $x \geq y$), an arc $x \xrightarrow{\downarrow} y$ ($x \xrightarrow{\overset{=}{\downarrow}} y'$).

– For a constraint $x = y$, an edge *(undirected arc) $x \to y$ (thus, G is a* mixed graph*)*[1].

The labeled arcs are referred to, verbally, as strict and non-strict and the edges are also called no-change arcs.

We remark that the lack of direction in edges is meaningful when we consider paths of size-change arcs. Note also that arcs may connect two source variables, two target variables or a source and a target variable—in any direction.

Henceforth, we identify a MC with its graph representation: it is a graph and a logical conjunction of constraints at the same time.

Definition 2.3 (semantics). *Let Val be a fixed, infinite well-ordered set. A* state *of $\mathcal{A}$ (or an abstract state) is $s = (\mathtt{f}, \sigma)$, where $\mathtt{f} \in F^A$ and $\sigma : \{1, \ldots, n\} \to$ Val represents an assignment of values to f's variables. A transition is a pair of states. The truth value of an expression such as $x_1 > x_2$ under σ is defined in the natural way.*

For $G : f \to g \in \mathcal{A}$, we write $G((f, \sigma) \mapsto (g, \sigma'))$ if all constraints in I_f, I_g and G are satisfied by σ and σ'. In this case, transition $(f, \sigma) \mapsto (g, \sigma')$ is described by G*.*

We say that G is unsatisfiable *if it describes no transition.*

The transition system *associated with $\mathcal{A}$ is the relation $T_{\mathcal{A}}$ defined by*

$$(s, s') \in T_{\mathcal{A}} \iff G(s \mapsto s') \text{ for some } G \in \mathcal{A}.$$

Definition 2.4 (termination). *A* run *of $\mathcal{T}_{\mathcal{A}}$ is a (finite or infinite) sequence of states $\tilde{s} = s_0, s_1, s_2 \ldots$ such that for all i, $(s_i, s_{i+1}) \in \mathcal{T}_G$. Transition system $\mathcal{T}_{\mathcal{A}}$ is* uniformly terminating *if it has no infinite run.*

[1] Hopefully, the use of $\to$ to denote both types of arcs will not confuse the reader.

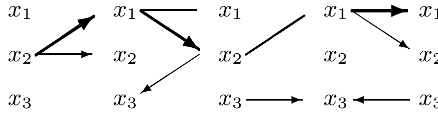

Fig. 2. A multipath

MCS $\mathcal{A}$ is said to be *terminating* if $\mathcal{T}_\mathcal{A}$ is uniformly terminating for any choice of *Val.*

Definition 2.5 (size-change graph). *A size-change graph (SCG) is a monotonicity constraint consisting only of relations of the forms $x_i > x'_j$ and $x_i \geq x'_j$.*

An SCT instance *is a MCS where all constraints are size-change graphs and all invariants are trivial.*

Let $P(s, s')$ be any predicate over states s, s', possibly written using variable names, e.g., $x_1 > x_2 \wedge x_2 < x'_2$. We write $G \models P$ if $G(s \mapsto s') \Rightarrow P(s, s')$.

Definition 2.6. *A (global)* ranking function *for a transition system $\mathcal{T}$ with state space St is a function $\rho : St \to W$, where W is a well-founded set, that decreases on every transition.*

A ranking function for a MCS $\mathcal{A}$ is a ranking function for $T_\mathcal{A}$. Equivalently, it satisfies $G \models \rho(s) > \rho(s')$ for every $G \in \mathcal{A}$.

Example 2.1 ([4]). Consider an abstract program with a single flow-point f, a trivial invariant, and a single abstract transition G: $x_1 > x'_2 \wedge x_2 \geq x_3 \wedge x'_1 \leq x'_3$.

This system is terminating; a possible proof is to note that the following *ranking function* descends (lexicographically) in every possible transition:

$$f(x_1, x_2, x_3) = \begin{cases} \langle 1, x_1 \rangle \text{ if } x_2 \geq x_3 \\ \langle 0, x_1 \rangle \text{ if } x_2 < x_3. \end{cases}$$

Clearly, G is not a size-change graph. Its best approximation as a size-change graph is $\{x_1 > x'_2\}$ which does not prove termination.

Example 2.2. Change G into $x_2 \geq x_3 \wedge x_3 = x'_3 \wedge x'_2 > x'_3$. Now, termination follows, not from the impossibility of infinite descent, but from the unsatisfiability of $G(s \mapsto s') \wedge G(s' \mapsto s'')$.

We remark that the issue of unsatisfiability never arises with SCT instances.

3 Multipaths, Walks and Termination

Definition 3.1. *A* multipath *M in $\mathcal{A}$ (an $\mathcal{A}$-multipath) is a finite or infinite sequence of MC graphs $G_1 G_2 \ldots$ that label a (finite or infinite) path in the CFG of $\mathcal{A}$.*

Most often, we view a multipath as the single (finite or infinite) graph obtained by concatenating the sequence of MC graphs, i.e., identifying the target nodes of G_i *with the source nodes of* G_{i+1} *(Figure 2). In this way it can also be seen as a conjunction of constraints on a set of variables which correspond to the nodes of the combined graph.*

Thus a multipath is a mixed graph. We will consider walks in this graph; a walk can cross an undirected arc in both directions. Recall also that walks are allowed to repeat nodes and arcs.

Definition 3.2. *A walk that includes a strict arc is said to be* descending*. A walk that includes infinitely many strict arcs is* infinitely descending*.*

An MCS is size-change terminating *if every infinite multipath has an infinitely descending walk.*

Note that the walk above may actually be a cycle! In this case it is contained in a finite section of the multipath, and, logically, it implies the condition $x > x$ for some variable x. Thus, such a multipath is unsatisfiable and corresponds to no concrete run. If the walk is not a cycle, it indicates an infinite descending chain of values and this contradicts well-foundedness. Thus, we see that if $\mathcal{A}$ is *size-change terminating* it can have no infinite runs. This proves the **if** direction of the next theorem.

Theorem 3.3. $\mathcal{A}$ *is terminating if and only if it is size-change terminating.*

Proof. (**only if**) suppose that $\mathcal{A}$ is not size-change terminating. Hence, an infinite multipath M can be formed, without infinite descent. Consider the infinite set $\mathcal{V}$ of the nodes of M as distinct variables; our aim is to show that there is a choice of *Val* such that these variables can all be assigned while satisfying all the constraints. This assignment will form an infinite run.

In fact, $\mathcal{V}$ itself is partially quasi-ordered by the constraints in M; more precisely, the constraints can be *completed* to a partial quasi-order by including additional inequalities, to satisfy the reflexive axiom $v = v$ for all v, and the various other axioms: $x > y \wedge y \geq z \Rightarrow x > z, \quad x \geq y \wedge y \geq x \Rightarrow x = y, \quad$ etc.

The closure ensures transitivity and symmetry of $=$ as well as transitivity of $>$, and the agreement of $\geq$ with $>$ and $=$. The asymetry of $>$ is guaranteed by the non-existence of a descending cycle.

Moreover, the partial quasi-order is well founded, because of the non-existence of infinite descent in M. Now, let *Val* be $\mathcal{V}/{=}$, which is a well-founded partial order, and extend the ordering to a total one in an arbitrary way while preserving well-foundedness. □

The SCT condition [9] is similar to the infinite-descent condition, but only concerns walks that proceed forwards in the multipath. Obviously, with SCT graphs, there are no other walks anyway.

Definition 3.4. *Let* $M = G_1 G_2 \ldots$ *be a multipath. A thread in* M *is a walk that only includes arcs in a forward direction* ($x_i \to x'_j$)*. A MCS* $\mathcal{A}$ satisfies SCT *if every infinite* $\mathcal{A}$*-multipath has an infinitely descending thread.*

As the examples of Section 1 show, the SCT condition, while clearly a sufficient condition for termination, is not a necessary one with respect to general monotonicity constraint systems.

4 Fully Elaborated Systems and Stability

In this section we describe a procedure that while increasing the size of an abstract program (by duplicating flow points), simplifies its termination proof. In order to express the correctness of a transformation on abstract programs we begin by defining "simulation."

4.1 Simulation

Definition 4.1. *Let $\mathcal{T}$, $\mathcal{S}$ be transition systems, with flow-point sets F, F^S respectively, and both having states described by n variables over Val. We say that $\mathcal{S}$* simulates *$\mathcal{T}$ if there is a relation $\phi \subseteq F \times F^S$ and, for all $(f,g) \in \phi$, a bijection $\psi_{f,g} : \{1,\ldots,n\} \to \{1,\ldots,n\}$ such that for every (finite or infinite) state-transition sequence $(f_1,\sigma_1) \mapsto (f_2,\sigma_2) \mapsto (f_3,\sigma_3) \mapsto \ldots$ of $\mathcal{T}$ there is a corresponding sequence $(g_1,\sigma'_1) \mapsto (g_2,\sigma'_2) \mapsto (g_3,\sigma'_3) \mapsto \ldots$ of $\mathcal{S}$ with $(f_i,g_i) \in \phi$ and $\sigma'_i = \sigma_i \circ (\psi_{f_i,g_i})$.*

Definition 4.2. *We say that an abstract program $\mathcal{A}$ simulates an abstract program $\mathcal{B}$ if $\mathcal{T}_\mathcal{A}$ simulates $\mathcal{T}_\mathcal{B}$, via mappings ϕ and ψ, as above.*

We say that $\mathcal{A}$ simulates $\mathcal{B}$ deterministically *if for every $f \in F^B$ and assignment σ' satisfying I_f there is a unique $g \in F^A$ with $(f,g) \in \phi$ such that, letting $\sigma' = sigma \circ (\psi_{f,g})$, assignment σ' satisfies I_g.*

Briefly, determinism means that the invariants of different $\mathcal{A}$ flow-points that simulate a given $\mathcal{B}$ flow-point have to be mutually exclusive (for valid states).

4.2 Elaboration

Definition 4.3. *An MCS $\mathcal{A}$ is* fully elaborated *if the following conditions hold:*

(1) Each state invariant fully specifies the relations among all variables. That is, for $i,j \le n$, one of $x_i = x_j$, $x_i < x_j$ or $x_i > x_j$ is implied by I_f.

(2) Each size-change graph G is closed under logical consequence (taking the invariants into account). That is, whenever $G((f,\sigma) \mapsto (g,\sigma')) \Rightarrow x \bowtie y$, for $\bowtie \in \{>,\geq,=,\}$, the corresponding arc is included in G.

(3) No MC in $\mathcal{A}$ is unsatisfiable.

In a fully elaborated system, the invariants can be assumed to have the form

$$x_1 \left\{ \substack{<\\=} \right\} x_2 \left\{ \substack{<\\=} \right\} \ldots \left\{ \substack{<\\=} \right\} x_n. \tag{1}$$

In this form, the variables are re-indexed according to the sorted order of their value; this has some convenient consequences. In particular, the closure under

logical consequence yields a "downward closure" of the graphs: if $x_i \to x'_j \in G$ then $x_i \to x'_k \in G$ for every $k \le j$, and if $x_i \stackrel{\downarrow}{\to} x'_j \in G$ then $x_i \stackrel{\downarrow}{\to} x'_k \in G$.

The number of possible orderings of n variables plays a role in the combinatorics of fully elaborated instances. Note that equalities are possible. Therefore, the number of orderings is not $n!$, but a slightly larger number called *the nth ordered Bell number* B_n. An easily proved upper bound is $B_n \le 2n^{n-1}$. We denote the set of these orderings by Bell_n.

Recall that $f = \tilde{O}(g)$ means that $f = O(g \cdot \log^k g)$ for some constant k. Thus $\tilde{O}(n^n)$ is asymptotically dominated by n^n times a polynomial in n.

Lemma 4.4. *Any MCS $\mathcal{B}$ with n variables at any point, and m flow-points, can be transformed into a fully-elaborated system $\mathcal{A}$, simulating $\mathcal{B}$ deterministically, in $\tilde{O}(mn^n)$ time and space.*

If $\mathcal{B}$ terminates, so does $\mathcal{A}$.

Proof. The algorithm for the transformation follows almost immediately from the definitions: first, for every $f \in F^B$, generate flow-points f_π where π ranges over Bell_n. A mapping $\psi_{f_\pi,f}$ is defined according to the ordering, to satisfy (1). The invariant I_{f_π} is also set to express the chosen ordering.

Next, for every MC $G : f \to g$ in $\mathcal{B}$, and every pair f_π, g_ϖ, create a size-change graph $G_{\pi,\varpi} : f_\pi \to g_\varpi$ as follows:

1. For every arc $x \to y \in G$, include the corresponding arc in $G_{\pi,\varpi}$, according to the variable renaming used in the two $\mathcal{A}$ flow-points.
2. Complete $G_{\pi,\varpi}$ by closure under consequences of the inequalities expressed by the given arcs, I_{f_π} and I_{g_ϖ}. Calculating this closure is easy since it is actually an All-Pairs Shortest-Path problem with weights in the set $\{\downarrow, \overset{=}{\downarrow}, =\}$. This can be done in polynomial time, at most $O(n^3)$, by a standard algorithm. Note that the algorithm will also find out whether the graph is satisfiable—it only fails to be so if the closure includes a strict self-loop $x > x$.

   Unsatisfiable graphs are removed from the constructed system.

If $\mathcal{B}$ terminates, so does $\mathcal{A}$—because every run of $\mathcal{A}$ represents a run of B. □

4.3 Stability

Definition 4.5. *An MC $G : f \to g$ is* stable *if it is closed under logical consequence, and, moreover, every relation among variables of state s (respectively, s') implied by $I_f(s) \wedge G(s, s') \wedge I_g(s')$ is already implied by I_f (respectively, I_g).*

MCS $\mathcal{A}$ is stable if all its abstract transitions are.

It is easy to see that a fully elaborated system is stable. Full elaboration can be seen as a brute-force solution; a system can be stabilized by an iterative fixed-point computation, which is likely to end up with fewer than B_n times every flow point. We do not elaborate further on the algorithm in this abstract.

Theorem 4.6. *A stable MCS $\mathcal{A}$ is terminating if and only if it satisfies SCT.*

Proof. If $\mathcal{A}$ satisfies SCT, it is terminating. For the converse direction, Let $M = G_1 G_2 \dots$ be an infinite $\mathcal{A}$-multipath. We know that it has an infinitely descending walk. We now have two cases, as the walk is either a cycle or extends to infinity. We claim that the first case cannot happen (and leave the proof to the reader; it uses the same ideas as the other case). In the second case, we shall prove that there is an infinitely descending thread.

For unique naming, let the nodes of the multipath be labeled $x[t,i]$ such that $x[t,i]$ is a source node of G_{t+1} representing x_i (and a target node of G_t). The index t is called the time coordinate. The walk is made out of arcs $x[t_k, i_k] \to x[t_{k+1}, i_{k+1}]$ for $k = k_0, k_0+1, \dots$ Let j_t be the first occurrence of t in the sequence; note that this is well defined for all $t \geq t_{k_0}$. The walk is broken into segments leading from $x[t, i_{j_t}]$ to $x[t+1, i_{j_{t+1}}]$, of which infinitely many are descending. We claim that each of this segments can be replaced with a single arc, strict when appropriate. This implies that SCT is satisfied.

As there is a walk from $x[t, i_{j_t}]$ to $x[t+1, i_{j_{t+1}}]$, consider the shortest one (the shortest strict one, if appropriate). We claim that it consists of one arc. For if it does not, consider a node $x[t^*, i_{j_{t^*}}]$ of smallest time coordinate that occurs inside the segment (not at its ends). Then consider the two arcs—one that enters that node and one that exits; they must both be in $G_{j_{t^*}+1}$. By consequence closure, there is a single arc (strict if any of the two arcs is) to replace these two arcs. Thus, the presumed shortest walk is not shortest. □

A decision algorithm. Theorem 4.6 has the immediate consequence of an algorithm to decide MCS termination. Namely, stabilize the system and apply an SCT algorithm. Note that for deciding SCT, we can ignore any "backward" arcs ($x'_j \to x_i$), as well as the state invariants, in other words retain just SCT graphs. This observation may possibly be useful in optimizing an implementation, in particular in conjunction with a subsumption test.

Anothe natural expectation is that it would be desirable in practice to avoid full elaboration when possible, as already remarked. The emphasis on "in practice" is due to the fact that when the theoretical worst-case complexity is considered, full elaboration may (somewhat paradoxically) be an improvement, specifically, when the decision procedure used is the closure-based algorithm [9]. The closure set of an SCT instance can reach (at the worst case) a size of $m^2\, 2^{\Theta(n^2)}$; the closure set of a fully-elaborated system is only $m^2\, 2^{O(n \log n)}$. However, as the next section will show, once we have fully elaborated the system, there is actually a polynomial-time algorithm that decides termination (and more), so there is no need to do anything as costly as a closure computation.

The decision algorithm for MCS that is embedded in the logic-program analysis of [7] is a "closure type" algorithm similar to the standard algorithm for SCT. It seems that the algorithm can be interpreted as applying the standard SCT algorithm to a stabilized constraint system (disregarding aspects that do not apply to our framework). Yet another similar algorithm is found in [4]. It uses "balancing" which is similar to our stabilization but on a local basis (consistent with the overall local approach of that paper).

5 Constructing a Global Ranking Function

This section describes our ranking-function construction. The general form of the constructed function is a "case expression" which associates tuples of variables and constants with guards. The function evaluates to the value of the tuple whose guard matches the current state. For an example see Section 1.

5.1 Background

We first cite some definitions and results from previous work, specifically [1, 2].

Definition 5.1 (vectors). *For flow-point $f \in F$ and positive integer B, V_f^B is the set of tuples $\mathbf{v}_f = \langle v_1, v_2, \dots \rangle$ of even length, where every odd position is a variable of f, such that every variable appears once; and every even position is an integer between 0 and B.*

Definition 5.2 (thread preserver). *Given MCS $\mathcal{A}$, a mapping $P : F^A \to \mathcal{P}(\{1, \dots, n\})$ is called a* thread preserver *of $\mathcal{A}$ if for every $G : f \to g$ in $\mathcal{A}$, it holds that whenever $i \in P(f)$, there is $j \in P(g)$ such that $x_i \to x_j \in G$.*

It is easy to see that the set of thread preservers of $\mathcal{A}$ is closed under union. Hence, there is a unique maximal thread preserver, which we denote by $\mathrm{MTP}(\mathcal{A})$. Given a standard representation of $\mathcal{A}$, $\mathrm{MTP}(\mathcal{A})$ can be found in linear time [1].

Definition 5.3 (complete thread). *A thread in a given multipath is* complete *if it starts at the beginning of the multipath, and is as long as the multipath.*

Lemma 5.4. *If a strongly connected MCS satisfies SCT, every finite multipath includes a complete thread.*

5.2 Preliminaries

The essential idea of the construction is to first fully-elaborate the program, and then process the resulting constraint system. This works, as shown next:

Lemma 5.5. *If $\mathcal{A}$ simulates $\mathcal{B}$* deterministically, *any ranking function for $\mathcal{A}$ can be transformed into a ranking function for $\mathcal{B}$.*

Proof. Let ρ be the $\mathcal{A}$ ranking function. The $\mathcal{B}$ ranking function is defined for state (g, σ') as $\rho(f, \sigma)$ where f is the unique point satisfying $(f, g) \in \phi$ and $I_f(\sigma' \circ (\psi_{f,g}^{-1}))$, and $\sigma = \sigma' \circ (\psi_{f,g}^{-1})$. □

Note that, if ρ assigns a unique vector to each flow-point in $\mathcal{A}$, the resulting $\mathcal{B}$ ranking function has just the form described at the beginning of this section.

We assume, henceforth, that $\mathcal{A}$ is a fully-elaborated, terminating constraint system. By Theorem 4.6, $\mathcal{A}$ satisfies SCT, and can be handled as an SCT instance.

Definition 5.6. *A variable x_i is called* thread-safe *at flow-point f if every finite $\mathcal{A}$-multipath, starting at f, includes a complete thread from x_i.*

Lemma 5.7. *Assume that $\mathcal{A}$ is strongly connected. For every f, let $S(f)$ be the set of indices of variables that are thread-safe at f. Then $S(f)$ is not empty for any $f \in F^A$ and S is a thread preserver.*

Proof. Let M be any finite $\mathcal{A}$-multipath starting at f. Observe that since $\mathcal{A}$ satisfies SCT and is strongly connected, there must be a complete thread in M, say starting at x_i. But then x_n can also start a thread (note the downward-closure of fully-elaborated MCs). It follows that $n \in S(f)$.

We now aim to show that S is a thread preserver. Let $i \in S(f)$, and let $G : f \to g$. Every finite multipath M beginning with G has a complete thread that begins with an arc from x_i, say $x_i \to x'_{j_M}$. Let J be the set of all such indices j_M, and $k = \max J$. Then $x_i \to x_k$ is an arc of G, because $k \in J$; and by the downward-closure property one can see that every M has a complete thread beginning with the arc $x_i \to x'_k$. Hence, $k \in S(g)$ and the proof is complete. □

Definition 5.8 (freezer). *Let $C : F^A \to \{1, \dots, n\}$. Such C is called a* freezer *for $\mathcal{A}$ if for every $G \in \mathcal{A}$, $G \models x_{C(f)} = x'_{C(g)}$.*

Lemma 5.9. *Suppose that $\mathcal{A}$ is strongly connected, satisfies SCT, and has a freezer C. If for every f, variable $x_{C(f)}$ is ignored, SCT will still be satisfied.*

Proof. Let M be an infinite multipath of $\mathcal{A}$; by the SCT property, M has an infinitely descending thread τ. Observe that C induces one infinite thread ϑ in M, consisting entirely of no-change arcs $x_i \xrightarrow{=} x'_j$; this thread represents a sequence of unchanging data values in any transition sequence described by M, and therefore can have at most finitely many intersections with τ. It follows that M has an infinitely descending thread that avoids the frozen variables. □

Lemma 5.10. *Let $\mathcal{A}$ have thread preserver S, where $S(f) \neq \emptyset$ for all f. For every $f \in F^A$, let $i_f = \min S(f)$. Then every MC $G : f \to g$ includes $x_{i_f} \to x'_{i_g}$.*

Proof. G must have an arc from x_{i_f} to some $x_j \in S(g)$; so by consequence-closure, G includes $x_{i_f} \to x'_{i_g}$. □

5.3 Constructing the Ranking Function

To construct a ranking function for a general MCS, we begin by transforming it into a fully-elaborated $\mathcal{A}$ as described in Lemma 4.4. We then proceed with the construction in an incremental way. To justify the incremental construction, we define a residual transition system and relate it to ranking functions.

Definition 5.11. *Let $\mathcal{T}$ be a transition system with state space St. A* quasi-ranking function *for $\mathcal{T}$ is a function $\rho : St \to W$, where W is a well-founded set, such that $\rho(s) \geq \rho(s')$ for every $(s, s') \in \mathcal{T}$.*

The residual transition system relative to ρ, denoted $\mathcal{T}/\rho$, includes all (and only) the transitions of $\mathcal{T}$ which do not *decrease ρ.*

Observe that Lemma 5.10 provides a quasi-ranking function: $\rho(f, \sigma) = \sigma(x_{i_f})$.

The next couple of lemmas are quite trivial but we spell them out because they clarify how a ranking function may be constructed incrementally. We use the notation $v \cdot u$ for concatenation of tuples.

Lemma 5.12. *Assume that ρ is a quasi-ranking function for $\mathcal{T}$, and ρ' a ranking function for $\mathcal{T}/\rho$ whose range consists of lexicographically-ordered tuples; then $\rho \cdot \rho'$ is a ranking function for $\mathcal{T}$.*

Lemma 5.13. *Assume that the CFG of $\mathcal{A}$ consists of a set $C_1, \dots, C_k$ of mutually disconnected components (that is, there is no arc from C_i to C_j with $i \neq j$). If for every i, ρ_i is a ranking function for $\mathcal{A}$ restricted to C_i, then $\cup_i \rho_i$ is a ranking function for $\mathcal{A}$.*

Lemma 5.14. *Suppose that the CFG of $\mathcal{A}$ consists of several strongly connected components (SCCs). Let $C_1, \dots, C_k$ be a reverse topological ordering of the components. Define a function $\rho(s)$ for $s = (f, \sigma)$ as the index i of the component C_i including f. Then ρ is a quasi-ranking function (with co-domain $[1, k]$) and it is strictly decreasing on every transition represented by an inter-component arc.*

The following algorithm puts all of this together. Note: a CFG whose arc set is empty is called *vacant*. A strongly connected component whose arc set is empty is called *trivial* (it may have connections to other components).

Algorithm 5.1. (ranking function construction for $\mathcal{A}$)

1. List the SCCs of $\mathcal{A}$ in reverse-topological order. For each $f \in F^A$, let κ_f be the position of the SCC of f. Form $\mathcal{A}'$ by deleting all the inter-component transitions. If $\mathcal{A}'$ is vacant, return ρ where $\rho(f, \sigma) = \kappa_f$.
2. For each SCC C, compute the MTP, using the algorithm in [1]. If empty, report failure and exit.
3. For every f, let x_{i_f} be the lowest MTP variable of f.
4. For every graph $G : f \to g$, if it includes $x_{i_f} \xrightarrow{\downarrow} x'_{i_g}$, delete the graph from $\mathcal{A}'$; otherwise, retain the graph but delete (or hide) the node x_{i_f} and incident arcs.
5. For every f, let $\rho(f, \sigma) = \langle \kappa_f, \sigma(x_{i_f}) \rangle$.
6. If $\mathcal{A}'$ is now vacant, return ρ. Otherwise, compute a ranking function ρ' recursively for $\mathcal{A}'$, and return $\rho \cdot \rho'$.

We claim that the abstract program $\mathcal{A}'$, passed to the recursive call, always represents the residual transition system $\mathcal{T}_{\mathcal{A}}/\rho$. This should be clear when we delete graphs that strictly decrease ρ. The less-trivial point is the treatment of graphs G where the MTP arc $x_{i_f} \to x_{i_g}$ is non-strict (Step 4). To obtain the residual system precisely we should have kept x_{i_f} with no-change arcs $x_{i_f} \xrightarrow{=} x_{i_g}$. However, having done so, the variables x_{i_f} for every f become a freezer, and therefore can be dropped (Lemma 5.9).

Dropping the "frozen" variables ensures that these variables will not be used again in ρ'. So in the final tuple $\rho_f \cdot \rho'_f$, each variable will occur at most once.

Finally, the ranking function for the original $\mathcal{A}$ can be obtained according to Lemma 5.5. We summarize the conclusion in a theorem:

Theorem 5.15. *Let $\mathcal{B}$ be a terminating MCS, with m flow-points and n variables per point. There is a ranking function ρ for $\mathcal{B}$ where $\rho_f(\sigma)$ is a case expression with inequality constraints as guards and elements of V_f^m as values. The complexity of construction (as well as the size of the output) is $\tilde{O}(m \cdot n^n)$.*

Even when restricting attention to SCT instances, these results improve upon previous publications. The improvement over [2] is that any positive instance can be handled, and the bound B in V_f^B is reduced from about $m \cdot 2^n$ to m; the improvement over [8] is that in that work, the vectors were possibly doubly exponential in length (as a function of n) and the complexity of the construction was only bounded by a triply exponential function.

6 Monotonicity Constraints over the Integers

In practice, the integers are clearly the predominant domain for the constraints; when they represent the size of a list or tree they are necessarily non-negative, which allows the well-founded model to be used, but it has often been pointed out that in imperative programming languages, in particular, the crucial variables often are of integer type and might be negative (by design or by mistake). With the integer type, monotonicity constraints are still useful: they can prove that in a loop, two values keep approaching each other—which can also be expressed as showing that $x \leq y$ is an invariant and $y - x$ descends. In this section we adapt the ideas of the previous sections to the integer domain. We do not consider the domain of non-negative integers separately but note that by including the constant 0 as a "variable" and indicating its relation to every variable known to be non-negative we reduce the problem to the general integer case.

An intuitive reduction of the integer case to the well-founded case is to create a new variable for every difference $x_i - x_j$ which can be deduced from the MCs to be non-negative, and also deduce relations among these new variables to obtain a constraint system. But this solution may square the number of variables, which is bad because this number is in the exponent of the complexity, and completeness is not obvious(why shouldn't other linear combinations be necessary?). We will tackle the problem directly. Due to space limitations, this part will be rather terse.

6.1 Termination

We begin by formulating a termination condition in combinatorial terms. To this end we generalize the notion of a thread and define an up-thread to be a thread that consists of inverse arcs (so it indicates an ascending chain). A down-thread is an ordinary thread.

Condition S (for Stable MCS). In any infinite multipath (with variables $x[t, i]$ as in Sec. 4.3) there is a up-thread $(x[k, l_k])_{k=k_0,k_0+1,...}$ and a down-thread $(x[k, h_k])_{k=k_0,k_0+1,...}$ such that all the constraints $x[k, l_k] \le x[k, h_j]$ are present in the corresponding invariants. In addition, at least one of the threads has infinitely many strict arcs.

The condition for a general MCS is similar but uses walks instead of threads.

Condition G (for General MCS). In any infinite multipath (with variables $x[t, i]$) there is an infinite sequence of triples (t_j, l_j, h_j) such that the constraints in the multipath imply: $x[t_j, l_j] \le x[t_j, h_j]$, $x[t_j, l_j] \le x[t_{j+1}, l_{j+1}]$ and $x[t_j, h_j] \ge x[t_{j+1}, h_{j+1}]$. In addition, at least one of the walks $(x[t_j, l_j])_{j=1,2,...}$ and $(x[t_j, h_j])_{j=1,2,...}$ has infinitely many strict arcs.

Theorem 6.1. *Conditions* G *and* S *are equivalent for a stable system. Both are sound and complete criteria for termination of the respective transition systems.*

Condition S can be decided by a closure-based algorithm, which is essentially described in [4] (they consider non-negative integers but this difference is trivial).

6.2 Ranking Functions

Consider a fully elaborated system over the integers. If for every pair $x_i \le x_j$ we create a variable x_{ij} to represent $x_j - x_i$ (as it is non-negative) and we connect such variables with the obvious size-change arcs (if $x_i \le x'_l$ and $x_j \ge x'_h$ then $x_{ij} \ge x'_{lh}$) then Condition S implies ordinary size-change termination in the new variables. Thus a ranking function exists with such differences as elements of the vectors. Elaborating this system will put n^2 in the exponent. But it turns out that the order relations among the original variables lead to enough information about relations among the differences, that we can apply a version of the algorithm of Section 5. Thus, we can construct a global ranking function in $\tilde{O}(m \cdot n^n)$ time.

A final comment. In principle, we could abstract from the integers and just state the assumption that for any two elements there are only finitely many elements strictly between them. However, this abstraction buys us no generality, as every total order with this property is isomporphic to a subset of the integers.

7 Conclusion

We introduced the MCS abstraction, partly as background to our construction of ranking functions (which was initially developed with size-change graphs in mind), but also in order to explicate the ways in which our knowledge about SCT extends to this more expressive (and practically appealing) framework. This aspect should be seen as a call for further research, involving the application of monotonicity constraints.

The algorithms in this article were aimed at simplicity of presentation and analysis and can certainly be improved, for example by avoiding full elaboration.

For the decision procedure we know that this can be done (by stabilization), but creating practical applications of the procedure is a non-trivial challenge; more so for our ranking-function construction, where full elaboration was directly used in the proof, though it is clear that this does not mean that it is really always necessary.

We should point out that in the worst-case, the exponential behaviour cannot be beaten with this sort of global ranking functions. This is shown in [2], which actually provides a rather tight lower bound as it shows that $n!$ vectors may be needed for programs with n variables. On the bright side, our exponent is tight in that the algorithm is of complexity $m\,2^{O(n \log n)}$, like the lower bound, and unlike the $m^2 \cdot 2^{O(n^2)}$ upper bound of the closure algorithm.

Acknowledgments. The author thanks Chin Soon Lee for inspiration, and the LIPN laboratory at Université Paris 13 for hospitality.

References

[1] Ben-Amram, A.M., Lee, C.S.: Size-change analysis in polynomial time. ACM Transactions on Programming Languages and Systems 29(1) (2007)

[2] Ben-Amram, A.M., Lee, C.S.: Ranking functions for size-change termination II. Logical Methods in Computer Science (to appear, 2009)

[3] Brodsky, A., Sagiv, Y.: Inference of inequality constraints in logic programs. In: Proceedings of the Tenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), pp. 227–240. ACM Press, New York (1991)

[4] Codish, M., Lagoon, V., Stuckey, P.J.: Testing for termination with monotonicity constraints. In: Gabbrielli, M., Gupta, G. (eds.) ICLP 2005. LNCS, vol. 3668, pp. 326–340. Springer, Heidelberg (2005)

[5] Codish, M., Taboch, C.: 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. Springer, Heidelberg (1997)

[6] Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: Conference Record of the Fifth annual ACM Symposium on Principles of Programming Languages, pp. 84–96. ACM, New York (1978)

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

[8] Lee, C.S.: Ranking functions for size-change termination. ACM Transactions on Programming Languages and Systems (to appear, 2009)

[9] Lee, C.S., Jones, N.D., Ben-Amram, A.M.: The size-change principle for program termination. In: Proceedings of the Twenty-Eigth ACM Symposium on Principles of Programming Languages, vol. 28, pp. 81–92. ACM Press, New York (2001)

[10] Lindenstrauss, N., Sagiv, Y.: Automatic termination analysis of Prolog programs. In: Naish, L. (ed.) Proceedings of the Fourteenth International Conference on Logic Programming, Leuven, Belgium, pp. 64–77. MIT Press, Cambridge (1997)