Size-Change Termination Algorithm (the local method)

Amir M. Ben-Amram

MTA

April 2010
Definition

The composition of $G : f \rightarrow g$ and $G' : g \rightarrow h$ is $G ; G' : f \rightarrow h$ with arc set $E$ defined as follows. For a pair of parameters $x, y$, if multipath $GG'$ has a descending path from $x$ to $y$, then $E$ contains an arc $x \geq y$. Otherwise, if $GG'$ has any path from $x$ to $y$, $E$ contains $x \geq y$. Otherwise, there is no arc from $x$ to $y$.

Significance: whenever $G_c \models s \leftrightarrow s'$ and $G_c' \models s' \leftrightarrow s''$, we have $G_c ; G_c' \models s \leftrightarrow s''$.

Definition

$G^*$ (or, more properly, $G^+$) is the composition closure of $G$.

- Bound on $|G^*|
- Algorithm to compute it
- Significance of $G^*$
**Definition**
An MC/SCG/multipath is **cyclic** if its initial and final flow-points are the same.

**Definition**
Let $G$ be a cyclic MC/SCG. The **circular variant** $G^\circ$ is obtained by adding **shortcut edges** between every $x_i$ and $x'_i$.
The Test for SCGs

Let $G$ be a cyclic size-change graph. The following defs relate to $G^\circ$.

A cycle: a path commencing and ending at the same node.

Zig-zag cycle: traverses shortcut edges only in the backward direction ($x'_i$ to $x_i$).

Descending cycle: includes a strict arc.

SCG Termination Test

$G$ passes the Local Termination Test for SCGs, or Sagiv’s Test, if $G^\circ$ has a descending zig-zag cycle.

(How to implement the test?)

Theorem

An SCT instance $G$ is size-change terminating if and only if every cyclic SCG in $G^*$ passes the Local Termination Test.
For a cyclic $G$, we show that multipath $G^\omega$ has an infinite descending thread.

Let $x_{i_0}, x_{i_1}, \ldots$ be the nodes of the source side of $G$ that participate in the descending cycle in $G$. We map the cycle onto a thread in $G^\omega$: $x[0, i_0], x[1, i_1], x[2, i_2], \ldots, x[k, i_0]$ and repeat ad inf.

Every repeat has a strict arc.

Termination of $G$ now follows based on the local method.
Suppose that $G$ is size-change terminating; let $G$ be a cyclic SCG in $G^\ast$. Our aim: to show that $G$ passes the local test.

Consider the multipath $G^\omega$. Note that $G = \overline{M}$ for some finite $G$-multipath $M$. Let $\ell$ be the length of $M$. Consider the infinite $G$-multipath $M^\omega$, obtained by concatenating infinitely many copies of $M$ (blocks).

By assumption, $M^\omega$ has an infinite descending walk. Therefore, $G^\omega$ has one.

Since the walk is infinitely descending, some index $i$ has to occur twice on the walk, say at nodes $x[t, i]$ and $x[t', i]$ where $t \leq t'$, and there has to be a strict arc between them. Such a walk can be transformed into a zig-zag descending cycle in $G^\circ$, starting at $x_i$. 
1. SCT (termination for SCGs) is decidable. The decision problem is in PSPACE.

2. Simple, linear LRFs suffice for SCT instances.

3. Another explanation for the test (from Codish & Taboch): we test whether

\[ G \land (x_1 \leq x'_1) \land \cdots \land (x_n \leq x'_n) \]

is satisfiable (if it isn't: derive conclusions and proceed by the local principle).
The Test for MCs

Let $G$ be a cyclic MC. The following defs relate to $G^\circ$.

- **A cycle:** a path commencing and ending at the same node.
- **Forward cycle:** traverses shortcut edges more often in the backward direction ($x_i'$ to $x_i$) than it does in the forward direction.
- **Balanced cycle:** traverses shortcut edges equally often in both directions.
- **Descending cycle:** includes a strict arc.

### Local Termination Test

$G$ passes the *Local Termination Test*, or LTT, if $G^\circ$ has a descending cycle, either forward or a balanced.

(How to implement the test?)

### Theorem

*MCS $A$ is size-change terminating if and only if every cyclic MC in $A^*$ passes the Local Termination Test.*
Based on the *local method*, it suffices to show that for a cyclic $G$, multipath $G^\omega$ has an infinite descending walk.

Let $\nu_1, e_1, \nu_2, e_2, \ldots, e_{s-1}, \nu_s$ be the nodes and arcs (alternatingly) of the descending cycle in $G$. We map the cycle onto a walk in $G^\omega$.

The first node is $x[s, i_1]$. If the arc $e_1$ is an ordinary arc of $G$, the walk follows this arc to $x[s + 1, i_2]$, $x[s, i_2]$ or $x[s - 1, i_2]$.

If $e_1$ is a shortcut arc, the walk is not extended: $\nu_2$ is also mapped to $x[s, i_1]$ (necessarily $i_2 = i_1$).

We proceed in this manner until we complete the cycle and return to $\nu_1$. At this point, our walk will have reached a node $x[s', i_1]$ for some $s'$.

If the cycle is *balanced*, we have $s' = s$.

If the cycle is *forward*, we have $s' > s$.

In both cases, this can be continued *ad infinitum* (what is the significance of each case?)
Example

(a) \( x_1 \rightarrow x_1' \rightarrow x_2 \rightarrow x_3 \)
(b) \( x_1 \leftarrow x_1' \rightarrow x_2 \rightarrow x_3 \)
(c) \( x_1 \leftarrow x_1' \rightarrow x_2 \rightarrow x_3 \)
(d) \( x[1, 0] \rightarrow x[1, 1] \rightarrow x[1, 2] \rightarrow x[1, 3] \)
\( x[2, 0] \rightarrow x[2, 1] \rightarrow x[2, 2] \rightarrow x[2, 3] \)
\( x[3, 0] \rightarrow x[3, 1] \rightarrow x[3, 2] \rightarrow x[3, 3] \)
Suppose that $A$ is size-change terminating; let $G$ be a cyclic MC in $A^*$. Our aim: to show that $G$ passes the LTT.

Consider the multipath $G^\omega$. Note that $G = \bar{M}$ for some finite, satisfiable $A$-multipath $M$. Let $\ell$ be the length of $M$. Consider the infinite $A$-multipath $M^\omega$, obtained by concatenating infinitely many copies of $M$ (blocks).

By assumption, $M^\omega$ has an infinite descending walk. If this walk is a cycle, contained entirely within one of the blocks, then $M$ is unsatisfiable, and $\bar{M}$ will not appear in $A^*$. Thus, the walk must cross block boundaries.

Concentrate on the variables $x[t, i]$ occurring on these boundaries. Since the walk is infinitely descending, some index $i$ has to occur twice on the walk, say at nodes $x[t\ell, i]$ and $x[t'\ell, i]$ where $t \leq t'$, and there has to be a strict arc between them.
Completeness (cont.)

There is thus a descending walk from \( x[t\ell, i] \) to \( x[t'\ell, i] \). Such a walk can be transformed into a cycle \( C \) in \( G^\circ \), starting at \( x_i \). We progress by segments \( S \) that begin and end on a block boundary.

Case 1: \( S \) ends up in \( x[t + \ell, j] \). Extend \( C \) to \( x_j' \) then to \( x_j \).

Case 2: \( S \) ends up in \( x[t, j] \) for some \( j \neq i \). Then \( G \) includes either an arc \( x_i \rightarrow x_j \) or an arc \( x_i' \rightarrow x_j' \)...

Case 3: \( S \) ends up in \( x[t - \ell, j] \). Extend \( C \) to \( x_i' \) then to \( x_j \).

Finally we get to \( x[t'\ell, i] \) in the original walk, whereupon \( C \) will end at \( x_i \), becoming a cycle.

\( t' \geq t \) implies that the number of backward shortcuts in \( C \) does not exceed the number of forward shortcuts.
Conclusion

Size-change termination for MCs is decidable. The decision problem is in PSPACE.
Efficiency in practice

Naturally we do not use the PSPACE version, that is, we construct the closure. Means to improve efficiency:

1. Based on the CFG (SCC decomposition, contraction of $\rightarrow\bullet\rightarrow$)
2. Subsumption
The idea of procedure summaries

How to abstract a program with procedures?

- Let’s begin with non-recursive programs, which we represent as m.c. systems in the usual way

(drawing: program which calls p() several times, p() to its side)

- Easy solution: in-line the procedure(s)
  Disadvantages: CFG size; closure size; no. of variables

- Clever solution: procedure summaries
Algorithm

Suppose procedure $p$ calls procedure $q$

$q$ is represented by $MCS_q$ with entry point $s_q$ and exit point $e_q$

Compute closure of $MCS_q$

$SUMMARY(q) = \text{the set of MCs leading from } s_q \text{ to } e_q$

Now, inline $SUMMARY(q)$ in $MCS_p$.

With more procedures: process them bottom-up.
Recursive Procedures

Suppose procedure $p$ calls itself (for simplicity we consider simple recursion; mutual recursion is an easy generalization). We compute the closure of $p$ by an LFP procedure.

Let $S = \emptyset$. Let $C$ be the closure of $\text{MCS}_p$ where call transitions are ignored.

Repeat:

Let $S' \subseteq C$ be the set of MCs leading from $s_p$ to $e_p$.

If $S' = S$, exit the loop.

For each MC $G \in S' \setminus S$, consider $G$ to describe a possible effect of calling $p$, and accordingly add a copy of $G$, with the necessary renaming of flow-points and possibly variables, to $\text{MCS}_p$, in every recursive call site. Now recalculate $C$ (practically, just compute compositions involving the newly-added MCs until closure).

Set $S = S'$ and loop.

On exit, $C$ is the closure set for $p$ (which can be tested on the fly for termination), and $S$ is the procedure summary.
Abstract Semantics and the Closure Algorithm

Amir M. Ben-Amram

MTA

May 2010
A **semantics** maps programs (and program parts) into an **abstract domain**.

**Note:**
- All semantics are abstractions.
- There is no single semantics for a programming language.

The theory of Abstract Semantics (or Abstract Interpretation) ties semantics at different levels of abstraction.
A program $P$ is a collection of instructions.  
An instruction $e \in P$ is a relation on states: $s \xrightarrow{e} s'$ 
We define two semantics for $P$.

**Transition-System Semantics**

We define the transition semantics $\llbracket P \rrbracket_{step}$ to be the union of all these relations, over a set $S$ of states. I.e., the *transition relation* of $P$.

**Binary Reachability Semantics**

$\llbracket P \rrbracket_{bin}$ is the transitive closure of $\llbracket P \rrbracket_{step}$:

$$\llbracket P \rrbracket_{bin} = (\llbracket P \rrbracket_{step})^+.$$
\[ [P]^{step} \in \mathcal{P}(S \times S). \]

Operations on \( \mathcal{P}(S \times S) \): union \( \cup \), composition \( ; \)

Using the above, define operator \( F \):

\[
F(X) = X \cup [P]^{step} \cup (X; [P]^{step})
\]

Then \( [P]^{bin} = \text{lim}_{n \to \infty} F^n(\emptyset) \).

In general, this is not a finite object.
What semantics to use?
What semantics to use?

Proof using a GRF: $[P]^{step}$. 

Theorem

Suppose $T_1, \ldots, T_k$ are well-founded binary relations on $S$, and $R$ another relation on $S$ where $R + \subseteq T_1 \cup \cdots \cup T_k$.

Then $R$ is well-founded.

We prove that $[P]$ step is well-founded by analysing $[P]$ bin.
What semantics to use?

Proof using a GRF: $\llbracket P \rrbracket^{step}$.

Proof using the local method: $\llbracket P \rrbracket^{bin}$. 
Semantics and Termination Proofs

What semantics to use?

Proof using a GRF: $\llbracket P \rrbracket^{step}$.

Proof using the local method: $\llbracket P \rrbracket^{bin}$.

Theorem

Suppose $T_1, \ldots, T_k$ are well-founded binary relations on $S$, and $R$ another relation on $S$ where

$$R^+ \subseteq T_1 \cup \cdots \cup T_k.$$ 

Then $R$ is well-founded.

We prove that $\llbracket P \rrbracket^{step}$ is well-founded by analysing $\llbracket P \rrbracket^{bin}$. 
Assume that a state is \( \langle \ell, \sigma \rangle \) with \( \ell \in \Lambda \) (locations or labels).

Choose \( A = \mathcal{P}(\Lambda \times \Lambda) \).

Define \( \alpha(\{(\langle \ell, \sigma \rangle, \langle \ell', \sigma' \rangle)\}) = \{(\ell, \ell')\} \).

- What is \( \alpha([P]_{\text{step}}) \)?

- What is \( \alpha([P]_{\text{bin}}) \)?

- When does \( \alpha([P]_{\text{bin}}) \) imply termination?
Abstraction for Termination: A Simple Example

Assume that a state is $\langle \ell, \sigma \rangle$ with $\ell \in \Lambda$ (locations or labels).

Choose $A = \mathcal{P}(\Lambda \times \Lambda)$.

Define $\alpha(\{\langle \ell, \sigma \rangle, \langle \ell', \sigma' \rangle\}) = \{\langle \ell, \ell' \rangle\}$.

- What is $\alpha([P]^{step})$?
- What is $\alpha([P]^{bin})$?
Abstraction for Termination: A Simple Example

Assume that a state is $\langle \ell, \sigma \rangle$ with $\ell \in \Lambda$ (locations or labels).

Choose $A = \mathcal{P}(\Lambda \times \Lambda)$.

Define $\alpha(\{ (\langle \ell, \sigma \rangle, \langle \ell', \sigma' \rangle) \}) = \{ (\ell, \ell') \}$.

- What is $\alpha(\llbracket P \rrbracket^{step})$?
- What is $\alpha(\llbracket P \rrbracket^{bin})$?
- When does $\alpha(\llbracket P \rrbracket^{bin})$ imply termination?
Assume that a state is \( \langle \ell, \sigma \rangle \) with \( \ell \in \Lambda \) (locations or labels).

Choose \( A = \mathcal{P}(\Lambda \times \Lambda) \).

Define \( \alpha(\{(\langle \ell, \sigma \rangle, \langle \ell', \sigma' \rangle)\}) = \{ (\ell, \ell') \} \).

- What is \( \alpha([P]^{\text{step}}) \)?
- What is \( \alpha([P]^{\text{bin}}) \)?
- When does \( \alpha([P]^{\text{bin}}) \) imply termination?

### Computing \( \alpha([P]^{\text{bin}}) \)

1. Define operations on \( A = \mathcal{P}(\Lambda \times \Lambda) \): union \( \cup \), composition \( ; \)
2. Induced abstract operator \( F^A \):

\[
F^A(Y) = Y \cup \alpha([P]^{\text{step}}) \cup (Y; \alpha([P]^{\text{step}}))
\]

3. 
\[\alpha([P]^{\text{bin}}) = \text{lfp}F^A = \lim_{n \to \infty} (F^A)^n(\emptyset)\]
Abstraction for Termination: Size-Change

Assume that a state is $\langle \ell, \sigma \rangle$ with $\ell \in \Lambda$ and $\sigma$ assigns a value (in well-founded $Val$) to variables $x_1, \ldots, x_n$.

Let $MC_n$ be the set of monotonicity constraints over $n$ variables (technically $G \in MC_n$ is a triple $(f, g, \phi)$, $\phi =$ set of relations).

Choose $A = \mathcal{P}(MC_n)$. Operations: union (usual) and MC composition ; lifted to sets.

The Galois connection is defined by $\gamma(\{G\}) = \Phi(G)$, the relation (over $S$) represented by $G$.

**Lemma (composition is sound)**

$\Phi(G; H) \supseteq \Phi(G); \Phi(H)$.

Abstraction of a program $P$ is a set $G \subseteq MC_n$ where

$\gamma(G) \supseteq \llbracket P \rrbracket^{step}$
Comment: the order on $P(MC_n)$ is a bit tricky: it is not $\subseteq$. Instead,

$$S_1 \preceq S_2 \equiv \text{def } \gamma(S_1) \subseteq \gamma(S_2).$$

Now, we abstract $\llbracket P \rrbracket^{bin}$ using the abstract operator $F^{MC}$:

$$F^{MC}(S) = S \cup \mathcal{G} \cup (S; \mathcal{G})$$

We get $\llbracket P \rrbracket^{MC-bin} = \text{lfp}F^{MC} = \lim_{n \to \infty} (F^{MC})^n(\emptyset)$.

This is the closure $\text{cl}(\mathcal{G})$. The Theorem of LFP Abstraction tells us that

$$\llbracket P \rrbracket^{bin} \subseteq \gamma(\llbracket P \rrbracket^{MC-bin}) = \bigcup_{G \in \text{cl}(\mathcal{G})} \Phi(G).$$

(Note that again, we abstract $\llbracket P \rrbracket^{bin}$ by a finite object.)
Testing for termination

Let \( G = (f, f, \phi) \). Sagiv's Test tells whether \( \phi \land (\bigwedge_{i=1}^{n} x_i > x'_i) \) is unsatisfiable; equivalently, \( \phi \rightarrow \bigvee_{i=1}^{n} x_i > x'_i \). That is,

\[
\Phi(G) \subseteq \bigcup_{i=1}^{n} D_i \quad \text{where} \quad D_i = \{(\ell, \bar{x}), (\ell, \bar{x'}) \mid x_i > x'_i \}
\]

For \( G = (f, g, \phi) \) with \( f \neq g \), \( \Phi(G) \) is trivially contained in \( T_{f,g} = \Phi((f, g, \text{true})) \) which is also well-founded.

**Conclusion:** if every \( G \in \text{cl}(\mathcal{G}) \) which is cyclic satisfies Sagiv's test, we have

\[
\llbracket P \rrbracket^{bin} \subseteq \bigcup_{G \in \text{cl}(\mathcal{G})} \bigcup_{i=1}^{n} D_i \cup \bigcup_{f \neq g} T_{f,g}
\]

so \( P \) terminates by the local principle.
Abstract Semantics and the Closure Algorithm

Amir M. Ben-Amram

MTA

May 2010
Language definition (abstract syntax):

\[
\begin{align*}
X, Y & \in \text{Variable} \quad ::= \quad X_1 | X_2 | X_3 | \ldots | X_n \\
E & \in \text{Expression} \quad ::= \quad X | X - 1 | \ldots \\
C & \in \text{Command} \quad ::= \quad \text{skip} | X := E \\
& \quad | \quad C_1; C_2 | \text{loop} \ C \\
& \quad | \quad \text{branch} \ C_1 \lor C_2
\end{align*}
\]
$S$ is the set of mappings from $\{X_1, \ldots, X_n\}$ to $\text{Val}$

Assume given: function $E[E] : S \rightarrow \text{Val}$ to evaluate expressions

Goal: define command semantics $C[C] \in \mathcal{P}(S \times S)$ where $c \in \text{Command}$.

The definition follows the program structure (bottom up).

\[
\begin{align*}
C[\text{skip}] &= \text{id}_S \\
C[X_i := E] &= \{(\sigma, \sigma[X_i \mapsto E[E]\sigma])\}
\end{align*}
\]

\[
\begin{align*}
C[C_1; C_2] &= C[C_1] \cup C[C_2] \\
C[\text{branch } C_1 \lor C_2] &= C[C_1] \cup C[C_2] \\
C[\text{loop } C] &= \text{id}_S \cup \text{lfp}(F(C[C]))
\end{align*}
\]

where $F(S)(X) = S \cup X \cup (X; S)$. 

Abstraction of Structured Programs

Note the **semantic operators**: \( id_S \), substitution, composition \( ; \), union, lfp.

For abstraction, let \( A = \mathcal{P}(MC_n) \) (here without flow-points—just graphs).

\( A \) has the operations \( id_A \) (what is it?), composition \( (;) \), union, lfp.

So, we can define the **abstraction function** \( \text{Abstract} : \text{Command} \rightarrow A \).

\[
\text{Abstract}(\text{skip}) = id_A \\
\text{Abstract}(X_i := E) = \ldots \\
\text{Abstract}(C_1;C_2) = \text{Abstract}(C_1); \text{Abstract}(C_2) \\
\text{Abstract}(\text{branch} \ C_1 \lor C_2) = \text{Abstract}(C_1) \cup \text{Abstract}(C_2) \\
\text{Abstract}(\text{loop} \ C) = id_A \cup \text{lfp}(F^A)(\text{Abstract}(C))
\]

where \( F^A(S)(X) = S \cup X \cup (X; S) \).

**The Termination test**: each time you compute an lfp, verify that all graphs in the set pass the LTT.
Proofs of correctness

1. Soundness of the abstraction: induction on the command structure.
   The claim is: \( \forall C \in \text{Command}, \)
   \[ C[[C]] \subseteq \gamma(\text{ABSTRACT}(C)). \]

2. Soundness of the termination proof:
   - Only loops matter.
   - Consider a loop \( \text{loop } C. \)
     The part \( \text{lfp}(F(C[[C]])) \) describes the effect of executing the loop body 1 or more times.
     If the loop body described by a relation \( R, \text{lfp}(F(C[[C]])) \) gives \( R^+ \).
     By the local principle, it suffices to cover \( R^+ \) by a finite number of well-founded relations.
     From (1) we have \( \text{lfp}(F(C[[C]])) \subseteq \gamma(Y) \) where
     \[ Y = \text{lfp}(F^A)(\text{ABSTRACT}(C)). \]
     Also, \( \gamma(Y) = \bigcup_{G \in Y} \Phi(G). \)
     But each \( \Phi(G) \) is well-founded because we tested it locally.
     So the loop relation is well-founded: the loop always terminates.