Fast Offline Partial Evaluation of Large Logic Programs

Germán Vidal, T. U. Valencia, Spain

Joint work with Michael Leuschel, U. Düsseldorf, Germany

Int'l Symp. on Logic-Based Program Synthesis and Transformation
LOPSTR 2008

July 17-18, 2008
Valencia, Spain

Introduction

context: offline partial evaluation of logic programs

- PE = BTA + specialization
- BTA = termination analysis + propagation of BTs + annotation

Annotations

- calls are annotated as unfold/memo
- arguments are annotated as static/dynamic

Introduction

context: offline partial evaluation of logic programs

- PE = BTA + specialization
- BTA = termination analysis + propagation of BTs + annotation

Annotations

- calls are annotated as unfold/memo
- arguments are annotated as static/dynamic

specialization

1. take an atom and unfold it as much as possible (following the unfold/memo annotations) — local control
2. for every atom in the leaves — global control
   - generalize arguments marked as dynamic
   - add it to the set of atoms to be partially evaluated
3. go to step (1)

Termination issues in PE are classified into

- **local termination:** no atom is infinitely unfolded
- **global termination:** no infinitely many atoms are unfolded

Annotations are **safe** when:

- unfold/memo annotations guarantee local termination
- static/dynamic annotations guarantee global termination
  [all arguments marked as static are ground at specialization time]

specialization

1. take an atom and unfold it as much as possible (following the unfold/memo annotations) — local control
2. for every atom in the leaves — global control
   - generalize arguments marked as dynamic
   - add it to the set of atoms to be partially evaluated
3. go to step (1)

Termination issues in PE are classified into

- **local termination:** no atom is infinitely unfolded
- **global termination:** no infinitely many atoms are unfolded

Annotations are **safe** when:

- unfold/memo annotations guarantee local termination
- static/dynamic annotations guarantee global termination

  [all arguments marked as static are ground at specialization time]

specialization

1. take an atom and unfold it as much as possible (following the unfold/memo annotations) — local control
2. for every atom in the leaves — global control
   - generalize arguments marked as dynamic
   - add it to the set of atoms to be partially evaluated
3. go to step (1)

Termination issues in PE are classified into

- **local termination:** no atom is infinitely unfolded
- **global termination:** no infinitely many atoms are unfolded

Annotations are **safe** when:

- unfold/memo annotations guarantee local termination
- static/dynamic annotations guarantee global termination
  [all arguments marked as static are ground at specialization time]

Motivation

Fully automatic BTA for logic programs
[Craig, Gallagher, Leuschel, Henriksen, LOPSTR 2004]

- current implementation does not guarantee global termination
- complex design (different analyses running on different Prolog systems) $\rightarrow$ difficult to maintain
- slow, does not scale to medium-sized examples

In this paper, we propose a new BTA:

- simpler and faster, scales better to larger examples
- ensures both local and global termination

Motivation

Fully automatic BTA for logic programs
[Craig, Gallagher, Leuschel, Henriksen, LOPSTR 2004]

- current implementation does not guarantee global termination
- complex design (different analyses running on different Prolog systems) → difficult to maintain
- slow, does not scale to medium-sized examples

In this paper, we propose a new BTA:

- simpler and faster, scales better to larger examples
- ensures both local and global termination

Termination analysis

Essential component of BTA

Choice:

dependent of a computation rule

$$H \leftarrow B_1, B_2, \ldots, B_n \qquad H \leftarrow B_1, B_2, \ldots, B_n$$

- more accurate
- slower (requires reexecution every time an annotation changes)

independent of a computation rule

$$H \leftarrow B_1, B_2, \ldots, B_n$$

- less accurate, faster (no reexecution is needed)

Termination analysis

Essential component of BTA

Choice:

dependent of a computation rule

$$H \leftarrow B_1, B_2, \ldots, B_n \qquad H \leftarrow B_1, B_2, \ldots, B_n$$

- more accurate
- slower (requires reexecution every time an annotation changes)

independent of a computation rule

$$H \leftarrow B_1, B_2, \ldots, B_n$$

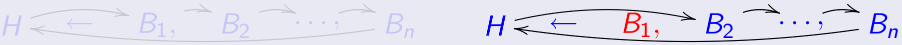
- less accurate, faster (no reexecution is needed)

Termination analysis

Essential component of BTA

Choice:

dependent of a computation rule



- more accurate
- slower (requires reexecution every time an annotation changes)

independent of a computation rule

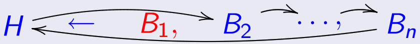
- less accurate, faster (no reexecution is needed)

Termination analysis

Essential component of BTA

Choice:

dependent of a computation rule



- more accurate
- slower (requires reexecution every time an annotation changes)

independent of a computation rule

- less accurate, faster (no reexecution is needed)

Termination analysis

Essential component of BTA

Choice:

dependent of a computation rule

$H \leftarrow B_1, B_2, \ldots, B_n$ $\qquad$ $H \leftarrow B_1, B_2, \ldots, B_n$

- more accurate
- slower (requires reexecution every time an annotation changes)

independent of a computation rule

$$H \leftarrow B_1, B_2, \ldots, B_n$$

- less accurate, faster (no reexecution is needed)

Quasi-termination analysis for logic programs [Vidal, PEPM 2007]

- considers **strong** termination (independent of the computation rule)
- based on size-change analysis [Lee, Jones, Ben-Amram, POPL 2001]
- covers termination (unfold/memo) and quasi-termination (static/dynamic)

size-change analysis

1. construction of size-change graphs
2. computation of transitive closure (by concatenation)
3. identification of program loops (maximal graphs)

Quasi-termination analysis for logic programs [Vidal, PEPM 2007]

- considers **strong** termination (independent of the computation rule)
- based on size-change analysis [Lee, Jones, Ben-Amram, POPL 2001]
- covers termination (unfold/memo) and quasi-termination (static/dynamic)

size-change analysis

1. construction of size-change graphs
2. computation of transitive closure (by concatenation)
3. identification of program loops (maximal graphs)

Construction of size-change graphs

Size-change graphs are used to trace size changes of predicate arguments from one call to another

Parameterized by a **reduction pair** $(\succsim, \succ)$

1. $\succsim$ is a quasi-order [reflexive & transitive]
2. $\succ$ is a well-founded order [irreflexive & transitive]
3. $\succsim$ and $\succ$ are closed under substitutions $[s \succsim t \;\Rightarrow\; \sigma(s) \succsim \sigma(t)]$
4. they are compatible

   (i.e., $\succsim \circ \succ \subseteq \succ$ and $\succ \circ \succsim \subseteq \succ$ but $\succsim \subseteq \succ$ is not necessary)

which can be induced from a **symbolic norm**

$$s \succ t \;\Leftrightarrow\; ||s|| > ||t|| \quad \text{and} \quad s \succsim t \;\Leftrightarrow\; ||s|| \geqslant ||t||$$

Construction of size-change graphs

Size-change graphs are used to trace size changes of predicate arguments from one call to another

Parameterized by a **reduction pair** $(\succsim, \succ)$

1. $\succsim$ is a quasi-order [reflexive & transitive]
2. $\succ$ is a well-founded order [irreflexive & transitive]
3. $\succsim$ and $\succ$ are closed under substitutions [$s \succsim t \;\Rightarrow\; \sigma(s) \succsim \sigma(t)$]
4. they are compatible
   (i.e., $\succsim \circ \succ \subseteq \succ$ and $\succ \circ \succsim \subseteq \succ$ but $\succsim \subseteq \succ$ is not necessary)

which can be induced from a **symbolic norm**

$$s \succ t \quad \Leftrightarrow \quad ||s|| > ||t|| \qquad \text{and} \qquad s \succsim t \quad \Leftrightarrow \quad ||s|| \geqslant ||t||$$

Symbolic norms

symbolic term-size norm

$$||t||_{ts} = \begin{cases} n + \sum_{i=0}^{n} ||t_i||_{ts} & \text{if } t = f(t_1, \ldots, t_n),\ n \geqslant 0 \\ t & \text{if } t \text{ is a variable} \end{cases}$$

E.g., $||\mathbf{f}(\mathbf{a}, \mathbf{b})||_{ts} = \mathbf{2}$, but $||\mathbf{f}(\mathbf{X}, \mathbf{Y})||_{ts} = \mathbf{2} + \mathbf{X} + \mathbf{Y}$

symbolic list-length norm

$$||t||_{ll} = \begin{cases} 1 + ||Xs||_{ll} & \text{if } t = [X|Xs] \\ t & \text{if } t \text{ is a variable} \\ 0 & \text{otherwise} \end{cases}$$

E.g., $||[\mathbf{1}, \mathbf{X}]||_{ll} = \mathbf{2}$, but $||[\mathbf{1}|\mathbf{X}]||_{ll} = \mathbf{1} + \mathbf{X}$

Symbolic norms

symbolic term-size norm

$$||t||_{ts} = \begin{cases} n + \sum_{i=0}^{n} ||t_i||_{ts} & \text{if } t = f(t_1, \ldots, t_n),\ n \geqslant 0 \\ t & \text{if } t \text{ is a variable} \end{cases}$$

E.g., $||\mathbf{f(a,b)}||_{ts} = \mathbf{2}$, but $||\mathbf{f(X,Y)}||_{ts} = \mathbf{2} + \mathbf{X} + \mathbf{Y}$

symbolic list-length norm

$$||t||_{ll} = \begin{cases} 1 + ||Xs||_{ll} & \text{if } t = [X|Xs] \\ t & \text{if } t \text{ is a variable} \\ 0 & \text{otherwise} \end{cases}$$

E.g., $||\mathbf{[1,X]}||_{ll} = \mathbf{2}$, but $||\mathbf{[1|X]}||_{ll} = \mathbf{1} + \mathbf{X}$

Example: construction of size-change graphs

$$
\begin{array}{lll}
\mathbf{incList}([\,], \_, [\,]). & & \\
\mathbf{incList}([\mathbf{X}|\mathbf{R}], \mathbf{I}, \mathbf{L}) & \leftarrow & \mathbf{iList}(\mathbf{X}, \mathbf{R}, \mathbf{I}, \mathbf{L}). \\
\mathbf{iList}(\mathbf{X}, \mathbf{R}, \mathbf{I}, [\mathbf{XI}|\mathbf{RI}]) & \leftarrow & \mathbf{add}(\mathbf{I}, \mathbf{X}, \mathbf{XI}), \\
 & & \mathbf{incList}(\mathbf{R}, \mathbf{I}, \mathbf{RI}).
\end{array}
$$

$$
\begin{array}{lll}
\mathbf{add}(\mathbf{0}, \mathbf{Y}, \mathbf{Y}). & & \\
\mathbf{add}(\mathbf{s}(\mathbf{X}), \mathbf{Y}, \mathbf{s}(\mathbf{Z})) & \leftarrow & \mathbf{add}(\mathbf{X}, \mathbf{Y}, \mathbf{Z}).
\end{array}
$$

Example: construction of size-change graphs

incList([ ], _, [ ]).
incList([X|R], I, L) ← iList(X, R, I, L).
iList(X, R, I, [XI|RI]) ← add(I, X, XI),
incList(R, I, RI).

add(0, Y, Y).
add(s(X), Y, s(Z)) ← add(X, Y, Z).

Example: construction of size-change graphs

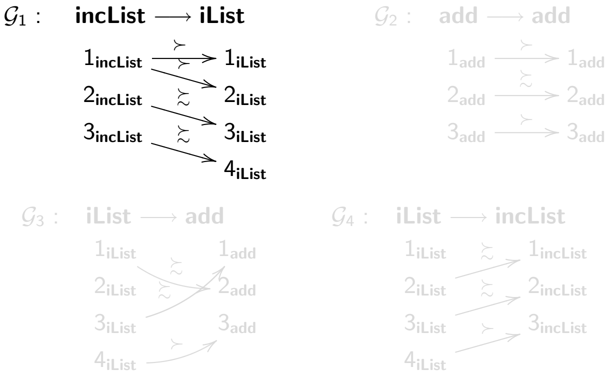
$\mathsf{incList}([\,], \_, [\,]).$

$\mathbf{incList}([\mathbf{X}|\mathbf{R}], \mathbf{I}, \mathbf{L}) \leftarrow \mathbf{iList}(\mathbf{X}, \mathbf{R}, \mathbf{I}, \mathbf{L}).$

$\mathsf{iList}(\mathsf{X}, \mathsf{R}, \mathsf{I}, [\mathsf{XI}|\mathsf{RI}]) \leftarrow \mathsf{add}(\mathsf{I}, \mathsf{X}, \mathsf{XI}), \mathsf{incList}(\mathsf{R}, \mathsf{I}, \mathsf{RI}).$

$\mathsf{add}(0, \mathsf{Y}, \mathsf{Y}).$

$\mathsf{add}(\mathsf{s}(\mathsf{X}), \mathsf{Y}, \mathsf{s}(\mathsf{Z})) \leftarrow \mathsf{add}(\mathsf{X}, \mathsf{Y}, \mathsf{Z}).$



Example: construction of size-change graphs

incList([ ], _, [ ]).
incList([X|R], I, L) ← iList(X, R, I, L).

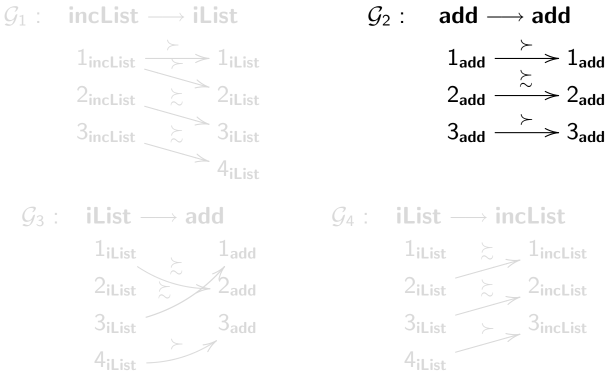
iList(X, R, I, [XI|RI]) ← add(I, X, XI),
    incList(R, I, RI).

add(0, Y, Y).
add(s(X), Y, s(Z)) ← add(X, Y, Z).

Example: construction of size-change graphs

incList([ ], _, [ ]).
incList([X|R], I, L) ← iList(X, R, I, L).
iList(X, R, I, [XI|RI]) ← add(I, X, XI),
incList(R, I, RI).

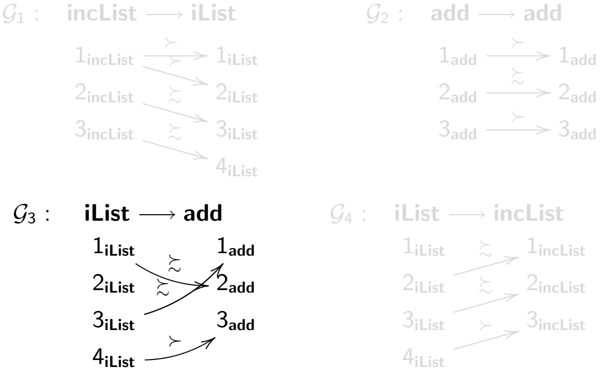
add(0, Y, Y).
add(s(X), Y, s(Z)) ← add(X, Y, Z).



Example: construction of size-change graphs

incList([ ], _, [ ]).
incList([X|R], I, L) ← iList(X, R, I, L).
iList(X, R, I, [XI|RI]) ← add(I, X, XI),
incList(R, I, RI).

add(0, Y, Y).
add(s(X), Y, s(Z)) ← add(X, Y, Z).



Example: construction of size-change graphs

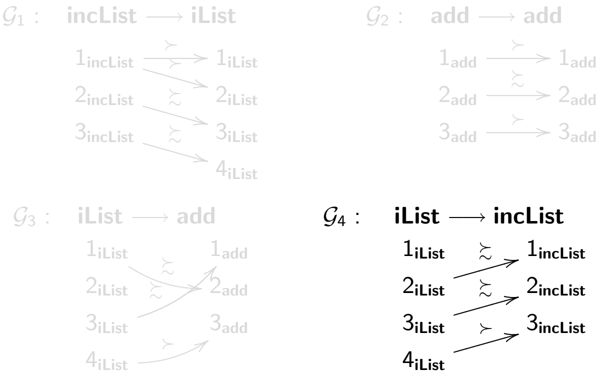
$\mathsf{incList}([\,], \_, [\,]).$

$\mathsf{incList}([X|R], I, L) \leftarrow \mathsf{iList}(X, R, I, L).$

$\mathsf{add}(0, Y, Y).$

$\mathsf{add}(s(X), Y, s(Z)) \leftarrow \mathsf{add}(X, Y, Z).$

$\mathbf{iList}(\mathbf{X}, \mathbf{R}, \mathbf{I}, [\mathbf{XI}|\mathbf{RI}]) \leftarrow \mathsf{add}(I, X, XI),$
$\qquad \mathbf{incList}(\mathbf{R}, \mathbf{I}, \mathbf{RI}).$

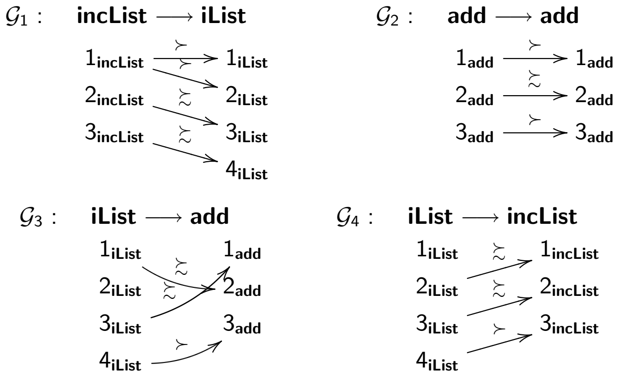


Example: construction of size-change graphs

incList([ ], _, [ ]).
incList([**X**|**R**], **I**, **L**) ← **iList**(**X**, **R**, **I**, **L**).

iList(**X**, **R**, **I**, [**XI**|**RI**]) ← **add**(**I**, **X**, **XI**),
incList(**R**, **I**, **RI**).

add(**0**, **Y**, **Y**).
add(**s**(**X**), **Y**, **s**(**Z**)) ← **add**(**X**, **Y**, **Z**).

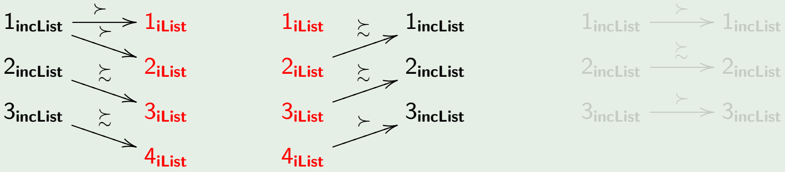


Transitive closure:

- compute all possible concatenations of graphs

Example

$\mathcal{G}_1$:**incList** $\longrightarrow$ **iList** $\bullet$ $\mathcal{G}_4$:**iList** $\longrightarrow$ **incList** $=$ $\mathcal{G}_{14}$:**incList** $\longrightarrow$ **incList**

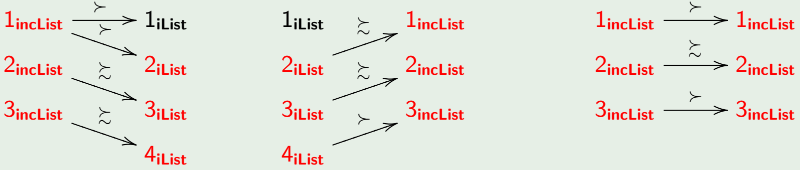


Transitive closure:

- compute all possible concatenations of graphs

Example

$$\mathcal{G}_1 : \textbf{incList} \longrightarrow \textbf{iList} \quad \bullet \quad \mathcal{G}_4 : \textbf{iList} \longrightarrow \textbf{incList} \quad = \quad \mathcal{G}_{14} : \textbf{incList} \longrightarrow \textbf{incList}$$



Identification of program loops

maximal graph

A size-change graph G is maximal if

1. its input and output nodes are the same
2. it is idempotent, i.e., $G = G \cdot G$

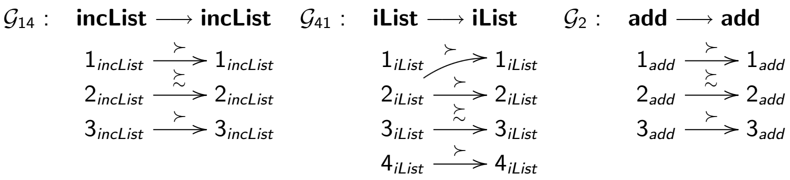
maximal graph $\approx$ program loop

Example

$\mathbf{incList}([\,], \_, [\,]).$
$\mathbf{incList}([\mathbf{X}|\mathbf{R}], \mathbf{I}, \mathbf{L}) \leftarrow \mathbf{iList}(\mathbf{X}, \mathbf{R}, \mathbf{I}, \mathbf{L}).$

$\mathbf{iList}(\mathbf{X}, \mathbf{R}, \mathbf{I}, [\mathbf{XI}|\mathbf{RI}]) \leftarrow \mathbf{add}(\mathbf{I}, \mathbf{X}, \mathbf{XI}),$
$\quad \mathbf{incList}(\mathbf{R}, \mathbf{I}, \mathbf{RI}).$

$\mathbf{add}(\mathbf{0}, \mathbf{Y}, \mathbf{Y}).$
$\mathbf{add}(\mathbf{s}(\mathbf{X}), \mathbf{Y}, \mathbf{s}(\mathbf{Z})) \leftarrow \mathbf{add}(\mathbf{X}, \mathbf{Y}, \mathbf{Z}).$



Size-change terminating! (acc. to [Lee, Jones, Ben-Amram, POPL 2001])

Too weak in logic programming...

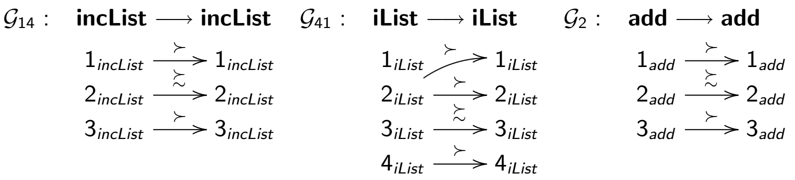
$$\mathbf{add}(\mathbf{X},\mathbf{Y},\mathbf{Z}) \Rightarrow_{\{X \mapsto s(X')],\ Z \mapsto s(Z')\}} \mathbf{add}(\mathbf{X}',\mathbf{Y},\mathbf{Z}') \Rightarrow_{\{X' \mapsto s(X''),\ Z' \mapsto s(Z'')\}} \mathbf{add}(\mathbf{X}'',\mathbf{Y},\mathbf{Z}'') \Rightarrow \infty$$

Example

$\textbf{incList}([\,], \_, [\,]).$
$\textbf{incList}([X|R], I, L) \leftarrow \textbf{iList}(X, R, I, L).$

$\textbf{iList}(X, R, I, [XI|RI]) \leftarrow \textbf{add}(I, X, XI), \textbf{incList}(R, I, RI).$

$\textbf{add}(0, Y, Y).$
$\textbf{add}(s(X), Y, s(Z)) \leftarrow \textbf{add}(X, Y, Z).$



Size-change terminating! (acc. to [Lee, Jones, Ben-Amram, POPL 2001])

Too weak in logic programming...

$$\textbf{add}(X, Y, Z) \Rightarrow_{\{X \mapsto s(X')], \; Z \mapsto s(Z')\}} \textbf{add}(X', Y, Z') \Rightarrow_{\{X' \mapsto s(X''), \; Z' \mapsto s(Z'')\}} \textbf{add}(X'', Y, Z'') \Rightarrow \infty$$

Example

$$\begin{array}{ll} \textbf{incList}([\,], \_, [\,]). & \textbf{add}(\mathbf{0}, \mathbf{Y}, \mathbf{Y}). \\ \textbf{incList}([\mathbf{X}|\mathbf{R}], \mathbf{I}, \mathbf{L}) \leftarrow \textbf{iList}(\mathbf{X}, \mathbf{R}, \mathbf{I}, \mathbf{L}). & \textbf{add}(\mathbf{s}(\mathbf{X}), \mathbf{Y}, \mathbf{s}(\mathbf{Z})) \leftarrow \textbf{add}(\mathbf{X}, \mathbf{Y}, \mathbf{Z}). \end{array}$$

$$\textbf{iList}(\mathbf{X}, \mathbf{R}, \mathbf{I}, [\mathbf{XI}|\mathbf{RI}]) \leftarrow \textbf{add}(\mathbf{I}, \mathbf{X}, \mathbf{XI}),\ \textbf{incList}(\mathbf{R}, \mathbf{I}, \mathbf{RI}).$$

$\mathcal{G}_{14}$: **incList** $\longrightarrow$ **incList**

$$1_{incList} \xrightarrow{\succ} 1_{incList} \qquad 2_{incList} \xrightarrow{\succsim} 2_{incList} \qquad 3_{incList} \xrightarrow{\succ} 3_{incList}$$

$\mathcal{G}_{41}$: **iList** $\longrightarrow$ **iList**

$$2_{iList} \xrightarrow{\succ} 1_{iList} \qquad 2_{iList} \xrightarrow{\succ} 2_{iList} \qquad 3_{iList} \xrightarrow{\succsim} 3_{iList} \qquad 4_{iList} \xrightarrow{\succ} 4_{iList}$$

$\mathcal{G}_{2}$: **add** $\longrightarrow$ **add**

$$1_{add} \xrightarrow{\succ} 1_{add} \qquad 2_{add} \xrightarrow{\succsim} 2_{add} \qquad 3_{add} \xrightarrow{\succ} 3_{add}$$

Size-change terminating! (acc. to [Lee, Jones, Ben-Amram, POPL 2001])

Too weak in logic programming...

$$\begin{array}{lll} \textbf{add}(\mathbf{X}, \mathbf{Y}, \mathbf{Z}) & \Rightarrow_{\{\mathbf{X} \mapsto \mathbf{s}(\mathbf{X}')],\ \mathbf{Z} \mapsto \mathbf{s}(\mathbf{Z}')\}} & \textbf{add}(\mathbf{X}', \mathbf{Y}, \mathbf{Z}') \\ & \Rightarrow_{\{\mathbf{X}' \mapsto \mathbf{s}(\mathbf{X}''),\ \mathbf{Z}' \mapsto \mathbf{s}(\mathbf{Z}'')\}} & \textbf{add}(\mathbf{X}'', \mathbf{Y}, \mathbf{Z}'') \quad \Rightarrow \quad \infty \end{array}$$

Local termination

instantiated enough [Lindenstrauss, Sagiv, ICLP 1997]

Term t is instantiated enough w.r.t. $||\cdot||$ if $||t||$ is an integer

sufficient condition for termination

If every maximal graph for P contains at least one edge

$$i_p \stackrel{\succ}{\longrightarrow} i_p$$

such that, in every possible call, $p(t_1, \ldots, t_n)$, the argument t_i is instantiated enough w.r.t. $||\cdot||$, then P is terminating

such that the i-th argument of p is classified as static, then P is terminating

and p is annotated with unfold (with memo otherwise)

Local termination

instantiated enough [Lindenstrauss, Sagiv, ICLP 1997]

Term t is instantiated enough w.r.t. $||\cdot||$ if $||t||$ is an integer

sufficient condition for termination

If every maximal graph for P contains at least one edge

$$i_p \stackrel{\succ}{\longrightarrow} i_p$$

such that, in every possible call, $p(t_1, \ldots, t_n)$, the argument t_i is instantiated enough w.r.t. $||\cdot||$, then P is terminating

such that the i-th argument of p is classified as **static**, then P is terminating

and p is annotated with **unfold** (with **memo** otherwise)

Local termination

instantiated enough [Lindenstrauss, Sagiv, ICLP 1997]

Term t is instantiated enough w.r.t. $||\cdot||$ if $||t||$ is an integer

sufficient condition for termination

If every maximal graph for P contains at least one edge

$$i_p \stackrel{\succ}{\longrightarrow} i_p$$

such that, in every possible call, $p(t_1, \ldots, t_n)$, the argument t_i is instantiated enough w.r.t. $||\cdot||$, then P is terminating

such that the i-th argument of p is classified as **static**, then P is terminating

and p is annotated with **unfold** (with **memo** otherwise)

Local termination

instantiated enough [Lindenstrauss, Sagiv, ICLP 1997]

Term t is instantiated enough w.r.t. $||\cdot||$ if $||t||$ is an integer

sufficient condition for termination

If every maximal graph for P contains at least one edge

$$i_p \xrightarrow{\succ} i_p$$

such that, in every possible call, $p(t_1, \ldots, t_n)$, the argument t_i is instantiated enough w.r.t. $||\cdot||$, then P is terminating

such that the i-th argument of p is classified as **static**, then P is terminating

and p is annotated with **unfold** (with **memo** otherwise)

Global termination

Must ensure quasi-termination (only finitely many different atoms)

In our previous approach, global termination is ensured when

- for every predicate *p* and for every maximal graph for *p*, either
  - the previous condition hold or
  - there is an edge to every argument
- and the considered symbolic norm is **bounded**

  i.e., the set $\{s \mid ||t|| \geqslant ||s||\}$ is finite for any term t

with 1_{add} or 2_{add} static

But the symbolic list-length norm is not bounded...

$||[a]|| = ||[s(a)]|| = ||[s(s(a))]|| = \ldots = 1$

Global termination

Must ensure quasi-termination (only finitely many different atoms)

In our previous approach, global termination is ensured when

- for every predicate p and for every maximal graph for p, either
  - the previous condition hold or
  - there is an edge to every argument
- and the considered symbolic norm is **bounded**

  i.e., the set $\{s \mid ||t|| \geqslant ||s||\}$ is finite for any term t

$$\begin{array}{lcl} 1_{add} & \xrightarrow{\succ} & 1_{add} \\ 2_{add} & \xrightarrow{\succsim} & 2_{add} \\ 3_{add} & \xrightarrow{\succ} & 3_{add} \end{array}$$

with 1_{add} or 2_{add} static

$$\begin{array}{lcl} 1_{iList} & \xrightarrow{\succ} & 1_{iList} \\ 2_{iList} & \xrightarrow{\succ} & 2_{iList} \\ 3_{iList} & \xrightarrow{\succsim} & 3_{iList} \\ 4_{iList} & \xrightarrow{\succ} & 4_{iList} \end{array}$$

But the symbolic list-length norm is not bounded. . .

$||[a]|| = ||[s(a)]|| = ||[s(s(a))]|| = \ldots = 1$

Global termination

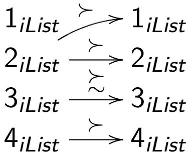
Must ensure quasi-termination (only finitely many different atoms)

In our previous approach, global termination is ensured when

- for every predicate p and for every maximal graph for p, either
  - the previous condition hold or
  - there is an edge to every argument
- and the considered symbolic norm is **bounded**

  i.e., the set $\{s \mid ||t|| \geqslant ||s||\}$ is finite for any term t

with 1_{add} or 2_{add} static



But the symbolic list-length norm is not bounded...

$||[a]|| = ||[s(a)]|| = ||[s(s(a))]|| = \ldots = 1$

Global termination

Must ensure quasi-termination (only finitely many different atoms)

In our previous approach, global termination is ensured when

- for every predicate p and for every maximal graph for p, either
  - the previous condition hold or
  - there is an edge to every argument
- and the considered symbolic norm is **bounded**

  i.e., the set $\{s \mid ||t|| \geqslant ||s||\}$ is finite for any term t

But the symbolic list-length norm is not bounded...

$||[a]|| = ||[s(a)]|| = ||[s(s(a))]|| = \ldots = 1$

Global termination

Must ensure quasi-termination (only finitely many different atoms)

In our previous approach, global termination is ensured when

- for every predicate p and for every maximal graph for p, either
  - the previous condition hold or
  - there is an edge to every argument
- and the considered symbolic norm is **bounded**

  i.e., the set $\{s \mid ||t|| \geqslant ||s||\}$ is finite for any term t

But the symbolic list-length norm is not bounded...

$|||[a]||| = |||[s(a)]||| = |||[s(s(a))]||| = \ldots = 1$

Is boundedness really needed?

Alternative approach

- use any symbolic norm (even if non-bounded)
- generalize problematic parts of atoms before adding them to the set of atoms to be partially evaluated (global level)

mgg: most general generalization of an atom w.r.t. a norm

$mgg^{||\cdot||}(t) = t'$

if t is the most general generalization of t such that $||t|| = ||mgg^{||\cdot||}(t)||$

$mgg^{||\cdot||_{ll}}([X]) = [X]$ $\quad$ $mgg^{||\cdot||_{ll}}([X,Y]) = [X,Y]$

$mgg^{||\cdot||_{ll}}([a]) = [X']$ $\quad$ $mgg^{||\cdot||_{ll}}([a,b]) = [X',Y']$

$mgg^{||\cdot||_{ll}}([s(a)]) = [X'']$ $\quad$ $mgg^{||\cdot||_{ll}}([s(a),s(b)]) = [X'',Y'']$

... $\quad$ ...

Is boundedness really needed?

Alternative approach

- use any symbolic norm (even if non-bounded)
- generalize problematic parts of atoms before adding them to the set of atoms to be partially evaluated (global level)

mgg: most general generalization of an atom w.r.t. a norm

$mgg^{||\cdot||}(t) = t'$

if t is the most general generalization of t such that $||t|| = ||mgg^{||\cdot||}(t)||$

$$mgg^{||\cdot||_{ll}}([X]) = [X] \qquad mgg^{||\cdot||_{ll}}([X,Y]) = [X,Y]$$
$$mgg^{||\cdot||_{ll}}([a]) = [X'] \qquad mgg^{||\cdot||_{ll}}([a,b]) = [X',Y']$$
$$mgg^{||\cdot||_{ll}}([s(a)]) = [X''] \qquad mgg^{||\cdot||_{ll}}([s(a),s(b)]) = [X'',Y'']$$
$$\ldots \qquad \ldots$$

Is boundedness really needed?

Alternative approach

- use any symbolic norm (even if non-bounded)
- generalize problematic parts of atoms before adding them to the set of atoms to be partially evaluated (global level)

mgg: most general generalization of an atom w.r.t. a norm

$mgg^{||\cdot||}(t) = t'$

if t is the most general generalization of t such that $||t|| = ||mgg^{||\cdot||}(t)||$

$mgg^{||\cdot||_{ll}}([X]) = [X]$ $\quad$ $mgg^{||\cdot||_{ll}}([X,Y]) = [X,Y]$

$mgg^{||\cdot||_{ll}}([a]) = [X']$ $\quad$ $mgg^{||\cdot||_{ll}}([a,b]) = [X',Y']$

$mgg^{||\cdot||_{ll}}([s(a)]) = [X'']$ $\quad$ $mgg^{||\cdot||_{ll}}([s(a),s(b)]) = [X'',Y'']$

$\ldots$ $\quad$ $\ldots$

Is boundedness really needed?

Alternative approach

- use any symbolic norm (even if non-bounded)
- generalize problematic parts of atoms before adding them to the set of atoms to be partially evaluated (global level)

mgg: most general generalization of an atom w.r.t. a norm

$mgg^{||\cdot||}(t) = t'$

if t is the most general generalization of t such that $||t|| = ||mgg^{||\cdot||}(t)||$

$$mgg^{||\cdot||_{ll}}([X]) = [X] \qquad mgg^{||\cdot||_{ll}}([X,Y]) = [X,Y]$$
$$mgg^{||\cdot||_{ll}}([a]) = [X'] \qquad mgg^{||\cdot||_{ll}}([a,b]) = [X',Y']$$
$$mgg^{||\cdot||_{ll}}([s(a)]) = [X''] \qquad mgg^{||\cdot||_{ll}}([s(a),s(b)]) = [X'',Y'']$$
$$\ldots \qquad \ldots$$

annotations for global termination

given a predicate *p* and an argument *i*:

1. if every maximal graph for *p* fulfills the local termination condition, i_p is marked as **static**
2. otherwise, if ∃ a maximal graph with no input edge to i_p, then it is marked as **dynamic**

Furthermore, arguments marked as **static** are changed to **list(dynamic)** if the list-length norm was used

annotations for global termination

given a predicate p and an argument i:

1. if every maximal graph for p fulfills the local termination condition, i_p is marked as **static**
2. otherwise, if $\exists$ a maximal graph with no input edge to i_p, then it is marked as **dynamic**

Furthermore, arguments marked as **static** are changed to **list(dynamic)** if the list-length norm was used

annotations for global termination

given a predicate p and an argument i:

1. if every maximal graph for p fulfills the local termination condition, i_p is marked as **static**
2. otherwise, if $\exists$ a maximal graph with no input edge to i_p, then it is marked as **dynamic**

Furthermore, arguments marked as **static** are changed to **list(dynamic)** if the list-length norm was used

Example

$$\begin{array}{ll}
\mathbf{incList}([\,], _, [\,]). & \mathbf{add}(\mathbf{0}, \mathbf{Y}, \mathbf{Y}). \\
\mathbf{incList}([\mathbf{X}|\mathbf{R}], \mathbf{I}, \mathbf{L}) \leftarrow \mathbf{iList}(\mathbf{X}, \mathbf{R}, \mathbf{I}, \mathbf{L}). & \mathbf{add}(\mathbf{s}(\mathbf{X}), \mathbf{Y}, \mathbf{s}(\mathbf{Z})) \leftarrow \mathbf{add}(\mathbf{X}, \mathbf{Y}, \mathbf{Z}).
\end{array}$$

$$\begin{array}{l}
\mathbf{iList}(\mathbf{X}, \mathbf{R}, \mathbf{I}, [\mathbf{XI}|\mathbf{RI}]) \leftarrow \mathbf{add}(\mathbf{I}, \mathbf{X}, \mathbf{XI}), \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathbf{incList}(\mathbf{R}, \mathbf{I}, \mathbf{RI}).
\end{array}$$

1. User's input: *incList(dynamic, static, dynamic)*
2. Propagation of BTs: *incList(D, S, D)*, *iList(D, D, S, D)*, *add(S, D, D)*
3. size-change analysis:
   - local termination: *incList* ↦ *memo*, *iList* ↦ *memo*, *add* ↦ *unfold*
   - global termination: no **static**→**dynamic** change required

Example

$$\textbf{incList}([\,], \_, [\,]).$$
$$\textbf{incList}([X|R], I, L) \leftarrow \textbf{iList}(X, R, I, L).$$
$$\textbf{iList}(X, R, I, [XI|RI]) \leftarrow \textbf{add}(I, X, XI),\ \textbf{incList}(R, I, RI).$$
$$\textbf{add}(0, Y, Y).$$
$$\textbf{add}(s(X), Y, s(Z)) \leftarrow \textbf{add}(X, Y, Z).$$

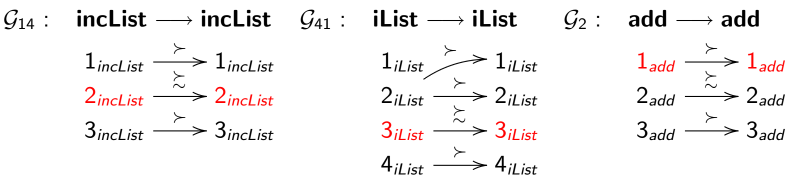
1. User's input: *incList(dynamic, static, dynamic)*
2. Propagation of BTs: *incList(D, S, D)*, *iList(D, D, S, D)*, *add(S, D, D)*
3. size-change analysis:
   - local termination: *incList* ↦ *memo*, *iList* ↦ *memo*, *add* ↦ *unfold*
   - global termination: no **static**→**dynamic** change required

Example

incList([ ], _, [ ]).
incList([**X**|**R**], **I**, **L**) ←**iList**(**X**, **R**, **I**, **L**).

iList(**X**, **R**, **I**, [**XI**|**RI**])←**add**(**I**, **X**, **XI**),
incList(**R**, **I**, **RI**).

add(**0**, **Y**, **Y**).
add(**s**(**X**), **Y**, **s**(**Z**))←**add**(**X**, **Y**, **Z**).



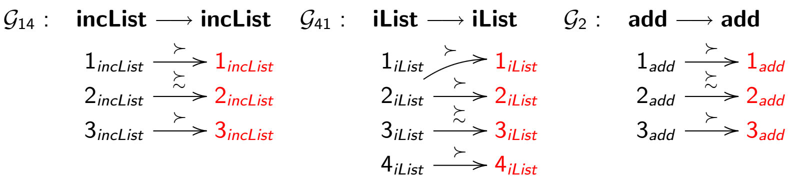
1. User's input: *incList*(*dynamic*, *static*, *dynamic*)
2. Propagation of BTs: *incList*(*D*, *S*, *D*), *iList*(*D*, *D*, *S*, *D*), *add*(*S*, *D*, *D*)
3. size-change analysis:
   - local termination: *incList* ↦ *memo*, *iList* ↦ *memo*, *add* ↦ *unfold*
   - global termination: no **static**→**dynamic** change required

Example

incList([ ], _, [ ]).
incList([**X**|**R**], **I**, **L**) ←**iList**(**X**, **R**, **I**, **L**).

iList(**X**, **R**, **I**, [**XI**|**RI**])←**add**(**I**, **X**, **XI**),
incList(**R**, **I**, **RI**).

add(**0**, **Y**, **Y**).
add(**s**(**X**), **Y**, **s**(**Z**))←**add**(**X**, **Y**, **Z**).



1. User's input: *incList(dynamic, static, dynamic)*
2. Propagation of BTs: *incList(D, S, D)*, *iList(D, D, S, D)*, *add(S, D, D)*
3. size-change analysis:
   - local termination: *incList* ↦ *memo*, *iList* ↦ *memo*, *add* ↦ *unfold*
   - global termination: no **static**→**dynamic** change required

Concluding remarks

Very fast BTA, scales well to medium-sized Prolog programs

Ensures both local and global termination

Less accurate than previous BTA. . .

Much room for improvement:

- improve accuracy of size-change analysis
- hybrid approach: replace **memo** and **dynamic** with **online** and use online techniques for them

Concluding remarks

Very fast BTA, scales well to medium-sized Prolog programs

Ensures both local and global termination

Less accurate than previous BTA. . .

Much room for improvement:

- improve accuracy of size-change analysis
- hybrid approach: replace **memo** and **dynamic** with **online** and use online techniques for them

M. Bezem.
Strong Termination of Logic Programs.
Journal of Logic Programming, 15(1&2):79–97, 1993.

S.-J. Craig, J. Gallagher, M. Leuschel, and K.S. Henriksen.
Fully Automatic Binding Time Analysis for Prolog.
In *Proc. of the Int'l Symposium on Logic-based Program Synthesis and Transformation (LOPSTR'04)*, pages 53–68. Springer LNCS 3573, 2005.

C.S. Lee, N.D. Jones, and A.M. Ben-Amram.
The Size-Change Principle for Program Termination.
SIGPLAN Notices (Proc. of POPL'01), 28:81–92, 2001.

N. Lindenstrauss and Y. Sagiv.
Automatic Termination Analysis of Logic Programs.
In *Proc. of Int'l Conf. on Logic Programming (ICLP'97)*, pages 63–77. The MIT Press, 1997.

G. Vidal.
Quasi-Terminating Logic Programs for Ensuring the Termination of Partial Evaluation.
In *Proc. of the ACM SIGPLAN 2007 Workshop on Partial Evaluation and Program Manipulation (PEPM'07)*, pages 51–60. ACM Press, 2007.