Proving the Termination of Narrowing by Proving the Termination of Rewriting

Germán Vidal
Technical University of Valencia, Spain

(Joint work with Naoki Nishida, University of Nagoya, Japan)

Research Project FAST, TIN2008-06622

Máster en Investigación
FDI, Universidad Complutense de Madrid

April 27, 2009 – Madrid, Spain

Outline

1. introduction
   - narrowing
2. termination of narrowing via termination of rewriting
   - data generators
   - main result
3. automating the termination analysis
   - abstract terms and argument filterings
   - a direct approach to termination analysis
   - a transformational approach
4. the technique in practice
   - the termination tool TNT
   - inference of safe argument filterings
   - some refinements
5. related work
6. conclusions

What is narrowing?

Standard definition of addition (TRS)

$$\begin{array}{rcll} \mathsf{add}(\mathsf{z}, y) & \to & y & (R_1) \\ \mathsf{add}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{add}(x, y)) & (R_2) \end{array}$$

With **rewriting**: $\mathsf{add}(\mathsf{s}(\mathsf{z}), \mathsf{z}) \to_{R_2} \mathsf{s}(\mathsf{add}(\mathsf{z}, \mathsf{z})) \to_{R_1} \mathsf{s}(\mathsf{z})$

With **narrowing**: $\mathsf{add}(\mathsf{s}(\mathsf{z}), \mathsf{z}) \leadsto_{R_2} \mathsf{s}(\mathsf{add}(\mathsf{z}, \mathsf{z})) \leadsto_{R_1} \mathsf{s}(\mathsf{z})$

but also:

$$\begin{tikzcd} \mathsf{add}(x, \mathsf{z}) \arrow[dr, "\text{``guess''}"'] & & \mathsf{s}(\mathsf{add}(y, \mathsf{z})) \arrow[dr, "\text{``guess''}"'] & & \mathsf{s}(\mathsf{z}) \\ & \mathsf{add}(\mathsf{s}(y), \mathsf{z}) \arrow[ur, "R_2"'] & & \mathsf{s}(\mathsf{add}(\mathsf{z}, \mathsf{z})) \arrow[ur, "R_1"'] & \end{tikzcd}$$

(many other non-deterministic reductions possible. . .)

What is narrowing?

Standard definition of addition (TRS)

$$
\begin{array}{rcll}
\mathsf{add}(\mathsf{z}, y) & \to & y & (R_1) \\
\mathsf{add}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{add}(x, y)) & (R_2)
\end{array}
$$

With **rewriting**: $\mathsf{add}(\mathsf{s}(\mathsf{z}), \mathsf{z}) \to_{R_2} \mathsf{s}(\mathsf{add}(\mathsf{z}, \mathsf{z})) \to_{R_1} \mathsf{s}(\mathsf{z})$

With **narrowing**: $\mathsf{add}(\mathsf{s}(\mathsf{z}), \mathsf{z}) \leadsto_{R_2} \mathsf{s}(\mathsf{add}(\mathsf{z}, \mathsf{z})) \leadsto_{R_1} \mathsf{s}(\mathsf{z})$

but also: $\mathsf{add}(x, \mathsf{z}) \xrightarrow{\text{“guess”}} \mathsf{add}(\mathsf{s}(y), \mathsf{z}) \xrightarrow{R_2} \mathsf{s}(\mathsf{add}(y, \mathsf{z})) \xrightarrow{\text{“guess”}} \mathsf{s}(\mathsf{add}(\mathsf{z}, \mathsf{z})) \xrightarrow{R_1} \mathsf{s}(\mathsf{z})$

(many other non-deterministic reductions possible. . .)

What is narrowing?

Standard definition of addition (TRS)

$$\begin{array}{rcll} \mathsf{add}(\mathsf{z}, y) & \to & y & (R_1) \\ \mathsf{add}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{add}(x, y)) & (R_2) \end{array}$$

With **rewriting**: $\mathsf{add}(\mathsf{s}(\mathsf{z}), \mathsf{z}) \to_{R_2} \mathsf{s}(\mathsf{add}(\mathsf{z}, \mathsf{z})) \to_{R_1} \mathsf{s}(\mathsf{z})$

With **narrowing**: $\mathsf{add}(\mathsf{s}(\mathsf{z}), \mathsf{z}) \leadsto_{R_2} \mathsf{s}(\mathsf{add}(\mathsf{z}, \mathsf{z})) \leadsto_{R_1} \mathsf{s}(\mathsf{z})$

but also:

$$\mathsf{add}(x, \mathsf{z}) \xrightarrow{\text{“guess”}} \mathsf{add}(\mathsf{s}(y), \mathsf{z}) \xrightarrow{R_2} \mathsf{s}(\mathsf{add}(y, \mathsf{z})) \xrightarrow{\text{“guess”}} \mathsf{s}(\mathsf{add}(\mathsf{z}, \mathsf{z})) \xrightarrow{R_1} \mathsf{s}(\mathsf{z})$$

(many other non-deterministic reductions possible. . .)

What is narrowing?

Standard definition of addition (TRS)

$$
\begin{array}{rcll}
\mathsf{add}(\mathsf{z}, y) & \to & y & (R_1) \\
\mathsf{add}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{add}(x, y)) & (R_2)
\end{array}
$$

With **rewriting**: $\mathsf{add}(\mathsf{s}(\mathsf{z}), \mathsf{z}) \to_{R_2} \mathsf{s}(\mathsf{add}(\mathsf{z}, \mathsf{z})) \to_{R_1} \mathsf{s}(\mathsf{z})$

With **narrowing**: $\mathsf{add}(\mathsf{s}(\mathsf{z}), \mathsf{z}) \leadsto_{R_2} \mathsf{s}(\mathsf{add}(\mathsf{z}, \mathsf{z})) \leadsto_{R_1} \mathsf{s}(\mathsf{z})$

but also:

$$
\begin{tikzcd}
\mathsf{add}(x, \mathsf{z}) \arrow[rr, phantom, "\leadsto_{R_2,\{x \mapsto \mathsf{s}(y)\}}"] \arrow[dr, "\text{``guess''}"'] & & \mathsf{s}(\mathsf{add}(y, \mathsf{z})) \arrow[rr, phantom, "\leadsto_{\{R_1, y \mapsto \mathsf{z}\}}"] \arrow[dr, "\text{``guess''}"'] & & \mathsf{s}(\mathsf{z}) \\
& \mathsf{add}(\mathsf{s}(y), \mathsf{z}) \arrow[ur, "R_2"'] & & \mathsf{s}(\mathsf{add}(\mathsf{z}, \mathsf{z})) \arrow[ur, "R_1"'] &
\end{tikzcd}
$$

(many other non-deterministic reductions possible. . .)

Formal definition

Definition (rewriting)

$\boxed{s \to_{p,R} s[r\sigma]_p}$ if there are $\begin{cases} \bullet \text{ a position } p \text{ of } s \\ \bullet \text{ a rule } R = (l \to r) \text{ in } \mathcal{R} \\ \bullet \text{ a substitution } \sigma \text{ such that } s|_p = l\sigma \end{cases}$

$\Downarrow$

Definition (narrowing)

$\boxed{s \leadsto_{p,R,\sigma} (s[r]_p)\sigma}$ if there are $\begin{cases} \bullet \text{ a nonvariable position } p \text{ of } s \\ \bullet \text{ a variant } R = (l \to r) \text{ of a rule in } \mathcal{R} \\ \bullet \text{ a substitution } \sigma \text{ such that } s|_p\sigma = l\sigma \\ \quad [\sigma = \mathrm{mgu}(s|_p, l)] \end{cases}$

Some motivation

We want to analyze the termination of narrowing

Why?

- narrowing is relevant in a number of areas: functional logic languages, partial evaluation, protocol verification, type inference, etc
- no termination prover for narrowing

We want to analyze the termination of narrowing by analyzing the termination of rewriting

Why?

- many techniques and tools for rewriting!

Main ideas

- replace logic variables by data generators
- analyze the termination of rewriting with data generators
- adapt direct and transformational approaches

Some motivation

We want to analyze the termination of narrowing

Why?

- narrowing is relevant in a number of areas: functional logic languages, partial evaluation, protocol verification, type inference, etc
- no termination prover for narrowing

We want to analyze the termination of narrowing by analyzing the termination of rewriting

Why?

- many techniques and tools for rewriting!

Main ideas

- replace logic variables by data generators
- analyze the termination of rewriting with data generators
- adapt direct and transformational approaches

Some motivation

We want to analyze the termination of narrowing

Why?

- narrowing is relevant in a number of areas: functional logic languages, partial evaluation, protocol verification, type inference, etc
- no termination prover for narrowing

We want to analyze the termination of narrowing by analyzing the termination of rewriting

Why?

- many techniques and tools for rewriting!

Main ideas

- replace logic variables by data generators
- analyze the termination of rewriting with data generators
- adapt direct and transformational approaches

termination of narrowing via termination of rewriting

Termination of narrowing

The termination problem

- given a TRS, are all possible narrowing derivations finite?

Too strong!

$$\mathsf{add}(x,y) \leadsto_{R_2,\{x \mapsto \mathsf{s}(x')\}} \mathsf{add}(x',y) \leadsto_{R_2,\{x' \mapsto \mathsf{s}(x'')\}} \cdots$$

In this work

- given a TRS $\mathcal{R}$ and a set of terms T,
  are all possible narrowing derivations $t_1 \leadsto t_2 \leadsto \ldots$ for $t_1 \in T$ finite?
  (in symbols: T is $\leadsto_{\mathcal{R}}$-terminating)

For instance, $\{\ \mathsf{add}(s,t) \mid s \text{ is ground}\ \}$ is $\leadsto_{\mathcal{R}}$-terminating

Termination of narrowing

The termination problem

- given a TRS, are all possible narrowing derivations finite?

Too strong!

$$\mathsf{add}(x,y) \leadsto_{R_2,\{x\mapsto \mathsf{s}(x')\}} \mathsf{add}(x',y) \leadsto_{R_2,\{x'\mapsto \mathsf{s}(x'')\}} \cdots$$

In this work

- given a TRS $\mathcal{R}$ and a set of terms T,
  are all possible narrowing derivations $t_1 \leadsto t_2 \leadsto \ldots$ for $t_1 \in T$ finite?
  (in symbols: T is $\leadsto_{\mathcal{R}}$-terminating)

For instance, $\{\ \mathsf{add}(s,t) \mid s \text{ is ground}\ \}$ is $\leadsto_{\mathcal{R}}$-terminating

Termination of narrowing

The termination problem

- given a TRS, are all possible narrowing derivations finite?

Too strong!

$$\mathsf{add}(x,y) \leadsto_{R_2,\{x \mapsto \mathsf{s}(x')\}} \mathsf{add}(x',y) \leadsto_{R_2,\{x' \mapsto \mathsf{s}(x'')\}} \cdots$$

$\boxed{\textbf{In this work}}$

- given a TRS $\mathcal{R}$ and a set of terms T,
  are all possible narrowing derivations $t_1 \leadsto t_2 \leadsto \ldots$ for $t_1 \in T$ finite?
  (in symbols: T is $\leadsto_{\mathcal{R}}$-terminating)

For instance, $\{\ \mathsf{add}(s,t) \mid s \text{ is ground}\ \}$ is $\leadsto_{\mathcal{R}}$-terminating

Termination of narrowing via termination of rewriting

Theorem

T is $\leadsto_{\mathcal{R}}$-terminating
if $\{t\sigma \mid t \in T \text{ and } t \leadsto^*_\sigma s \text{ in } \mathcal{R}\}$ is finite and $\to_{\mathcal{R}}$-terminating

Drawbacks:

- very difficult to approximate
- sufficient but not necessary:

$$\begin{array}{rcl} \mathsf{f(a)} & \to & \mathsf{b} \\ \mathsf{a} & \to & \mathsf{a} \end{array}$$

The set $\{\mathsf{f}(x)\}$ is $\leadsto_{\mathcal{R}}$-terminating
however $\{\mathsf{f(a)}\}$ is finite but not $\to_{\mathcal{R}}$-terminating:

$$\mathsf{f(a)} \to \mathsf{f(a)} \to \mathsf{f(a)} \to \ldots$$

Termination of narrowing via termination of rewriting

Theorem

T is $\rightsquigarrow_{\mathcal{R}}$-terminating
if $\{t\sigma \mid t \in T$ and $t \rightsquigarrow_\sigma^* s$ in $\mathcal{R}\}$ is finite and $\to_{\mathcal{R}}$-terminating

Drawbacks:

- very difficult to approximate
- sufficient but not necessary:

$$\begin{array}{rcl} \mathsf{f(a)} & \to & \mathsf{b} \\ \mathsf{a} & \to & \mathsf{a} \end{array}$$

The set $\{\mathsf{f}(x)\}$ is $\rightsquigarrow_{\mathcal{R}}$-terminating
however $\{\mathsf{f(a)}\}$ is finite but not $\to_{\mathcal{R}}$-terminating:

$$\mathsf{f(a)} \to \mathsf{f(a)} \to \mathsf{f(a)} \to \ldots$$

Termination of narrowing via termination of rewriting

Theorem

T is $\leadsto_{\mathcal{R}}$-terminating
if $\{t\sigma \mid t \in T \text{ and } t \leadsto_{\sigma}^{*} s \text{ in } \mathcal{R}\}$ is finite and $\to_{\mathcal{R}}$-terminating

Drawbacks:

- very difficult to approximate
- sufficient but not necessary:

$$\begin{array}{rcl} \mathsf{f(a)} & \to & \mathsf{b} \\ \mathsf{a} & \to & \mathsf{a} \end{array}$$

The set $\{\mathsf{f}(x)\}$ is $\leadsto_{\mathcal{R}}$-terminating
however $\{\mathsf{f(a)}\}$ is finite but not $\to_{\mathcal{R}}$-terminating:

$$\mathsf{f(a)} \to \mathsf{f(a)} \to \mathsf{f(a)} \to \ldots$$

A first solution

Variables in narrowing can be seen as generators of possibly infinite terms

Therefore $\{t\sigma \mid t \in T \text{ and } t \leadsto_\sigma^* s \text{ in } \mathcal{R}\}$

$$\Downarrow$$

$$\{t\sigma \mid t \in T \text{ and } \sigma \text{ maps variables to possibly infinite terms }\}$$

Why infinite terms?

Example

$$\begin{array}{rcll} \mathsf{add}(\mathsf{z}, y) & \to & y & (R_1) \\ \mathsf{add}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{add}(x, y)) & (R_2) \end{array}$$

- $\mathsf{add}(x, \mathsf{z})$ is $\to_\mathcal{R}$-terminating for any σ mapping x to a finite term
- **however**, if σ maps x to an infinite term of the form $\mathsf{s}(\mathsf{s}(\ldots))$, then the derivation for $\mathsf{add}(x, \mathsf{z})\sigma$ is now infinite:

$$\mathsf{add}(\mathsf{s}(\mathsf{s}(\ldots)), \mathsf{z}) \to_\mathcal{R} \mathsf{s}(\mathsf{add}(\mathsf{s}(\mathsf{s}(\ldots)), \mathsf{z})) \to_\mathcal{R} \ldots$$

A first solution

Variables in narrowing can be seen as generators of possibly infinite terms

Therefore $\{t\sigma \mid t \in T \text{ and } t \leadsto^*_\sigma s \text{ in } \mathcal{R}\}$

$$\Downarrow$$

$$\{t\sigma \mid t \in T \text{ and } \sigma \text{ maps variables to possibly infinite terms }\}$$

Why infinite terms?

Example

$$\begin{array}{rcll} \mathsf{add}(\mathsf{z}, y) & \to & y & (R_1) \\ \mathsf{add}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{add}(x, y)) & (R_2) \end{array}$$

- $\mathsf{add}(x, \mathsf{z})$ is $\to_\mathcal{R}$-terminating for any σ mapping x to a finite term
- **however**, if σ maps x to an infinite term of the form $\mathsf{s}(\mathsf{s}(\ldots))$, then the derivation for $\mathsf{add}(x, \mathsf{z})\sigma$ is now infinite:

$$\mathsf{add}(\mathsf{s}(\mathsf{s}(\ldots)), \mathsf{z}) \to_\mathcal{R} \mathsf{s}(\mathsf{add}(\mathsf{s}(\mathsf{s}(\ldots)), \mathsf{z})) \to_\mathcal{R} \ldots$$

A first solution

Variables in narrowing can be seen as generators of possibly infinite terms

Therefore $\{t\sigma \mid t \in T \text{ and } t \leadsto^*_\sigma s \text{ in } \mathcal{R}\}$

$$\Downarrow$$

$$\{t\sigma \mid t \in T \text{ and } \sigma \text{ maps variables to possibly infinite terms }\}$$

Why infinite terms?

Example

$$\begin{array}{rcll} \mathsf{add}(\mathsf{z}, y) & \to & y & (R_1) \\ \mathsf{add}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{add}(x, y)) & (R_2) \end{array}$$

- $\mathsf{add}(x, \mathsf{z})$ is $\to_\mathcal{R}$-terminating for any σ mapping x to a finite term
- **however**, if σ maps x to an infinite term of the form $\mathsf{s}(\mathsf{s}(\ldots))$, then the derivation for $\mathsf{add}(x, \mathsf{z})\sigma$ is now infinite:

$$\mathsf{add}(\mathsf{s}(\mathsf{s}(\ldots)), \mathsf{z}) \to_\mathcal{R} \mathsf{s}(\mathsf{add}(\mathsf{s}(\mathsf{s}(\ldots)), \mathsf{z})) \to_\mathcal{R} \ldots$$

Problem

proving that the set

$$\{t\sigma \mid t \in T \text{ and } \sigma \text{ maps variables to possibly infinite terms } \}$$

is $\rightarrow_{\mathcal{R}}$-terminating is often **too strong**...

Example Given the TRS

$$\begin{array}{rcl} \mathsf{a} & \rightarrow & \mathsf{a} \\ \mathsf{f}(x) & \rightarrow & x \end{array}$$

$\mathsf{f}(x)$ is clearly $\rightsquigarrow_{\mathcal{R}}$-terminating
but $\exists\sigma$ such that $\mathsf{f}(x)\sigma$ is not $\rightarrow_{\mathcal{R}}$-terminating (e.g., $\sigma = \{x \mapsto \mathsf{a}\}$)

$\Rightarrow$ an infinite computation $\mathsf{f}(\mathsf{a}) \rightarrow_{\mathcal{R}} \mathsf{f}(\mathsf{a}) \rightarrow_{\mathcal{R}} \ldots$ is introduced by σ !!

Problem

proving that the set

$$\{t\sigma \mid t \in T \text{ and } \sigma \text{ maps variables to possibly infinite terms }\}$$

is $\rightarrow_{\mathcal{R}}$-terminating is often **too strong**...

Example Given the TRS

$$\begin{array}{rcl} \mathsf{a} & \rightarrow & \mathsf{a} \\ \mathsf{f}(x) & \rightarrow & x \end{array}$$

$\mathsf{f}(x)$ is clearly $\rightsquigarrow_{\mathcal{R}}$-terminating

but $\exists\sigma$ such that $\mathsf{f}(x)\sigma$ is not $\rightarrow_{\mathcal{R}}$-terminating (e.g., $\sigma = \{x \mapsto \mathsf{a}\}$)

$\Rightarrow$ an infinite computation $\mathsf{f}(\mathsf{a}) \rightarrow_{\mathcal{R}} \mathsf{f}(\mathsf{a}) \rightarrow_{\mathcal{R}} \ldots$ is introduced by σ !!

Problem

proving that the set

$$\{t\sigma \mid t \in T \text{ and } \sigma \text{ maps variables to possibly infinite terms }\}$$

is $\rightarrow_{\mathcal{R}}$-terminating is often **too strong**...

Example Given the TRS

$$\begin{array}{rcl} \mathsf{a} & \rightarrow & \mathsf{a} \\ \mathsf{f}(x) & \rightarrow & x \end{array}$$

$\mathsf{f}(x)$ is clearly $\leadsto_{\mathcal{R}}$-terminating
but $\exists\sigma$ such that $\mathsf{f}(x)\sigma$ is not $\rightarrow_{\mathcal{R}}$-terminating (e.g., $\sigma = \{x \mapsto \mathsf{a}\}$)

$\Rightarrow$ an infinite computation $\mathsf{f}(\mathsf{a}) \rightarrow_{\mathcal{R}} \mathsf{f}(\mathsf{a}) \rightarrow_{\mathcal{R}} \ldots$ is introduced by σ !!

A second solution

$\Rightarrow$ forbid the reduction of redexes introduced by σ...

A second problem...

...this restriction makes the condition unsound!

Example Given the TRS

$$\begin{array}{rcl} \mathsf{a} & \to & \mathsf{a} \\ \mathsf{f(a)} & \to & \mathsf{c(b,b)} \end{array}$$

- $\mathsf{c}(y, \mathsf{f}(y))\sigma$ is $\to_{\mathcal{R}}$-terminating if the reduction of the terms introduced by σ is forbidden
- but $\mathsf{c}(y, \mathsf{f}(y))$ is not $\rightsquigarrow_{\mathcal{R}}$-terminating!!

  (e.g., $\mathsf{c}(y, \mathsf{f}(y)) \rightsquigarrow_{\{y \mapsto \mathsf{a}\}} \mathsf{c}(\mathsf{a}, \mathsf{c}(\mathsf{b}, \mathsf{b})) \rightsquigarrow_{id} \mathsf{c}(\mathsf{a}, \mathsf{c}(\mathsf{b}, \mathsf{b})) \rightsquigarrow_{id} \ldots$)

A second solution

$\Rightarrow$ forbid the reduction of redexes introduced by σ...

A second problem...

...this restriction makes the condition unsound!

Example Given the TRS

$$\begin{array}{rcl} \mathsf{a} & \rightarrow & \mathsf{a} \\ \mathsf{f(a)} & \rightarrow & \mathsf{c(b,b)} \end{array}$$

- $\mathsf{c}(y, \mathsf{f}(y))\sigma$ is $\rightarrow_{\mathcal{R}}$-terminating if the reduction of the terms introduced by σ is forbidden
- but $\mathsf{c}(y, \mathsf{f}(y))$ is not $\leadsto_{\mathcal{R}}$-terminating!!

  (e.g., $\mathsf{c}(y, \mathsf{f}(y)) \leadsto_{\{y \mapsto \mathsf{a}\}} \mathsf{c}(\mathsf{a}, \mathsf{c}(\mathsf{b}, \mathsf{b})) \leadsto_{id} \mathsf{c}(\mathsf{a}, \mathsf{c}(\mathsf{b}, \mathsf{b})) \leadsto_{id} \ldots$)

A second solution

$\Rightarrow$ forbid the reduction of redexes introduced by σ...

A second problem...

...this restriction makes the condition unsound!

Example Given the TRS

$$\begin{array}{rcl} \mathsf{a} & \rightarrow & \mathsf{a} \\ \mathsf{f}(\mathsf{a}) & \rightarrow & \mathsf{c}(\mathsf{b},\mathsf{b}) \end{array}$$

- $\mathsf{c}(y,\mathsf{f}(y))\sigma$ is $\rightarrow_{\mathcal{R}}$-terminating if the reduction of the terms introduced by σ is forbidden
- but $\mathsf{c}(y,\mathsf{f}(y))$ is not $\leadsto_{\mathcal{R}}$-terminating!!
  (e.g., $\mathsf{c}(y,\mathsf{f}(y)) \leadsto_{\{y\mapsto \mathsf{a}\}} \mathsf{c}(\mathsf{a},\mathsf{c}(\mathsf{b},\mathsf{b})) \leadsto_{id} \mathsf{c}(\mathsf{a},\mathsf{c}(\mathsf{b},\mathsf{b})) \leadsto_{id} \ldots$)

A second solution

$\Rightarrow$ forbid the reduction of redexes introduced by σ...

A second problem...

...this restriction makes the condition unsound!

Example Given the TRS

$$\begin{array}{rcl} \mathsf{a} & \rightarrow & \mathsf{a} \\ \mathsf{f}(\mathsf{a}) & \rightarrow & \mathsf{c}(\mathsf{b},\mathsf{b}) \end{array}$$

- $\mathsf{c}(y,\mathsf{f}(y))\sigma$ is $\rightarrow_{\mathcal{R}}$-terminating if the reduction of the terms introduced by σ is forbidden
- but $\mathsf{c}(y,\mathsf{f}(y))$ is not $\leadsto_{\mathcal{R}}$-terminating!!
  (e.g., $\mathsf{c}(y,\mathsf{f}(y)) \leadsto_{\{y\mapsto \mathsf{a}\}} \mathsf{c}(\mathsf{a},\mathsf{c}(\mathsf{b},\mathsf{b})) \leadsto_{id} \mathsf{c}(\mathsf{a},\mathsf{c}(\mathsf{b},\mathsf{b})) \leadsto_{id} \ldots$)

Last (good) solution

$$\Rightarrow \begin{cases} \text{restrict to narrowing derivations} \\ \text{where terms introduced by instantiation cannot be narrowed!} \end{cases}$$

For instance,

- (innermost) basic narrowing over arbitrary TRSs
- lazy and needed narrowing over left-linear constructor TRSs
- ...

Any narrowing strategy over left-linear constructor TRSs can only introduce constructor substitutions $\Longrightarrow$

Termination of narrowing via termination of rewriting

In the following, we consider **left-linear** **constructor** TRSs:

$$\begin{array}{rcl} f_1(t_{11},\ldots,t_{1m_1}) & \rightarrow & r_1 \\ & \ldots & \\ f_n(t_{n1},\ldots,t_{nm_n}) & \rightarrow & r_n \end{array}$$

with

- $f_i(t_{i1},\ldots,t_{in_i})$ linear (no multiple occurrences of the same variable)
- $t_{i1},\ldots,t_{in_i}$ constructor terms (no occurrence of $f_1,\ldots,f_n$)

Property variables are bound to (irreducible) constructor terms

$\Downarrow$

Our approach we replace variables by "data generators" that only produce (ground) constructor terms

Data generators [Antoy, Hanus, 2006; de Dios-Castro, López-Fraguas 2006]

For every TRS $\mathcal{R}$, we define $\mathcal{R}_{\mathsf{gen}}$ as $\mathcal{R}$ augmented with

$$\boxed{\mathsf{gen} \quad \to \quad \mathsf{c}(\overbrace{\mathsf{gen}, \ldots, \mathsf{gen}}^{n\ times})}$$

for all constructor $\mathsf{c}/n \in \mathcal{C}, \;\; n \geqslant 0$

E.g., for $\mathcal{C} = \{\mathsf{z}/0, \mathsf{s}/1\}$, we have

$$\mathcal{R}_{\mathsf{gen}} = \mathcal{R} \cup \left\{ \begin{array}{lcl} \mathsf{gen} & \to & \mathsf{z} \\ \mathsf{gen} & \to & \mathsf{s}(\mathsf{gen}) \end{array} \right\}$$

Some notation: $\widehat{t} = t\sigma$, with $\sigma = \{x \mapsto \mathsf{gen} \mid x \in \mathcal{V}ar(t)\}$

Data generators [Antoy, Hanus, 2006; de Dios-Castro, López-Fraguas 2006]

For every TRS $\mathcal{R}$, we define $\mathcal{R}_{\mathsf{gen}}$ as $\mathcal{R}$ augmented with

$$\mathsf{gen} \;\to\; \mathsf{c}(\overbrace{\mathsf{gen},\ldots,\mathsf{gen}}^{n\ \textit{times}})$$

for all constructor $\mathsf{c}/n \in \mathcal{C}$, $n \geqslant 0$

E.g., for $\mathcal{C} = \{\mathsf{z}/0, \mathsf{s}/1\}$, we have

$$\mathcal{R}_{\mathsf{gen}} = \mathcal{R} \cup \left\{ \begin{array}{lcl} \mathsf{gen} & \to & \mathsf{z} \\ \mathsf{gen} & \to & \mathsf{s}(\mathsf{gen}) \end{array} \right\}$$

Some notation: $\widehat{t} = t\sigma$, with $\sigma = \{x \mapsto \mathsf{gen} \mid x \in \mathcal{V}ar(t)\}$

Data generators [Antoy, Hanus, 2006; de Dios-Castro, López-Fraguas 2006]

For every TRS $\mathcal{R}$, we define $\mathcal{R}_{\mathsf{gen}}$ as $\mathcal{R}$ augmented with

$$\boxed{\mathsf{gen} \;\to\; \mathsf{c}(\overbrace{\mathsf{gen},\ldots,\mathsf{gen}}^{n\ \textit{times}})}$$

for all constructor $\mathsf{c}/n \in \mathcal{C}$, $n \geqslant 0$

E.g., for $\mathcal{C} = \{\mathsf{z}/0, \mathsf{s}/1\}$, we have

$$\mathcal{R}_{\mathsf{gen}} = \mathcal{R} \cup \left\{ \begin{array}{lcl} \mathsf{gen} & \to & \mathsf{z} \\ \mathsf{gen} & \to & \mathsf{s}(\mathsf{gen}) \end{array} \right\}$$

Some notation: $\widehat{t} = t\sigma$, with $\sigma = \{x \mapsto \mathsf{gen} \mid x \in \mathcal{V}ar(t)\}$

Correctness of data generators

Completeness

If $s \leadsto_\sigma t$ in $\mathcal{R}$ then $\widehat{s} \to^*_{\mathsf{gen}} \widehat{s\sigma} \to \widehat{t}$ in $\mathcal{R}_{\mathsf{gen}}$

Generally unsound

E.g., $\mathsf{add}(\mathsf{gen},\mathsf{gen}) \to \mathsf{add}(\mathsf{z},\mathsf{gen}) \to \mathsf{gen} \to \mathsf{s}(\mathsf{gen}) \to \mathsf{s}(\mathsf{z})$

but $\mathsf{add}(x,x) \leadsto_{\{x\mapsto \mathsf{z}\}} \mathsf{z}$

$\mathsf{add}(x,x) \leadsto_{\{x\mapsto \mathsf{s}(x')\}} \mathsf{s}(\mathsf{add}(x',\mathsf{s}(x'))) \leadsto_{\{x'\mapsto \mathsf{z}\}} \mathsf{s}(\mathsf{s}(\mathsf{z}))$

$\ldots$

Soundness is preserved for admissible derivations

- a derivation is admissible iff all the occurrences of gen originating from the replacement of the same variable are reduced to the same term

Correctness of data generators

Completeness

If $s \leadsto_\sigma t$ in $\mathcal{R}$ then $\widehat{s} \rightarrow^*_{\text{gen}} \widehat{s\sigma} \rightarrow \widehat{t}$ in $\mathcal{R}_{\text{gen}}$

Generally unsound

E.g., $\mathsf{add}(\mathsf{gen}, \mathsf{gen}) \rightarrow \mathsf{add}(\mathsf{z}, \mathsf{gen}) \rightarrow \mathsf{gen} \rightarrow \mathsf{s}(\mathsf{gen}) \rightarrow \mathsf{s}(\mathsf{z})$

but $\mathsf{add}(x, x) \leadsto_{\{x \mapsto \mathsf{z}\}} \mathsf{z}$

$\mathsf{add}(x, x) \leadsto_{\{x \mapsto \mathsf{s}(x')\}} \mathsf{s}(\mathsf{add}(x', \mathsf{s}(x'))) \leadsto_{\{x' \mapsto \mathsf{z}\}} \mathsf{s}(\mathsf{s}(\mathsf{z}))$

$\ldots$

Soundness is preserved for admissible derivations

- a derivation is admissible iff all the occurrences of gen originating from the replacement of the same variable are reduced to the same term

Correctness of data generators

Completeness

If $s \rightsquigarrow_\sigma t$ in $\mathcal{R}$ then $\widehat{s} \to^*_{\mathsf{gen}} \widehat{s\sigma} \to \widehat{t}$ in $\mathcal{R}_{\mathsf{gen}}$

Generally unsound

E.g., $\mathsf{add}(\mathsf{gen},\mathsf{gen}) \to \mathsf{add}(\mathsf{z},\mathsf{gen}) \to \mathsf{gen} \to \mathsf{s}(\mathsf{gen}) \to \mathsf{s}(\mathsf{z})$

but $\mathsf{add}(x,x) \rightsquigarrow_{\{x\mapsto \mathsf{z}\}} \mathsf{z}$

$\mathsf{add}(x,x) \rightsquigarrow_{\{x\mapsto \mathsf{s}(x')\}} \mathsf{s}(\mathsf{add}(x',\mathsf{s}(x'))) \rightsquigarrow_{\{x'\mapsto \mathsf{z}\}} \mathsf{s}(\mathsf{s}(\mathsf{z}))$

$\ldots$

Soundness is preserved for admissible derivations

- a derivation is admissible iff all the occurrences of gen originating from the replacement of the same variable are reduced to the same term

What about termination in $\mathcal{R}_{\text{gen}}$?

Clearly, no term with occurrences of gen terminates!

Fortunately, **relative termination** of $\mathcal{R}_{\text{gen}}$ suffices:

- T is relatively $\mathcal{R}_{\text{gen}}$-terminating to $\mathcal{R}$ if every derivation $t_1 \to t_2 \ldots$ for $t_1 \in T$ contains finitely many $\to_{\mathcal{R}}$ steps

Theorem (termination of narrowing via termination of rewriting)

Let $\mathcal{R}$ be a left-linear constructor TRS
T is $\leadsto_{\mathcal{R}}$-terminating
$\widehat{T}$ is relatively $\to_{\mathcal{R}_{\text{gen}}}$-terminating to $\mathcal{R}$

$\Longrightarrow$ **sufficient condition**

What about termination in $\mathcal{R}_{\text{gen}}$?

Clearly, no term with occurrences of gen terminates!

Fortunately, **relative termination** of $\mathcal{R}_{\text{gen}}$ suffices:

- T is relatively $\mathcal{R}_{\text{gen}}$-terminating to $\mathcal{R}$ if every derivation $t_1 \to t_2 \ldots$ for $t_1 \in T$ contains finitely many $\to_{\mathcal{R}}$ steps

Theorem (termination of narrowing via termination of rewriting)

Let $\mathcal{R}$ be a left-linear constructor TRS
T is $\leadsto_{\mathcal{R}}$-terminating
$\widehat{T}$ is relatively $\to_{\mathcal{R}_{\text{gen}}}$-terminating to $\mathcal{R}$

$\Longrightarrow$ **sufficient condition**

What about termination in $\mathcal{R}_{\text{gen}}$?

Clearly, no term with occurrences of gen terminates!

Fortunately, **relative termination** of $\mathcal{R}_{\text{gen}}$ suffices:

- T is relatively $\mathcal{R}_{\text{gen}}$-terminating to $\mathcal{R}$ if every derivation $t_1 \to t_2 \ldots$ for $t_1 \in T$ contains finitely many $\to_{\mathcal{R}}$ steps

Theorem (termination of narrowing via termination of rewriting)

Let $\mathcal{R}$ be a left-linear constructor TRS
T is $\leadsto_{\mathcal{R}}$-terminating
iff
$\widehat{T}$ is relatively $\to_{\mathcal{R}_{\text{gen}}}$-terminating to $\mathcal{R}$ w.r.t. admissible derivations

$\Longrightarrow$ **sufficient condition**

What about termination in $\mathcal{R}_{\text{gen}}$?

Clearly, no term with occurrences of gen terminates!

Fortunately, **relative termination** of $\mathcal{R}_{\text{gen}}$ suffices:

- T is relatively $\mathcal{R}_{\text{gen}}$-terminating to $\mathcal{R}$ if every derivation $t_1 \to t_2 \dots$ for $t_1 \in T$ contains finitely many $\to_{\mathcal{R}}$ steps

Theorem (termination of narrowing via termination of rewriting)

Let $\mathcal{R}$ be a left-linear constructor TRS
T is $\leadsto_{\mathcal{R}}$-terminating
~~*iff*~~ *if*
$\widehat{T}$ is relatively $\to_{\mathcal{R}_{\text{gen}}}$-terminating to $\mathcal{R}$ ~~*w.r.t. admissible derivations*~~

$\Longrightarrow$ **sufficient condition**

automating the termination analysis

Proving termination automatically

The problem

Given $\mathcal{R}$ and T,

T is $\rightsquigarrow_{\mathcal{R}}$-terminating if $\widehat{T}$ is relatively $\rightarrow_{\mathcal{R}_{\text{gen}}}$-terminating to $\mathcal{R}$

Drawback

- the set T is generally infinite

Solution: use abstract terms

- similar to modes in logic programming
- E.g., add(g, v) denotes the set of terms add(t_1, t_2) with
  - t_1 (definitely) ground
  - t_2 (possibly) variable
- concretization funcion γ,

  e.g., $\gamma(\text{add}(g, v)) = \{\text{add}(z, x), \text{add}(z, z), \text{add}(s(z), x), \text{add}(s(z), z), \ldots\}$

Proving termination automatically

The problem

Given $\mathcal{R}$ and T,

T is $\rightsquigarrow_{\mathcal{R}}$-terminating if $\widehat{T}$ is relatively $\rightarrow_{\mathcal{R}_{\text{gen}}}$-terminating to $\mathcal{R}$

Drawback

- the set T is generally infinite

Solution: use abstract terms

- similar to modes in logic programming
- E.g., add(g, v) denotes the set of terms add(t_1, t_2) with
  - t_1 (definitely) ground
  - t_2 (possibly) variable
- concretization funcion γ,

  e.g., $\gamma(\text{add}(g, v)) = \{\text{add}(z, x), \text{add}(z, z), \text{add}(s(z), x), \text{add}(s(z), z), \ldots\}$

Proving termination automatically

The problem

Given $\mathcal{R}$ and T,

T is $\leadsto_{\mathcal{R}}$-terminating if $\widehat{T}$ is relatively $\rightarrow_{\mathcal{R}_{\text{gen}}}$-terminating to $\mathcal{R}$

Drawback

- the set T is generally infinite

Solution: use abstract terms

- similar to modes in logic programming
- E.g., $\mathsf{add}(g, v)$ denotes the set of terms $\mathsf{add}(t_1, t_2)$ with
  - t_1 (definitely) **g**round
  - t_2 (possibly) **v**ariable
- concretization funcion γ,

  e.g., $\gamma(\mathsf{add}(g, v)) = \{\mathsf{add}(\mathsf{z}, x), \mathsf{add}(\mathsf{z}, \mathsf{z}), \mathsf{add}(\mathsf{s}(\mathsf{z}), x), \mathsf{add}(\mathsf{s}(\mathsf{z}), \mathsf{z}), \ldots\}$

Proving termination automatically

The problem (revised)

Given $\mathcal{R}$ and t^α,

$\gamma(t^\alpha)$ is $\rightsquigarrow_{\mathcal{R}}$-terminating if $\widehat{\gamma(t^\alpha)}$ is relatively $\rightarrow_{\mathcal{R}_{\mathsf{gen}}}$-terminating to $\mathcal{R}$

Drawback

- checking relative termination requires non-standard techniques

Solution: use argument filterings

- to filter away non-ground arguments of terms
  (equivalently, to filter away occurrences of gen)

Proving termination automatically

The problem (revised)

Given $\mathcal{R}$ and t^α,

$\gamma(t^\alpha)$ is $\leadsto_{\mathcal{R}}$-terminating if $\widehat{\gamma(t^\alpha)}$ is relatively $\rightarrow_{\mathcal{R}_{\mathsf{gen}}}$-terminating to $\mathcal{R}$

Drawback

- checking relative termination requires non-standard techniques

Solution: use argument filterings

- to filter away non-ground arguments of terms
  (equivalently, to filter away occurrences of gen)

Proving termination automatically

The problem (revised)

Given $\mathcal{R}$ and t^α,

$\gamma(t^\alpha)$ is $\leadsto_{\mathcal{R}}$-terminating if $\widehat{\gamma(t^\alpha)}$ is relatively $\rightarrow_{\mathcal{R}_{\mathsf{gen}}}$-terminating to $\mathcal{R}$

Drawback

- checking relative termination requires non-standard techniques

Solution: use argument filterings

- to filter away non-ground arguments of terms
  (equivalently, to filter away occurrences of gen)

Argument filterings [Kusakari, Nakamura, Toyama 1999]

$\boxed{\pi(\mathsf{f}) \subseteq \{1, \ldots, n\} \text{ for every defined function } \mathsf{f}/n}$

Argument filterings over terms & TRSs:

$$\pi(t) = \begin{cases} x & \text{if } t = x \\ \mathsf{c}(\pi(t_1), \ldots, \pi(t_n)) & \text{if } t = \mathsf{c}(t_1, \ldots, t_n) \\ \mathsf{f}(\pi(t_{i_1}), \ldots, \pi(t_{i_m})) & \text{if } t = \mathsf{f}(t_1, \ldots, t_n) \text{ and } \pi(\mathsf{f}) = \{i_1, \ldots, i_m\} \end{cases}$$

$$\pi(l \to r) \;=\; \pi(l) \to \pi(r)$$

From t^α we infer a safe argument filtering π for t^α

- $\pi(t^\alpha) = \mathsf{f}(g, g, \ldots, g)$
- for all $s \rightsquigarrow t$, if $\pi(s|_p)$ are ground then $\pi(t|_q)$ are ground too

Argument filterings [Kusakari, Nakamura, Toyama 1999]

$\boxed{\pi(\mathsf{f}) \subseteq \{1,\ldots,n\} \text{ for every defined function } \mathsf{f}/n}$

Argument filterings over terms & TRSs:

$$\pi(t) = \begin{cases} x & \text{if } t = x \\ \mathsf{c}(\pi(t_1),\ldots,\pi(t_n)) & \text{if } t = \mathsf{c}(t_1,\ldots,t_n) \\ \mathsf{f}(\pi(t_{i_1}),\ldots,\pi(t_{i_m})) & \text{if } t = \mathsf{f}(t_1,\ldots,t_n) \text{ and } \pi(\mathsf{f}) = \{i_1,\ldots,i_m\} \end{cases}$$

$$\pi(l \to r) \;=\; \pi(l) \to \pi(r)$$

From t^α we infer a safe argument filtering π for t^α

- $\pi(t^\alpha) = \mathsf{f}(g, g, \ldots, g)$
- for all $s \rightsquigarrow t$, if $\pi(s|_p)$ are ground then $\pi(t|_q)$ are ground too

Proving termination automatically: approaches

A direct approach

- based on dependency pairs [Arts, Giesl 2000]
- only a slight extension needed

A transformational approach

- based on argument filtering transformation [Kusakari, Nakamura, Toyama 1999]
- we use a simplified version (except for extra-variables)

Dependency pairs approach

Dependency pairs $DP(\mathcal{R})$ of a TRS $\mathcal{R}$

$$DP(\mathcal{R}) = \{\ F(s_1,\ldots,s_n) \to G(t_1,\ldots,t_m) \mid f(s_1,\ldots,s_n) \to r \in \mathcal{R},\ r|_p = g(t_1,\ldots,t_m)\ \}$$

where F, G are tuple symbols

Example

$$\begin{aligned}
\mathrm{append}(\mathrm{nil}, y) &\to y \\
\mathrm{append}(\mathrm{cons}(x, xs), y) &\to \mathrm{cons}(x, \mathrm{append}(xs, y)) \\
\mathrm{reverse}(\mathrm{nil}) &\to \mathrm{nil} \\
\mathrm{reverse}(\mathrm{cons}(x, xs)) &\to \mathrm{append}(\mathrm{reverse}(xs), \mathrm{cons}(x, \mathrm{nil}))
\end{aligned}$$

$$\begin{aligned}
\mathrm{APPEND}(\mathrm{cons}(x, xs), y) &\to \mathrm{APPEND}(xs, y) && (1) \\
\mathrm{REVERSE}(\mathrm{cons}(x, xs)) &\to \mathrm{REVERSE}(xs) && (2) \\
\mathrm{REVERSE}(\mathrm{cons}(x, xs)) &\to \mathrm{APPEND}(\mathrm{reverse}(xs), \mathrm{cons}(x, \mathrm{nil})) && (3)
\end{aligned}$$

Dependency pairs approach

Dependency pairs $DP(\mathcal{R})$ of a TRS $\mathcal{R}$

$$DP(\mathcal{R}) = \{\ F(s_1,\ldots,s_n) \to G(t_1,\ldots,t_m) \mid \begin{array}{l} f(s_1,\ldots,s_n) \to r \in \mathcal{R} \\ r|_p = g(t_1,\ldots,t_m) \end{array} \ \}$$

where F, G are tuple symbols

Example

$$\begin{aligned}
\mathrm{append}(\mathrm{nil}, y) &\to y \\
\mathrm{append}(\mathrm{cons}(x, xs), y) &\to \mathrm{cons}(x, \mathrm{append}(xs, y)) \\
\mathrm{reverse}(\mathrm{nil}) &\to \mathrm{nil} \\
\mathrm{reverse}(\mathrm{cons}(x, xs)) &\to \mathrm{append}(\mathrm{reverse}(xs), \mathrm{cons}(x, \mathrm{nil}))
\end{aligned}$$

$$\begin{aligned}
\mathrm{APPEND}(\mathrm{cons}(x, xs), y) &\to \mathrm{APPEND}(xs, y) && (1)\\
\mathrm{REVERSE}(\mathrm{cons}(x, xs)) &\to \mathrm{REVERSE}(xs) && (2)\\
\mathrm{REVERSE}(\mathrm{cons}(x, xs)) &\to \mathrm{APPEND}(\mathrm{reverse}(xs), \mathrm{cons}(x, \mathrm{nil})) && (3)
\end{aligned}$$

Dependency pairs approach

Dependency pairs $DP(\mathcal{R})$ of a TRS $\mathcal{R}$

$$DP(\mathcal{R}) = \{\ F(s_1,\ldots,s_n) \to G(t_1,\ldots,t_m) \mid \begin{array}{l} f(s_1,\ldots,s_n) \to r \in \mathcal{R} \\ r|_p = g(t_1,\ldots,t_m) \end{array} \}$$

where F, G are tuple symbols

Example

$$\begin{array}{rcl} \mathrm{append}(\mathrm{nil}, y) & \to & y \\ \mathrm{append}(\mathrm{cons}(x, xs), y) & \to & \mathrm{cons}(x, \mathrm{append}(xs, y)) \\ \mathrm{reverse}(\mathrm{nil}) & \to & \mathrm{nil} \\ \mathrm{reverse}(\mathrm{cons}(x, xs)) & \to & \mathrm{append}(\mathrm{reverse}(xs), \mathrm{cons}(x, \mathrm{nil})) \end{array}$$

$$\begin{array}{rcll} \mathrm{APPEND}(\mathrm{cons}(x, xs), y) & \to & \mathrm{APPEND}(xs, y) & (1) \\ \mathrm{REVERSE}(\mathrm{cons}(x, xs)) & \to & \mathrm{REVERSE}(xs) & (2) \\ \mathrm{REVERSE}(\mathrm{cons}(x, xs)) & \to & \mathrm{APPEND}(\mathrm{reverse}(xs), \mathrm{cons}(x, \mathrm{nil})) & (3) \end{array}$$

Dependency pairs approach

Dependency pairs $DP(\mathcal{R})$ of a TRS $\mathcal{R}$

$$DP(\mathcal{R}) = \{\ F(s_1,\ldots,s_n) \to G(t_1,\ldots,t_m) \mid \begin{array}{l} f(s_1,\ldots,s_n) \to r \in \mathcal{R} \\ r|_p = g(t_1,\ldots,t_m) \end{array} \ \}$$

where F, G are tuple symbols

Example

$$\begin{aligned}
\mathrm{append}(\mathrm{nil}, y) &\to y\\
\mathrm{append}(\mathrm{cons}(x, xs), y) &\to \mathrm{cons}(x, \mathrm{append}(xs, y))\\
\mathrm{reverse}(\mathrm{nil}) &\to \mathrm{nil}\\
\mathrm{reverse}(\mathrm{cons}(x, xs)) &\to \mathrm{append}(\mathrm{reverse}(xs), \mathrm{cons}(x, \mathrm{nil}))
\end{aligned}$$

$$\begin{aligned}
\mathrm{APPEND}(\mathrm{cons}(x, xs), y) &\to \mathrm{APPEND}(xs, y) & (1)\\
\mathrm{REVERSE}(\mathrm{cons}(x, xs)) &\to \mathrm{REVERSE}(xs) & (2)\\
\mathrm{REVERSE}(\mathrm{cons}(x, xs)) &\to \mathrm{APPEND}(\mathrm{reverse}(xs), \mathrm{cons}(x, \mathrm{nil})) & (3)
\end{aligned}$$

Dependency pairs approach

Dependency pairs $DP(\mathcal{R})$ of a TRS $\mathcal{R}$

$$DP(\mathcal{R}) = \{ \; F(s_1,\ldots,s_n) \to G(t_1,\ldots,t_m) \mid \begin{array}{l} f(s_1,\ldots,s_n) \to r \in \mathcal{R} \\ r|_p = g(t_1,\ldots,t_m) \end{array} \; \}$$

where F, G are tuple symbols

Example

$$\begin{aligned}
\mathrm{append}(\mathrm{nil}, y) &\to y \\
\mathrm{append}(\mathrm{cons}(x, xs), y) &\to \mathrm{cons}(x, \mathrm{append}(xs, y)) \\
\mathrm{reverse}(\mathrm{nil}) &\to \mathrm{nil} \\
\mathrm{reverse}(\mathrm{cons}(x, xs)) &\to \mathrm{append}(\mathrm{reverse}(xs), \mathrm{cons}(x, \mathrm{nil}))
\end{aligned}$$

$$\begin{aligned}
\mathrm{APPEND}(\mathrm{cons}(x, xs), y) &\to \mathrm{APPEND}(xs, y) & (1)\\
\mathrm{REVERSE}(\mathrm{cons}(x, xs)) &\to \mathrm{REVERSE}(xs) & (2)\\
\mathrm{REVERSE}(\mathrm{cons}(x, xs)) &\to \mathrm{APPEND}(\mathrm{reverse}(xs), \mathrm{cons}(x, \mathrm{nil})) & (3)
\end{aligned}$$

Dependency pairs approach: differences

Definition (chain)

A (possibly infinite) sequence of dependency pairs $s_1 \to t_1,\ s_2 \to t_2, \ldots$ from $DP(\mathcal{R})$ is a $(DP(\mathcal{R}), \mathcal{R}, \pi)$-chain if

- $\exists$ (constructor) substitution σ such that $\widehat{t_i\sigma} \to^*_{\mathcal{R}_{\mathsf{gen}}} \widehat{s_{i+1}\sigma}$ for $i \geqslant 1$
- $\pi(\widehat{s_i\sigma}), \pi(\widehat{t_i\sigma})$ contain no occurrences of gen

Three main extensions w.r.t. the standard notion:

- it is parameterized by π
- variables are replaced by gen and reductions w.r.t. $\mathcal{R}_{\mathsf{gen}}$
- π should filter away all occurrences of gen

Dependency pairs approach: differences

Definition (chain)

A (possibly infinite) sequence of dependency pairs $s_1 \to t_1$, $s_2 \to t_2, \ldots$ from $DP(\mathcal{R})$ is a $(DP(\mathcal{R}), \mathcal{R}, \pi)$-chain if

- $\exists$ (constructor) substitution σ such that $\widehat{t_i\sigma} \to^*_{\mathcal{R}_{\mathrm{gen}}} \widehat{s_{i+1}\sigma}$ for $i \geqslant 1$
- $\pi(\widehat{s_i\sigma}), \pi(\widehat{t_i\sigma})$ contain no occurrences of gen

Three main extensions w.r.t. the standard notion:

- it is parameterized by π
- variables are replaced by gen and reductions w.r.t. $\mathcal{R}_{\mathrm{gen}}$
- π should filter away all occurrences of gen

Example

Given the dependency pair

$$\mathsf{APPEND}(\mathsf{cons}(x, xs), y) \to \mathsf{APPEND}(xs, y) \qquad (1)$$

we have an infinite $(DP(\mathcal{R}), \mathcal{R}, \pi)$-chain, (1),(1),..., for

$$\pi(\mathsf{append}) = \pi(\mathsf{APPEND}) = \{2\}$$

since there exists $\sigma = \{y \mapsto \mathsf{nil}\}$ such that

$$\begin{array}{ccc}
\mathsf{APPEND}(\mathsf{cons}(x, xs), y) & \to & \mathsf{APPEND}(xs, y) \quad (1) \\
\widehat{t\sigma}\uparrow & & \downarrow\widehat{t\sigma} \\
\mathsf{APPEND}(\mathsf{cons}(\mathsf{gen}, \mathsf{gen}), \mathsf{nil}) & \leftarrow & \mathsf{APPEND}(gen, \mathsf{nil})
\end{array}$$

where $\pi(\mathsf{APPEND}(\mathsf{gen}, \mathsf{nil})) = \pi(\mathsf{APPEND}(\mathsf{cons}(\mathsf{gen}, \mathsf{gen}), \mathsf{nil})) \in \mathcal{T}(\mathcal{F})$

(not a chain in the standard framework of rewriting)

Theorem

Let π be a safe argument filtering for t^α in $\mathcal{R}$
If there is no infinite $(DP(\mathcal{R}), \mathcal{R}, \pi)$-chain, then $\gamma(t^\alpha)$ is $\leadsto_{\mathcal{R}}$-terminating

Now, we could adapt the **processors** of the dependency pair framework...

Argument filtering processor

E.g., we prove the soundness of transforming the DP problem

$$(DP(\mathcal{R}), \mathcal{R}, \pi) \qquad \Longrightarrow \qquad (\pi(DP(\mathcal{R})), \pi(\mathcal{R}), id)$$

where $id(\mathsf{f}) = \{1, \ldots, n\}$ for all f/n occurring in $\pi(\mathcal{R})$

$(\pi(DP(\mathcal{R})), \pi(\mathcal{R}), id)$ is a standard DP problem, therefore,

- **all DP processors [GTSKF06] for proving the termination of rewriting can be used for proving the termination of narrowing**

Theorem

Let π be a safe argument filtering for t^α in $\mathcal{R}$
If there is no infinite $(DP(\mathcal{R}), \mathcal{R}, \pi)$-chain, then $\gamma(t^\alpha)$ is $\leadsto_{\mathcal{R}}$-terminating

Now, we could adapt the **processors** of the dependency pair framework. . .

Argument filtering processor

E.g., we prove the soundness of transforming the DP problem

$$(DP(\mathcal{R}), \mathcal{R}, \pi) \qquad \Longrightarrow \qquad (\pi(DP(\mathcal{R})), \pi(\mathcal{R}), id)$$

where $id(\mathsf{f}) = \{1, \ldots, n\}$ for all f/n occurring in $\pi(\mathcal{R})$

$(\pi(DP(\mathcal{R})), \pi(\mathcal{R}), id)$ is a standard DP problem, therefore,

- **all DP processors [GTSKF06] for proving the termination of rewriting can be used for proving the termination of narrowing**

Theorem

Let π be a safe argument filtering for t^α in $\mathcal{R}$
If there is no infinite $(DP(\mathcal{R}), \mathcal{R}, \pi)$-chain, then $\gamma(t^\alpha)$ is $\leadsto_{\mathcal{R}}$-terminating

Now, we could adapt the **processors** of the dependency pair framework...

Argument filtering processor

E.g., we prove the soundness of transforming the DP problem

$$(DP(\mathcal{R}), \mathcal{R}, \pi) \quad \Longrightarrow \quad (\pi(DP(\mathcal{R})), \pi(\mathcal{R}), id)$$

where $id(\mathsf{f}) = \{1, \ldots, n\}$ for all f/n occurring in $\pi(\mathcal{R})$

$(\pi(DP(\mathcal{R})), \pi(\mathcal{R}), id)$ is a standard DP problem, therefore,

- **all DP processors [GTSKF06] for proving the termination of rewriting can be used for proving the termination of narrowing**

Example

$t^\alpha = \mathsf{append}(g, v)$

$\pi = \{\mathsf{append} \mapsto \{1\},\ \mathsf{reverse} \mapsto \{1\}\}$ (π is safe for t^α)

The argument filtering processor returns:

Dependency pairs:
$$\left\{\begin{array}{lcl} \mathsf{APPEND}(\mathsf{cons}(x, xs)) & \to & \mathsf{APPEND}(xs) \\ \mathsf{REVERSE}(\mathsf{cons}(x, xs)) & \to & \mathsf{REVERSE}(xs) \\ \mathsf{REVERSE}(\mathsf{cons}(x, xs)) & \to & \mathsf{APPEND}(\mathsf{reverse}(xs)) \end{array}\right.$$

Rewrite system:
$$\left\{\begin{array}{rcl} \mathsf{append}(\mathsf{nil}) & \to & y \\ \mathsf{append}(\mathsf{cons}(x, xs)) & \to & \mathsf{cons}(x, \mathsf{append}(xs)) \\ \mathsf{reverse}(\mathsf{nil}) & \to & \mathsf{nil} \\ \mathsf{reverse}(\mathsf{cons}(x, xs)) & \to & \mathsf{append}(\mathsf{reverse}(xs)) \end{array}\right.$$

Argument filtering: $id = \{\mathsf{append} \mapsto \{1\},\ \mathsf{reverse} \mapsto \{1\}\}$

Example

$t^\alpha = \mathsf{append}(g, v)$
$\pi = \{\mathsf{append} \mapsto \{1\},\ \mathsf{reverse} \mapsto \{1\}\}$ (π is safe for t^α)

The argument filtering processor returns:

Dependency pairs:
$$\left\{ \begin{array}{lcl} \mathsf{APPEND}(\mathsf{cons}(x, xs)) & \to & \mathsf{APPEND}(xs) \\ \mathsf{REVERSE}(\mathsf{cons}(x, xs)) & \to & \mathsf{REVERSE}(xs) \\ \mathsf{REVERSE}(\mathsf{cons}(x, xs)) & \to & \mathsf{APPEND}(\mathsf{reverse}(xs)) \end{array} \right.$$

Rewrite system:
$$\left\{ \begin{array}{rcl} \mathsf{append}(\mathsf{nil}) & \to & y \qquad \text{PROBLEM!} \\ \mathsf{append}(\mathsf{cons}(x, xs)) & \to & \mathsf{cons}(x, \mathsf{append}(xs)) \\ \mathsf{reverse}(\mathsf{nil}) & \to & \mathsf{nil} \\ \mathsf{reverse}(\mathsf{cons}(x, xs)) & \to & \mathsf{append}(\mathsf{reverse}(xs)) \end{array} \right.$$

Argument filtering: $id = \{\mathsf{append} \mapsto \{1\},\ \mathsf{reverse} \mapsto \{1\}\}$

Removing extra-variables from filtered TRSs

Luckily, some extra-variables can be safely ignored...

- If the argument filtering is safe, extra-variables may only appear *above* the maximal function calls of the right-hand sides (thus $\pi(DP(\mathcal{R}))$ never contains extra-variables)

- As for $\pi(\mathcal{R})$, it should preserve the chains of dependency pairs:
  if $s_1 \to t_1$, $s_2 \to t_2$, $\ldots$ is a chain in $\mathcal{R}$
  then $\pi(s_1) \to \pi(t_1)$, $\pi(s_2) \to \pi(t_2)$, $\ldots$ should be a chain in $\pi(\mathcal{R})$

- For this purpose, it suffices to consider extra-vars in those functions
  - that are reachable from t^α and
  - occur below a maximal function call of the right-hand side
  - **all other extra-variables can be safely ignored**
    (e.g., replaced by a fresh constructor constant $\perp$)

Removing extra-variables from filtered TRSs

Luckily, some extra-variables can be safely ignored...

- If the argument filtering is safe, extra-variables may only appear *above* the maximal function calls of the right-hand sides (thus $\pi(DP(\mathcal{R}))$ never contains extra-variables)

- As for $\pi(\mathcal{R})$, it should preserve the chains of dependency pairs:
  if $s_1 \to t_1,\ s_2 \to t_2,\ \ldots$ is a chain in $\mathcal{R}$
  then $\pi(s_1) \to \pi(t_1),\ \pi(s_2) \to \pi(t_2),\ \ldots$ should be a chain in $\pi(\mathcal{R})$

- For this purpose, it suffices to consider extra-vars in those functions
  - that are reachable from t^α and
  - occur below a maximal function call of the right-hand side
  - $\boxed{\textbf{all other extra-variables can be safely ignored}}$
    (e.g., replaced by a fresh constructor constant $\perp$)

Removing extra-variables from filtered TRSs

Luckily, some extra-variables can be safely ignored...

- If the argument filtering is safe, extra-variables may only appear *above* the maximal function calls of the right-hand sides (thus $\pi(DP(\mathcal{R}))$ never contains extra-variables)

- As for $\pi(\mathcal{R})$, it should preserve the chains of dependency pairs:
  if $s_1 \to t_1,\ s_2 \to t_2,\ \ldots$ is a chain in $\mathcal{R}$
  then $\pi(s_1) \to \pi(t_1),\ \pi(s_2) \to \pi(t_2),\ \ldots$ should be a chain in $\pi(\mathcal{R})$

- For this purpose, it suffices to consider extra-vars in those functions
  - that are reachable from t^α and
  - occur below a maximal function call of the right-hand side
  - **all other extra-variables can be safely ignored**
    (e.g., replaced by a fresh constructor constant $\perp$)

Removing extra-variables from filtered TRSs

Luckily, some extra-variables can be safely ignored...

- If the argument filtering is safe, extra-variables may only appear *above* the maximal function calls of the right-hand sides (thus $\pi(DP(\mathcal{R}))$ never contains extra-variables)

- As for $\pi(\mathcal{R})$, it should preserve the chains of dependency pairs:
  if $s_1 \to t_1,\ s_2 \to t_2,\ \ldots$ is a chain in $\mathcal{R}$
  then $\pi(s_1) \to \pi(t_1),\ \pi(s_2) \to \pi(t_2),\ \ldots$ should be a chain in $\pi(\mathcal{R})$

- For this purpose, it suffices to consider extra-vars in those functions
  - that are reachable from t^α and
  - occur below a maximal function call of the right-hand side
  - **all other extra-variables can be safely ignored**
    (e.g., replaced by a fresh constructor constant $\perp$)

Example

$t^\alpha = \mathsf{append}(g, v)$

$\pi = \{\mathsf{append} \mapsto \{1\},\ \mathsf{reverse} \mapsto \{1\}\}$ (π is safe for t^α)

The argument filtering processor returns:

Dependency pairs:
$$\left\{ \begin{array}{lcl} \mathsf{APPEND}(\mathsf{cons}(x, xs)) & \to & \mathsf{APPEND}(xs) \\ \mathsf{REVERSE}(\mathsf{cons}(x, xs)) & \to & \mathsf{REVERSE}(xs) \\ \mathsf{REVERSE}(\mathsf{cons}(x, xs)) & \to & \mathsf{APPEND}(\mathsf{reverse}(xs)) \end{array} \right.$$

Rewrite system:
$$\left\{ \begin{array}{rcl} \mathsf{append}(\mathsf{nil}) & \to & \perp \\ \mathsf{append}(\mathsf{cons}(x, xs)) & \to & \mathsf{cons}(x, \mathsf{append}(xs)) \\ \mathsf{reverse}(\mathsf{nil}) & \to & \mathsf{nil} \\ \mathsf{reverse}(\mathsf{cons}(x, xs)) & \to & \mathsf{append}(\mathsf{reverse}(xs)) \end{array} \right.$$

Argument filtering: $id = \{\mathsf{append} \mapsto \{1\},\ \mathsf{reverse} \mapsto \{1\}\}$

A transformational approach

Our aim

- transform the original TRS $\mathcal{R}$ into a new TRS $\mathcal{R}'$
- narrowing terminates in $\mathcal{R}$ if rewriting terminates in $\mathcal{R}'$

Hence any termination technique for rewrite systems can be used to prove the termination of narrowing

Our transformation is a simplification of the *argument filtering transformation* (AFT) of [Kusakari, Nakamura, Toyama 1999]

The transformation $\mathsf{AFT}_\pi(\mathcal{R})$

for every rule $l \to r$ of the original rewrite system, produce

- a filtered rule $\pi(l) \to \pi(r)$ and
- an additional rule $\pi(l) \to \pi(t)$, for each subterm t of r that is filtered away in $\pi(r)$ and such that $\pi(t)$ is not a constructor term.

A transformational approach

Our aim

- transform the original TRS $\mathcal{R}$ into a new TRS $\mathcal{R}'$
- narrowing terminates in $\mathcal{R}$ if rewriting terminates in $\mathcal{R}'$

Hence any termination technique for rewrite systems can be used to prove the termination of narrowing

Our transformation is a simplification of the *argument filtering transformation* (AFT) of [Kusakari, Nakamura, Toyama 1999]

The transformation $\mathsf{AFT}_\pi(\mathcal{R})$

for every rule $l \to r$ of the original rewrite system, produce

- a filtered rule $\pi(l) \to \pi(r)$ and
- an additional rule $\pi(l) \to \pi(t)$, for each subterm t of r that is filtered away in $\pi(r)$ and such that $\pi(t)$ is not a constructor term.

Main result

Theorem

Let π be a safe argument filtering for t^α in $\mathcal{R}$
$\gamma(t^\alpha)$ is $\rightsquigarrow_\mathcal{R}$-terminating if $\mathsf{AFT}_\pi(\mathcal{R})$ is terminating

Therefore,

- $\mathsf{AFT}_\pi(\mathcal{R})$ can be analyzed using standard techniques and tools for proving the termination of TRSs

  (no data generator is involved in the derivations of $\mathsf{AFT}_\pi(\mathcal{R})$)

Example

$$
\begin{aligned}
\mathsf{append}(\mathsf{nil}, y) &\to y \\
\mathsf{append}(\mathsf{cons}(x, xs), y) &\to \mathsf{cons}(x, \mathsf{append}(xs, y)) \\
\mathsf{reverse}(\mathsf{nil}) &\to \mathsf{nil} \\
\mathsf{reverse}(\mathsf{cons}(x, xs)) &\to \mathsf{append}(\mathsf{reverse}(xs), \mathsf{cons}(x, \mathsf{nil}))
\end{aligned}
$$

$t^\alpha = \mathsf{append}(g, v)$

$\pi = \{\mathsf{append} \mapsto \{1\},\ \mathsf{reverse} \mapsto \{1\}\}$ (π is safe for t^α)

The transformation $\mathsf{AFT}_\pi(\mathcal{R})$ returns

$$
\begin{aligned}
\mathsf{append}(\mathsf{nil}) &\to y \qquad\qquad (y \text{ is an extra variable}) \\
\mathsf{append}(\mathsf{cons}(x, xs)) &\to \mathsf{cons}(x, \mathsf{append}(xs)) \\
\mathsf{reverse}(\mathsf{nil}) &\to \mathsf{nil} \\
\mathsf{reverse}(\mathsf{cons}(x, xs)) &\to \mathsf{append}(\mathsf{reverse}(xs))
\end{aligned}
$$

which is clearly **not** terminating

Example

$$
\begin{aligned}
\mathsf{append}(\mathsf{nil}, y) &\to y \\
\mathsf{append}(\mathsf{cons}(x, xs), y) &\to \mathsf{cons}(x, \mathsf{append}(xs, y)) \\
\mathsf{reverse}(\mathsf{nil}) &\to \mathsf{nil} \\
\mathsf{reverse}(\mathsf{cons}(x, xs)) &\to \mathsf{append}(\mathsf{reverse}(xs), \mathsf{cons}(x, \mathsf{nil}))
\end{aligned}
$$

$t^\alpha = \mathsf{append}(g, v)$

$\pi = \{\mathsf{append} \mapsto \{1\},\ \mathsf{reverse} \mapsto \{1\}\}$ (π is safe for t^α)

The transformation $\mathsf{AFT}_\pi(\mathcal{R})$ returns

$$
\begin{aligned}
\mathsf{append}(\mathsf{nil}) &\to \cancel{y}\ \ \bot && (\bot \text{ is a fresh constant}) \\
\mathsf{append}(\mathsf{cons}(x, xs)) &\to \mathsf{cons}(x, \mathsf{append}(xs)) \\
\mathsf{reverse}(\mathsf{nil}) &\to \mathsf{nil} \\
\mathsf{reverse}(\mathsf{cons}(x, xs)) &\to \mathsf{append}(\mathsf{reverse}(xs))
\end{aligned}
$$

which is clearly not terminating

the technique in practice

The termination tool TNT

It takes as input

- a left-linear constructor TRS
- an abstract term

and proceeds as follows:

- infers a safe argument filtering for the abstract term (a binding-time analysis)
- returns a transformed TRS using AFT_π

Website: `http://german.dsic.upv.es/filtering.html`

The termination of the transformed TRS can be checked with APROVE

[DEMO]

Inference of safe argument filterings

We have adapted a simple **binding-time analysis**

- **binding-times**: definitively **g**round / possibly **v**ariable

$$g \sqcup g = g \qquad g \sqcup v = v \qquad v \sqcup g = v \qquad v \sqcup v = v$$

$$(g, v, g) \sqcup (g, g, v) = (g, v, v)$$

$$\{\mathsf{f} \mapsto (g, v),\ \mathsf{g} \mapsto (g, v)\} \sqcup \{\mathsf{f} \mapsto (g, g),\ \mathsf{g} \mapsto (v, g)\} = \{\mathsf{f} \mapsto (g, v),\ \mathsf{g} \mapsto (v, v)\}$$

- **binding-time environment**: a substitution mapping variables to binding-times
- **division**: a mapping $\mathsf{f}/n \mapsto (m_1, \ldots, m_n)$ for every defined function, where each m_i is a binding-time

Auxiliary functions

$$
\begin{array}{lll}
B_v[\![x]\!]\ \mathsf{g}/n\ \rho & = (\overbrace{g, \ldots, g}^{n\ times}) & (\text{if } x \in \mathcal{V}) \\
B_v[\![\mathsf{c}(t_1, \ldots, t_n)]\!]\ \mathsf{g}/n\ \rho & = B_v[\![t_1]\!]\ \mathsf{g}/n\ \rho \sqcup \ldots \sqcup B_v[\![t_n]\!]\ \mathsf{g}/n\ \rho & (\text{if c} \in \mathcal{C}) \\
B_v[\![\mathsf{f}(t_1, \ldots, t_n)]\!]\ \mathsf{g}/n\ \rho & = bt \sqcup (B_e[\![t_1]\!]\ \rho, \ldots, B_e[\![t_n]\!]\ \rho) & (\text{if f} = \text{g},\ \text{f} \in \mathcal{D}) \\
 & bt & (\text{if f} \neq \text{g},\ \text{f} \in \mathcal{D}) \\
 & \multicolumn{2}{l}{\text{where } bt = B_v[\![t_1]\!]\ \mathsf{g}/n\ \rho \sqcup \ldots \sqcup B_v[\![t_n]\!]\ \mathsf{g}/n\ \rho}
\end{array}
$$

$$
\begin{array}{lcll}
B_e[\![x]\!]\ \rho & = & x\rho & (\text{if } x \in \mathcal{V}) \\
B_e[\![\mathsf{h}(t_1, \ldots, t_n)]\!]\ \rho & = & B_e[\![t_1]\!]\ \rho \sqcup \ldots \sqcup B_e[\![t_n]\!]\ \rho & (\text{if h} \in \mathcal{C} \cup \mathcal{D})
\end{array}
$$

Roughly speaking,

- $(B_v[\![t]\!]\ \mathsf{g}/n\ \rho)$ returns a sequence of n binding-times that denote the (lub of the) binding-times of the arguments of the calls to g/n that occur in t in the context of the binding-time environment ρ
- $(B_e[\![t]\!]\ \rho)$ then returns g if t contains no variable which is bound to v in ρ, and v otherwise

Auxiliary functions

$$
\begin{array}{lll}
B_v[[x]]\ \mathsf{g}/n\ \rho & = (\overbrace{g,\ldots,g}^{n\ times}) & (\text{if } x \in \mathcal{V}) \\
B_v[[\mathsf{c}(t_1,\ldots,t_n)]]\ \mathsf{g}/n\ \rho & = B_v[[t_1]]\ \mathsf{g}/n\ \rho \sqcup \ldots \sqcup B_v[[t_n]]\ \mathsf{g}/n\ \rho & (\text{if } \mathsf{c} \in \mathcal{C}) \\
B_v[[\mathsf{f}(t_1,\ldots,t_n)]]\ \mathsf{g}/n\ \rho & = bt \sqcup (B_e[[t_1]]\ \rho,\ldots,B_e[[t_n]]\ \rho) & (\text{if } \mathsf{f} = \mathsf{g},\ \mathsf{f} \in \mathcal{D}) \\
 & \ bt & (\text{if } \mathsf{f} \neq \mathsf{g},\ \mathsf{f} \in \mathcal{D}) \\
 & \quad \text{where } bt = B_v[[t_1]]\ \mathsf{g}/n\ \rho \sqcup \ldots \sqcup B_v[[t_n]]\ \mathsf{g}/n\ \rho &
\end{array}
$$

$$
\begin{array}{lll}
B_e[[x]]\ \rho & = x\rho & (\text{if } x \in \mathcal{V}) \\
B_e[[\mathsf{h}(t_1,\ldots,t_n)]]\ \rho & = B_e[[t_1]]\ \rho \sqcup \ldots \sqcup B_e[[t_n]]\ \rho & (\text{if } \mathsf{h} \in \mathcal{C} \cup \mathcal{D})
\end{array}
$$

Roughly speaking,

- $(B_v[[t]]\ \mathsf{g}/n\ \rho)$ returns a sequence of n binding-times that denote the (lub of the) binding-times of the arguments of the calls to g/n that occur in t in the context of the binding-time environment ρ
- $(B_e[[t]]\ \rho)$ then returns g if t contains no variable which is bound to v in ρ, and v otherwise

Auxiliary functions

$$
\begin{array}{lcl}
B_v[[x]]\ \mathsf{g}/n\ \rho & = & (\overbrace{g,\ldots,g}^{n\ times}) \quad (\text{if } x \in \mathcal{V})\\
B_v[[\mathsf{c}(t_1,\ldots,t_n)]]\ \mathsf{g}/n\ \rho & = & B_v[[t_1]]\ \mathsf{g}/n\ \rho \sqcup \ldots \sqcup B_v[[t_n]]\ \mathsf{g}/n\ \rho \quad (\text{if } \mathsf{c} \in \mathcal{C})\\
B_v[[\mathsf{f}(t_1,\ldots,t_n)]]\ \mathsf{g}/n\ \rho & = & bt \sqcup (B_e[[t_1]]\ \rho,\ldots,B_e[[t_n]]\ \rho) \quad (\text{if } \mathsf{f} = \mathsf{g},\ \mathsf{f} \in \mathcal{D})\\
 & & bt \quad (\text{if } \mathsf{f} \neq \mathsf{g},\ \mathsf{f} \in \mathcal{D})\\
 & & \text{where } bt = B_v[[t_1]]\ \mathsf{g}/n\ \rho \sqcup \ldots \sqcup B_v[[t_n]]\ \mathsf{g}/n\ \rho
\end{array}
$$

$$
\begin{array}{lcl}
B_e[[x]]\ \rho & = & x\rho \quad (\text{if } x \in \mathcal{V})\\
B_e[[\mathsf{h}(t_1,\ldots,t_n)]]\ \rho & = & B_e[[t_1]]\ \rho \sqcup \ldots \sqcup B_e[[t_n]]\ \rho \quad (\text{if } \mathsf{h} \in \mathcal{C} \cup \mathcal{D})
\end{array}
$$

Roughly speaking,

- $(B_v[[t]]\ \mathsf{g}/n\ \rho)$ returns a sequence of n binding-times that denote the (lub of the) binding-times of the arguments of the calls to g/n that occur in t in the context of the binding-time environment ρ
- $(B_e[[t]]\ \rho)$ then returns g if t contains no variable which is bound to v in ρ, and v otherwise

BTA algorithm

Given an abstract term $\mathsf{f}_1(m_1, \ldots, m_{n_1})$, the initial division is

$$div_0 = \{\mathsf{f}_1 \mapsto (m_1, \ldots, m_{n_1}),\ \mathsf{f}_2 \mapsto (g, \ldots, g),\ \ \ldots,\ \mathsf{f}_k \mapsto (g, \ldots, g)\}$$

where $\mathsf{f}_1/n_1, \ldots, \mathsf{f}_k/n_k$ are the defined functions of the TRS

Iterative process

$$div_i = \{\mathsf{f}_1 \mapsto b_1,\ \ldots,\ \mathsf{f}_k \mapsto b_k\}$$
$$\Downarrow$$
$$\begin{array}{ll} div_{i+1} = \{ & \mathsf{f}_1 \mapsto b_1 \sqcup B_v[[r_1]]\ \mathsf{f}_1/n_1\ e(b_1, l_1) \sqcup \ldots \sqcup B_v[[r_j]]\ \mathsf{f}_1/n_1\ e(b_j, l_j), \\ & \ldots, \\ & \mathsf{f}_k \mapsto b_k \sqcup B_v[[r_1]]\ \mathsf{f}_k/n_k\ e(b_1, l_1) \sqcup \ldots \sqcup B_v[[r_j]]\ \mathsf{f}_k/n_k\ e(b_j, l_j)\ \} \end{array}$$

where $l_1 \to r_1, \ldots, l_j \to r_j$, $j \geq k$, are the rules of the TRS

$$\begin{array}{rcl} e((m_1, \ldots, m_n), \mathsf{f}(t_1, \ldots, t_n)) & = & \{x \mapsto m_1 \mid x \in \mathcal{V}\mathsf{ar}(t_1)\} \\ & \cup & \ldots \\ & \cup & \{x \mapsto m_n \mid x \in \mathcal{V}\mathsf{ar}(t_n)\} \end{array}$$

BTA algorithm

Given an abstract term $\mathsf{f}_1(m_1,\ldots,m_{n_1})$, the initial division is

$$div_0 = \{\mathsf{f}_1 \mapsto (m_1,\ldots,m_{n_1}),\ \mathsf{f}_2 \mapsto (g,\ldots,g),\ \ \ldots,\ \mathsf{f}_k \mapsto (g,\ldots,g)\}$$

where $\mathsf{f}_1/n_1,\ldots,\mathsf{f}_k/n_k$ are the defined functions of the TRS

Iterative process

$$div_i = \{\mathsf{f}_1 \mapsto b_1,\ \ldots,\ \mathsf{f}_k \mapsto b_k\}$$
$$\Downarrow$$
$$\begin{array}{rl} div_{i+1} = \{ & \mathsf{f}_1 \mapsto b_1 \sqcup B_v[[r_1]]\ \mathsf{f}_1/n_1\ e(b_1,l_1) \sqcup \ldots \sqcup B_v[[r_j]]\ \mathsf{f}_1/n_1\ e(b_j,l_j), \\ & \ldots, \\ & \mathsf{f}_k \mapsto b_k \sqcup B_v[[r_1]]\ \mathsf{f}_k/n_k\ e(b_1,l_1) \sqcup \ldots \sqcup B_v[[r_j]]\ \mathsf{f}_k/n_k\ e(b_j,l_j)\ \} \end{array}$$

where $l_1 \to r_1,\ldots,l_j \to r_j$, $j \geq k$, are the rules of the TRS

$$\begin{array}{rcl} e((m_1,\ldots,m_n),\mathsf{f}(t_1,\ldots,t_n)) & = & \{x \mapsto m_1 \mid x \in \mathcal{V}\mathrm{ar}(t_1)\} \\ & \cup & \ldots \\ & \cup & \{x \mapsto m_n \mid x \in \mathcal{V}\mathrm{ar}(t_n)\} \end{array}$$

When $div_i = div_{i+1}$ (fixpoint), the corresponding safe argument filtering π is obtained as follows:

Given the division

$$\boxed{div = \{\mathsf{f}_1 \mapsto (m_1^1, \ldots, m_{n_1}^1), \ldots, \mathsf{f}_k \mapsto (m_1^k, \ldots, m_{n_k}^k)\}}$$

we have

$$\boxed{\pi(div) = \{\mathsf{f}_1 \mapsto \{i \mid m_i^1 = g\}, \ldots, \mathsf{f}_k \mapsto \{i \mid m_i^k = g\}\}}$$

$\pi(div)$ is a safe argument filtering since the computed division *div* is *congruent* [JGS93]

Example

$$
\begin{array}{rcl}
\mathsf{mult}(\mathsf{z}, y) & \to & \mathsf{z} \\
\mathsf{mult}(\mathsf{s}(x), y) & \to & \mathsf{add}(\mathsf{mult}(x, y), y)
\end{array}
\qquad
\begin{array}{rcl}
\mathsf{add}(\mathsf{z}, y) & \to & y \\
\mathsf{add}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{add}(x, y))
\end{array}
$$

Given the abstract term $\mathsf{mult}(g, v)$, the associated initial division is

$$div_0 = \{\mathsf{mult} \mapsto (g, v),\ \mathsf{add} \mapsto (g, g)\}$$

The next division, div_1, is obtained from the following expression:

$$
\begin{array}{rcl}
div_1 = \{\mathsf{mult} \mapsto (g, v) & \sqcup & B_v[[\mathsf{z}]]\ \mathsf{mult}/2\ \{y \mapsto v\} \\
 & \sqcup & B_v[[\mathsf{add}(\mathsf{mult}(x, y), y)]]\ \mathsf{mult}/2\ \{x \mapsto g,\ y \mapsto v\} \\
 & \sqcup & B_v[[y]]\ \mathsf{mult}/2\ \{y \mapsto g\} \\
 & \sqcup & B_v[[\mathsf{s}(\mathsf{add}(x, y))]]\ \mathsf{mult}/2\ \{x \mapsto g,\ y \mapsto g\}, \\
\mathsf{add} \mapsto (g, g) & \sqcup & B_v[[\mathsf{z}]]\ \mathsf{add}/2\ \{y \mapsto v\} \\
 & \sqcup & B_v[[\mathsf{add}(\mathsf{mult}(x, y), y)]]\ \mathsf{add}/2\ \{x \mapsto g,\ y \mapsto v\} \\
 & \sqcup & B_v[[y]]\ \mathsf{add}/2\ \{y \mapsto g\} \\
 & \sqcup & B_v[[\mathsf{s}(\mathsf{add}(x, y))]]\ \mathsf{add}/2\ \{x \mapsto g,\ y \mapsto g\} \quad \}
\end{array}
$$

Example

$$\begin{array}{rcl} \mathsf{mult}(\mathsf{z}, y) & \to & \mathsf{z} \\ \mathsf{mult}(\mathsf{s}(x), y) & \to & \mathsf{add}(\mathsf{mult}(x, y), y) \end{array} \qquad \begin{array}{rcl} \mathsf{add}(\mathsf{z}, y) & \to & y \\ \mathsf{add}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{add}(x, y)) \end{array}$$

Given the abstract term $\mathsf{mult}(g, v)$, the associated initial division is

$$div_0 = \{\mathsf{mult} \mapsto (g, v),\ \mathsf{add} \mapsto (g, g)\}$$

The next division, div_1, is obtained from the following expression:

$$\begin{array}{rll} div_1 = \{\mathsf{mult} \mapsto (g, v) & \sqcup & B_v[[\mathsf{z}]]\ \mathsf{mult}/2\ \{y \mapsto v\} \\ & \sqcup & B_v[[\mathsf{add}(\mathsf{mult}(x, y), y)]]\ \mathsf{mult}/2\ \{x \mapsto g,\ y \mapsto v\} \\ & \sqcup & B_v[[y]]\ \mathsf{mult}/2\ \{y \mapsto g\} \\ & \sqcup & B_v[[\mathsf{s}(\mathsf{add}(x, y))]]\ \mathsf{mult}/2\ \{x \mapsto g,\ y \mapsto g\}, \\ \mathsf{add} \mapsto (g, g) & \sqcup & B_v[[\mathsf{z}]]\ \mathsf{add}/2\ \{y \mapsto v\} \\ & \sqcup & B_v[[\mathsf{add}(\mathsf{mult}(x, y), y)]]\ \mathsf{add}/2\ \{x \mapsto g,\ y \mapsto v\} \\ & \sqcup & B_v[[y]]\ \mathsf{add}/2\ \{y \mapsto g\} \\ & \sqcup & B_v[[\mathsf{s}(\mathsf{add}(x, y))]]\ \mathsf{add}/2\ \{x \mapsto g,\ y \mapsto g\} \quad \} \end{array}$$

Example (cont'd)

Therefore, by evaluating the calls to B_v, we get

$$\boxed{div_1 = \{\text{mult} \mapsto (g, v),\ \text{add} \mapsto (v, v)\}}$$

Note that the change in the binding-times of add comes from the evaluation of

$$B_v[[\text{add}(\text{mult}(x, y), y)]]\ \text{add}/2\ \{x \mapsto g,\ y \mapsto v\}$$

where a call to add appears
(and every argument contains at least one possibly unknown value)

$\Rightarrow$ **If we compute** div_2 **we get** $div_1 = div_2 \Longrightarrow$ $\boxed{div_1 \text{ is a fixpoint}}$

From this division, the associated safe argument filtering is

$$\boxed{\pi = \{\ \text{mult} \mapsto \{1\},\ \text{add} \mapsto \{\ \}\ \}}$$

Example (cont'd)

Therefore, by evaluating the calls to B_v, we get

$$\boxed{div_1 = \{\mathsf{mult} \mapsto (g, v),\ \mathsf{add} \mapsto (v, v)\}}$$

Note that the change in the binding-times of add comes from the evaluation of

$$B_v[[\mathsf{add}(\mathsf{mult}(x, y), y)]]\ \mathsf{add}/2\ \{x \mapsto g,\ y \mapsto v\}$$

where a call to add appears
(and every argument contains at least one possibly unknown value)

$\Rightarrow$ **If we compute** div_2 **we get** $div_1 = div_2 \Longrightarrow$ $\boxed{div_1 \textbf{ is a fixpoint}}$

From this division, the associated safe argument filtering is

$$\boxed{\pi = \{\ \mathsf{mult} \mapsto \{1\},\ \mathsf{add} \mapsto \{\ \}\ \}}$$

Example (cont'd)

Therefore, by evaluating the calls to B_v, we get

$$\boxed{div_1 = \{\mathsf{mult} \mapsto (g, v),\ \mathsf{add} \mapsto (v, v)\}}$$

Note that the change in the binding-times of add comes from the evaluation of

$$B_v[\![\mathsf{add}(\mathsf{mult}(x, y), y)]\!]\ \mathsf{add}/2\ \{x \mapsto g,\ y \mapsto v\}$$

where a call to add appears
(and every argument contains at least one possibly unknown value)

$\Rightarrow$ **If we compute** div_2 **we get** $div_1 = div_2 \Longrightarrow$ $\boxed{div_1 \textbf{ is a fixpoint}}$

From this division, the associated safe argument filtering is

$$\boxed{\pi = \{\ \mathsf{mult} \mapsto \{1\},\ \mathsf{add} \mapsto \{\ \}\ \}}$$

Some refinements

Multiple abstract terms

Consider, e.g.,

$$\begin{array}{rcl} \mathsf{eq}(\mathsf{z},\mathsf{z}) & \rightarrow & \mathsf{true} \\ \mathsf{eq}(\mathsf{s}(x),\mathsf{s}(y)) & \rightarrow & \mathsf{eq}(x,y) \end{array}$$

and the set

$$T^\alpha = \{\mathsf{eq}(g,v),\ \mathsf{eq}(v,g)\}$$

Here, starting from

$$div_0 = \{\ \mathsf{eq} \mapsto (g,v) \sqcup (v,g)\ \} = \{\ \mathsf{eq} \mapsto (v,v)\ \}$$

is not a good idea . . .

Solution

Lemma

Let $\mathcal{R}$ be a TRS and T^α be a finite set of abstract terms. $\gamma(T^\alpha)$ is $\leadsto_\mathcal{R}$-terminating iff $\gamma(t^\alpha)$ is $\leadsto_\mathcal{R}$-terminating for all $t^\alpha \in T^\alpha$.

Some refinements

Multiple abstract terms

Consider, e.g.,

$$\begin{array}{rcl} \mathsf{eq}(\mathsf{z},\mathsf{z}) & \rightarrow & \mathsf{true} \\ \mathsf{eq}(\mathsf{s}(x),\mathsf{s}(y)) & \rightarrow & \mathsf{eq}(x,y) \end{array}$$

and the set

$$T^\alpha = \{\mathsf{eq}(g,v),\ \mathsf{eq}(v,g)\}$$

Here, starting from

$$div_0 = \{\ \mathsf{eq} \mapsto (g,v) \sqcup (v,g)\ \} = \{\ \mathsf{eq} \mapsto (v,v)\ \}$$

is not a good idea . . .

Solution

Lemma

Let $\mathcal{R}$ be a TRS and T^α be a finite set of abstract terms. $\gamma(T^\alpha)$ is $\leadsto_\mathcal{R}$-terminating iff $\gamma(t^\alpha)$ is $\leadsto_\mathcal{R}$-terminating for all $t^\alpha \in T^\alpha$.

Some refinements (cont'd)

Non well-moded programs

Consider, e.g.,

$$\begin{array}{rcl} \mathsf{eq}(\mathsf{z},\mathsf{z}) & \rightarrow & \mathsf{true} \\ \mathsf{eq}(\mathsf{s}(x),\mathsf{s}(y)) & \rightarrow & \mathsf{eq}(y,x) \end{array}$$

If we start with

$$\mathsf{eq}(g, v)$$

the only safe argument filtering is

$$\pi = \{\mathsf{eq} \mapsto \{\,\}\}$$

Solution

$$\begin{array}{rcl} \mathsf{eq}_{gg}(\mathsf{z},\mathsf{z}) & \rightarrow & \mathsf{true} \\ \mathsf{eq}_{gg}(\mathsf{s}(x),\mathsf{s}(y)) & \rightarrow & \mathsf{eq}_{gg}(y,x) \\ \mathsf{eq}_{vg}(\mathsf{z},\mathsf{z}) & \rightarrow & \mathsf{true} \\ \mathsf{eq}_{vg}(\mathsf{s}(x),\mathsf{s}(y)) & \rightarrow & \mathsf{eq}_{gv}(y,x) \end{array}$$

$$\begin{array}{rcl} \mathsf{eq}_{gv}(\mathsf{z},\mathsf{z}) & \rightarrow & \mathsf{true} \\ \mathsf{eq}_{gv}(\mathsf{s}(x),\mathsf{s}(y)) & \rightarrow & \mathsf{eq}_{vg}(y,x) \\ \mathsf{eq}_{vv}(\mathsf{z},\mathsf{z}) & \rightarrow & \mathsf{true} \\ \mathsf{eq}_{vv}(\mathsf{s}(x),\mathsf{s}(y)) & \rightarrow & \mathsf{eq}_{vv}(y,x) \end{array}$$

Some refinements (cont'd)

Non well-moded programs

Consider, e.g.,

$$\begin{array}{rcl} \mathsf{eq}(\mathsf{z},\mathsf{z}) & \to & \mathsf{true} \\ \mathsf{eq}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{eq}(y,x) \end{array}$$

If we start with

$$\mathsf{eq}(g,v)$$

the only safe argument filtering is

$$\pi = \{\mathsf{eq} \mapsto \{\,\}\}$$

Solution

$$\begin{array}{rcl} \mathsf{eq}_{gg}(\mathsf{z},\mathsf{z}) & \to & \mathsf{true} \\ \mathsf{eq}_{gg}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{eq}_{gg}(y,x) \\ \mathsf{eq}_{vg}(\mathsf{z},\mathsf{z}) & \to & \mathsf{true} \\ \mathsf{eq}_{vg}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{eq}_{gv}(y,x) \end{array} \qquad \begin{array}{rcl} \mathsf{eq}_{gv}(\mathsf{z},\mathsf{z}) & \to & \mathsf{true} \\ \mathsf{eq}_{gv}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{eq}_{vg}(y,x) \\ \mathsf{eq}_{vv}(\mathsf{z},\mathsf{z}) & \to & \mathsf{true} \\ \mathsf{eq}_{vv}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{eq}_{vv}(y,x) \end{array}$$

Some refinements (cont'd)

Removing non-reachable functions

Consider, e.g.,

$$\begin{array}{lcl} \mathsf{a} & \rightarrow & \mathsf{a} \\ \mathsf{b} & \rightarrow & \mathsf{c} \\ \mathsf{c} & \rightarrow & \mathsf{d} \end{array}$$

Although narrowing terminates for the abstract term b we get the argument filtering

$$\pi = \{\mathsf{a} \mapsto \{\,\},\ \mathsf{b} \mapsto \{\,\},\ \mathsf{c} \mapsto \{\,\}\}$$

and then we fail to prove its termination...

Solution

Remove function definitions not reachable from b (i.e., a $\rightarrow$ a)

Some refinements (cont'd)

Removing non-reachable functions

Consider, e.g.,

$$\begin{array}{lcl} a & \to & a \\ b & \to & c \\ c & \to & d \end{array}$$

Although narrowing terminates for the abstract term b we get the argument filtering

$$\pi = \{a \mapsto \{\,\},\ b \mapsto \{\,\},\ c \mapsto \{\,\}\}$$

and then we fail to prove its termination...

Solution

Remove function definitions not reachable from b (i.e., $a \to a$)

related work and conclusions

Related work

Schneider-Kamp *et al* [SKGST07] presented an automated termination analysis for logic programs:

- logic programs are first translated to TRSs
- logic variables are simulated by infinite terms

Main differences:

- data generators (reuse of results relating narrowing and rewriting)
- no transformational approach in [SKGST07]

Nishida et al [NSS03, NM06] adapted the dependency pair method for proving the termination of narrowing:

- direct approach (not based on using generators & rewriting)
- allow extra variables in TRSs and do not consider initial terms
- do not remove some (unnecessary) extra-variables (as we do)

Alpuente, Escobar, and Iborra [AEI08]

- extend the use of dependency pairs to narrowing over arbitrary TRSs

Related work

Schneider-Kamp *et al* [SKGST07] presented an automated termination analysis for logic programs:

- logic programs are first translated to TRSs
- logic variables are simulated by infinite terms

Main differences:

- data generators (reuse of results relating narrowing and rewriting)
- no transformational approach in [SKGST07]

Nishida et al [NSS03, NM06] adapted the dependency pair method for proving the termination of narrowing:

- direct approach (not based on using generators & rewriting)
- allow extra variables in TRSs and do not consider initial terms
- do not remove some (unnecessary) extra-variables (as we do)

Alpuente, Escobar, and Iborra [AEI08]

- extend the use of dependency pairs to narrowing over arbitrary TRSs

Related work

Schneider-Kamp *et al* [SKGST07] presented an automated termination analysis for logic programs:

- logic programs are first translated to TRSs
- logic variables are simulated by infinite terms

Main differences:

- data generators (reuse of results relating narrowing and rewriting)
- no transformational approach in [SKGST07]

Nishida et al [NSS03, NM06] adapted the dependency pair method for proving the termination of narrowing:

- direct approach (not based on using generators & rewriting)
- allow extra variables in TRSs and do not consider initial terms
- do not remove some (unnecessary) extra-variables (as we do)

Alpuente, Escobar, and Iborra [AEI08]

- extend the use of dependency pairs to narrowing over arbitrary TRSs

Conclusions

Conclusions

- new techniques for proving the termination of narrowing in left-linear constructor systems
- good potential for reusing existing techniques and tools for rewriting
- first tool for proving the termination of narrowing

Future work

- extension to deal with extra-variables
- improve accuracy
- consider strategies (e.g., termination of lazy narrowing)

Conclusions

Conclusions

- new techniques for proving the termination of narrowing in left-linear constructor systems
- good potential for reusing existing techniques and tools for rewriting
- first tool for proving the termination of narrowing

Future work

- extension to deal with extra-variables
- improve accuracy
- consider strategies (e.g., termination of lazy narrowing)

M. Alpuente, S. Escobar, and J. Iborra.
Termination of narrowing using dependency pairs.
In Maria Garcia de la Banda and Enrico Pontelli, editors, *Proc. of the 24th International Conference on Logic Programming (ICLP 2008)*, pages 317–331. Springer LNCS 5366, 2008.

J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke.
Mechanizing and Improving Dependency Pairs.
Journal of Automated Reasoning, 37(3):155–203, 2006.

N.D. Jones, C.K. Gomard, and P. Sestoft.
Partial Evaluation and Automatic Program Generation.
Prentice-Hall, Englewood Cliffs, NJ, 1993.

N. Nishida and K. Miura.
Dependency Graph Method for Proving Termination of Narrowing.
In *Proc. of WST'06*, pages 12–16, 2006.

N. Nishida, M. Sakai, and T. Sakabe.

Narrowing-Based Simulation of Term Rewriting Systems with Extra Variables.
ENTCS, 86(3), 2003.

P. Schneider-Kamp, J. Giesl, A. Serebrenik, and R. Thiemann.
Automated Termination Analysis for Logic Programs by Term Rewriting.
In *Proc. of LOPSTR'06*, pages 177–193. Springer LNCS 4407, 2007.