

Termination of Narrowing in Left-Linear Constructor Systems

Germán Vidal
Technical University of Valencia, Spain

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

MÁSTER EN TECNOLOGÍAS INFORMÁTICAS AVANZADAS

MAY 20, 2008
ALBACETE, 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)

$$\begin{array}{l} \text{add}(z, y) \rightarrow y \quad (R_1) \\ \text{add}(s(x), y) \rightarrow s(\text{add}(x, y)) \quad (R_2) \end{array}$$

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

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

but also:

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

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

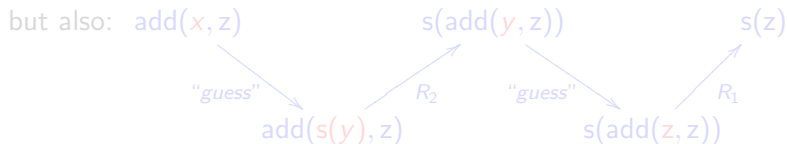
What is narrowing?

Standard definition
of addition (TRS)

$$\begin{array}{lcl} \text{add}(z, y) & \rightarrow & y & (R_1) \\ \text{add}(s(x), y) & \rightarrow & s(\text{add}(x, y)) & (R_2) \end{array}$$

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

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



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

What is narrowing?

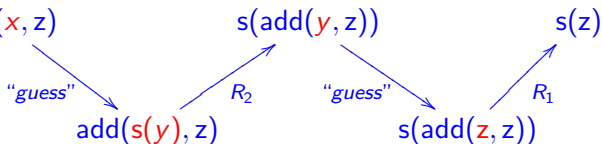
Standard definition
of addition (TRS)

$$\begin{array}{l} \text{add}(z, y) \rightarrow y \quad (R_1) \\ \text{add}(s(x), y) \rightarrow s(\text{add}(x, y)) \quad (R_2) \end{array}$$

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

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

but also: $\text{add}(x, z)$



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

What is narrowing?

Standard definition
of addition (TRS)

$$\begin{array}{lcl} \text{add}(z, y) & \rightarrow & y & (R_1) \\ \text{add}(s(x), y) & \rightarrow & s(\text{add}(x, y)) & (R_2) \end{array}$$

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

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

but also: $\text{add}(x, z) \rightsquigarrow_{R_2, \{x \mapsto s(y)\}} s(\text{add}(y, z)) \rightsquigarrow_{\{R_1, y \mapsto z\}} s(z)$

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

Formal definition

Definition (rewriting)

$s \rightarrow_{p,R} s[r\sigma]_p$ if there are

- a position p of s
- a rule $R = (l \rightarrow r)$ in \mathcal{R}
- a substitution σ such that $s|_p = l\sigma$

⇓

Definition (narrowing)

$s \rightsquigarrow_{p,R,\sigma} (s[r]_p)\sigma$ if there are

- a **nonvariable** position p of s
- a **variant** $R = (l \rightarrow r)$ of a rule in \mathcal{R}
- a substitution σ such that $s|_p\sigma = l\sigma$
 $[\sigma = \text{mgu}(s|_p, l)]$

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!

$$\text{add}(x, y) \rightsquigarrow_{R_2, \{x \mapsto s(x')\}} \text{add}(x', y) \rightsquigarrow_{R_2, \{x' \mapsto s(x'')\}} \dots$$

In this work

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

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

Termination of narrowing

The termination problem

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

Too strong!

$$\text{add}(x, y) \rightsquigarrow_{R_2, \{x \mapsto s(x')\}} \text{add}(x', y) \rightsquigarrow_{R_2, \{x' \mapsto s(x'')\}} \dots$$

In this work

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

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

Termination of narrowing

The termination problem

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

Too strong!

$$\text{add}(x, y) \rightsquigarrow_{R_2, \{x \mapsto s(x')\}} \text{add}(x', y) \rightsquigarrow_{R_2, \{x' \mapsto s(x'')\}} \dots$$

In this work

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

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

Termination of narrowing via termination of rewriting

Theorem

T is $\rightsquigarrow_{\mathcal{R}}$ -terminating

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

Drawbacks:

- very difficult to approximate
- sufficient but not necessary:

$$\begin{array}{l} f(a) \rightarrow b \\ a \rightarrow a \end{array}$$

The set $\{f(x)\}$ is $\rightsquigarrow_{\mathcal{R}}$ -terminating

however $\{f(a)\}$ is finite but not $\rightarrow_{\mathcal{R}}$ -terminating:

$$f(a) \rightarrow f(a) \rightarrow f(a) \rightarrow \dots$$

Termination of narrowing via termination of rewriting

Theorem

T is $\rightsquigarrow_{\mathcal{R}}$ -terminating

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

Drawbacks:

- very difficult to approximate
- sufficient but not necessary:

$$\begin{array}{lcl} f(a) & \rightarrow & b \\ a & \rightarrow & a \end{array}$$

The set $\{f(x)\}$ is $\rightsquigarrow_{\mathcal{R}}$ -terminating

however $\{f(a)\}$ is finite but not $\rightarrow_{\mathcal{R}}$ -terminating:

$$f(a) \rightarrow f(a) \rightarrow f(a) \rightarrow \dots$$

Termination of narrowing via termination of rewriting

Theorem

T is $\rightsquigarrow_{\mathcal{R}}$ -terminating

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

Drawbacks:

- very difficult to approximate
- sufficient but not necessary:

$$\begin{array}{l} f(a) \rightarrow b \\ a \rightarrow a \end{array}$$

The set $\{f(x)\}$ is $\rightsquigarrow_{\mathcal{R}}$ -terminating

however $\{f(a)\}$ is finite but not $\rightarrow_{\mathcal{R}}$ -terminating:

$$f(a) \rightarrow f(a) \rightarrow f(a) \rightarrow \dots$$

A first solution

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

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

\Downarrow

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

Why infinite terms?

Example

$$\begin{array}{ll} \text{add}(z, y) & \rightarrow y & (R_1) \\ \text{add}(s(x), y) & \rightarrow s(\text{add}(x, y)) & (R_2) \end{array}$$

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

$$\text{add}(s(s(\dots)), z) \rightarrow_{\mathcal{R}} s(\text{add}(s(s(\dots)), z)) \rightarrow_{\mathcal{R}} \dots$$

A first solution

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

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

\Downarrow

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

Why infinite terms?

Example

$$\begin{array}{l} \text{add}(z, y) \rightarrow y \quad (R_1) \\ \text{add}(s(x), y) \rightarrow s(\text{add}(x, y)) \quad (R_2) \end{array}$$

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

$$\text{add}(s(s(\dots)), z) \rightarrow_{\mathcal{R}} s(\text{add}(s(s(\dots)), z)) \rightarrow_{\mathcal{R}} \dots$$

A first solution

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

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

\Downarrow

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

Why infinite terms?

Example

$$\begin{array}{ll} \text{add}(z, y) & \rightarrow y & (R_1) \\ \text{add}(s(x), y) & \rightarrow s(\text{add}(x, y)) & (R_2) \end{array}$$

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

$$\text{add}(s(s(\dots)), z) \rightarrow_{\mathcal{R}} s(\text{add}(s(s(\dots)), z)) \rightarrow_{\mathcal{R}} \dots$$

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

$$\begin{array}{l} a \rightarrow a \\ f(x) \rightarrow x \end{array}$$

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

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

\Rightarrow an infinite computation $f(a) \rightarrow_{\mathcal{R}} f(a) \rightarrow_{\mathcal{R}} \dots$ 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

$$\begin{array}{l} a \rightarrow a \\ f(x) \rightarrow x \end{array}$$

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

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

\Rightarrow an infinite computation $f(a) \rightarrow_{\mathcal{R}} f(a) \rightarrow_{\mathcal{R}} \dots$ 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

$$\begin{array}{l} a \rightarrow a \\ f(x) \rightarrow x \end{array}$$

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

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

\Rightarrow an infinite computation $f(a) \rightarrow_{\mathcal{R}} f(a) \rightarrow_{\mathcal{R}} \dots$ 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

$$\begin{array}{l} a \rightarrow a \\ f(a) \rightarrow c(b, b) \end{array}$$

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

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

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

$$\begin{array}{l} a \rightarrow a \\ f(a) \rightarrow c(b, b) \end{array}$$

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

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

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

$$\begin{array}{l} a \rightarrow a \\ f(a) \rightarrow c(b, b) \end{array}$$

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

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

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

$$\begin{array}{l} a \rightarrow a \\ f(a) \rightarrow c(b, b) \end{array}$$

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

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

Last (good) solution

\Rightarrow $\left\{ \begin{array}{l} \text{restrict to narrowing derivations} \\ \text{where terms introduced by instantiation cannot be narrowed!} \end{array} \right.$

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 \Rightarrow

Termination of narrowing via termination of rewriting

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

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

with

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

Property variables are bound to (irreducible) constructor terms



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}_{gen} as \mathcal{R} augmented with

$$\text{gen} \rightarrow c(\overbrace{\text{gen}, \dots, \text{gen}}^{n \text{ times}})$$

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

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

$$\mathcal{R}_{\text{gen}} = \mathcal{R} \cup \left\{ \begin{array}{l} \text{gen} \rightarrow z \\ \text{gen} \rightarrow s(\text{gen}) \end{array} \right\}$$

Some notation: $\hat{t} = t\sigma$, with $\sigma = \{x \mapsto \text{gen} \mid x \in \text{Var}(t)\}$

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

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

$$\text{gen} \rightarrow c(\overbrace{\text{gen}, \dots, \text{gen}}^{n \text{ times}})$$

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

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

$$\mathcal{R}_{\text{gen}} = \mathcal{R} \cup \left\{ \begin{array}{l} \text{gen} \rightarrow z \\ \text{gen} \rightarrow s(\text{gen}) \end{array} \right\}$$

Some notation: $\hat{t} = t\sigma$, with $\sigma = \{x \mapsto \text{gen} \mid x \in \text{Var}(t)\}$

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

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

$$\text{gen} \rightarrow c(\overbrace{\text{gen}, \dots, \text{gen}}^{n \text{ times}})$$

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

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

$$\mathcal{R}_{\text{gen}} = \mathcal{R} \cup \left\{ \begin{array}{l} \text{gen} \rightarrow z \\ \text{gen} \rightarrow s(\text{gen}) \end{array} \right\}$$

Some notation: $\hat{t} = t\sigma$, with $\sigma = \{x \mapsto \text{gen} \mid x \in \text{Var}(t)\}$

Correctness of data generators [Antoy, Hanus 2006]

Completeness

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

Generally unsound

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

but $\text{add}(x, x) \rightsquigarrow_{\{x \mapsto z\}} z$
 $\text{add}(x, x) \rightsquigarrow_{\{x \mapsto s(x')\}} s(\text{add}(x', s(x'))) \rightsquigarrow_{\{x' \mapsto z\}} s(s(z))$
 ...

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 [Antoy, Hanus 2006]

Completeness

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

Generally unsound

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

but $\text{add}(x, x) \rightsquigarrow_{\{x \mapsto z\}} z$
 $\text{add}(x, x) \rightsquigarrow_{\{x \mapsto \text{s}(x')\}} \text{s}(\text{add}(x', \text{s}(x')))) \rightsquigarrow_{\{x' \mapsto z\}} \text{s}(\text{s}(z))$
 ...

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 [Antoy, Hanus 2006]

Completeness

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

Generally unsound

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

but $\text{add}(x, x) \rightsquigarrow_{\{x \mapsto z\}} z$
 $\text{add}(x, x) \rightsquigarrow_{\{x \mapsto \text{s}(x')\}} \text{s}(\text{add}(x', \text{s}(x')))) \rightsquigarrow_{\{x' \mapsto z\}} \text{s}(\text{s}(z))$
 ...

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}_{gen} ?

Clearly, no term with occurrences of **gen** terminates!

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

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

Theorem (termination of narrowing via termination of rewriting)

Let \mathcal{R} be a left-linear constructor TRS

T is $\rightsquigarrow_{\mathcal{R}}$ -terminating

iff

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

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

Clearly, no term with occurrences of **gen** terminates!

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

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

Theorem (termination of narrowing via termination of rewriting)

Let \mathcal{R} be a left-linear constructor TRS

T is $\rightsquigarrow_{\mathcal{R}}$ -terminating

iff

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

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

Clearly, no term with occurrences of **gen** terminates!

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

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

Theorem (termination of narrowing via termination of rewriting)

Let \mathcal{R} be a left-linear constructor TRS

T is $\rightsquigarrow_{\mathcal{R}}$ -terminating

iff

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

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., $\text{add}(g, v)$ denotes the set of terms $\text{add}(t_1, t_2)$ with
 - t_1 (definitely) **g**round
 - t_2 (possibly) **v**ariable
- concretization function γ ,

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

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., $\text{add}(g, v)$ denotes the set of terms $\text{add}(t_1, t_2)$ with
 - t_1 (definitely) **g**round
 - t_2 (possibly) **v**ariable
- concretization function γ ,
 e.g., $\gamma(\text{add}(g, v)) = \{\text{add}(z, x), \text{add}(z, z), \text{add}(s(z), x), \text{add}(s(z), z), \dots\}$

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., $\text{add}(g, v)$ denotes the set of terms $\text{add}(t_1, t_2)$ with
 - t_1 (definitely) **g**round
 - t_2 (possibly) **v**ariable
- concretization function γ ,
 e.g., $\gamma(\text{add}(g, v)) = \{\text{add}(z, x), \text{add}(z, z), \text{add}(s(z), x), \text{add}(s(z), z), \dots\}$

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

$\pi(f) \subseteq \{1, \dots, n\}$ for every defined function f/n

Argument filterings over terms & TRSs:

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

$$\pi(l \rightarrow r) = \pi(l) \rightarrow \pi_{\text{rhs}}(r)$$

where $\pi_{\text{rhs}} \approx \pi$ but replaces some extra-variables by a fresh constant \perp

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

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

Argument filterings [Kusakari, Nakamura, Toyama 1999]

$\pi(f) \subseteq \{1, \dots, n\}$ for every defined function f/n

Argument filterings over terms & TRSs:

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

$$\pi(l \rightarrow r) = \pi(l) \rightarrow \pi_{\text{rhs}}(r)$$

where $\pi_{\text{rhs}} \approx \pi$ but replaces some extra-variables by a fresh constant \perp

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

- $\pi(t^\alpha) = f(g, g, \dots, g)$
- 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}

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

where F, G are tuple symbols

Example

```

append(nil, y) → y
append(cons(x, xs), y) → cons(x, append(xs, y))
reverse(nil) → nil
reverse(cons(x, xs)) → append(reverse(xs), cons(x, nil))

```

APPEND(cons(x, xs), y) → APPEND(xs, y) (1)

REVERSE(cons(x, xs)) → REVERSE(xs) (2)

REVERSE(cons(x, xs)) → APPEND(reverse(xs), cons(x, nil)) (3)

Dependency pairs approach

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

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

where F, G are tuple symbols

Example

```

append(nil, y) → y
append(cons(x, xs), y) → cons(x, append(xs, y))
reverse(nil) → nil
reverse(cons(x, xs)) → append(reverse(xs), cons(x, nil))

```

APPEND(cons(x, xs), y) → APPEND(xs, y) (1)

REVERSE(cons(x, xs)) → REVERSE(xs) (2)

REVERSE(cons(x, xs)) → APPEND(reverse(xs), cons(x, nil)) (3)

Dependency pairs approach

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

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

where F, G are tuple symbols

Example

```

append(nil, y) → y
append(cons(x, xs), y) → cons(x, append(xs, y))
reverse(nil) → nil
reverse(cons(x, xs)) → append(reverse(xs), cons(x, nil))

```

APPEND(cons(x, xs), y) → APPEND(xs, y) (1)

REVERSE(cons(x, xs)) → REVERSE(xs) (2)

REVERSE(cons(x, xs)) → APPEND(reverse(xs), cons(x, nil)) (3)

Dependency pairs approach

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

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

where F, G are tuple symbols

Example

```

append(nil, y) → y
append(cons(x, xs), y) → cons(x, append(xs, y))
reverse(nil) → nil
reverse(cons(x, xs)) → append(reverse(xs), cons(x, nil))

```

APPEND(cons(x, xs), y) → APPEND(xs, y) (1)

REVERSE(cons(x, xs)) → REVERSE(xs) (2)

REVERSE(cons(x, xs)) → APPEND(reverse(xs), cons(x, nil)) (3)

Dependency pairs approach

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

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

where F, G are tuple symbols

Example

```

append(nil, y) → y
append(cons(x, xs), y) → cons(x, append(xs, y))
reverse(nil) → nil
reverse(cons(x, xs)) → append(reverse(xs), cons(x, nil))

```

APPEND(cons(x, xs), y) → APPEND(xs, y) (1)

REVERSE(cons(x, xs)) → REVERSE(xs) (2)

REVERSE(cons(x, xs)) → APPEND(reverse(xs), cons(x, nil)) (3)

Dependency pairs approach: differences

Definition (chain)

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

- \exists (constructor) substitution σ such that $\widehat{t_i\sigma} \rightarrow_{\mathcal{R}_{gen}}^* \widehat{s_{i+1}\sigma}$ for $i \geq 1$
- $\pi(\widehat{s_i\sigma}), \pi(\widehat{t_i\sigma})$ 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}_{gen}
- π should filter away all occurrences of gen

Dependency pairs approach: differences

Definition (chain)

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

- \exists (constructor) substitution σ such that $\widehat{t_i\sigma} \rightarrow_{\mathcal{R}_{gen}}^* \widehat{s_{i+1}\sigma}$ for $i \geq 1$
- $\pi(\widehat{s_i\sigma}), \pi(\widehat{t_i\sigma})$ 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}_{gen}
- π should filter away all occurrences of gen

Example

Given the dependency pair

$$\text{APPEND}(\text{cons}(x, xs), y) \rightarrow \text{APPEND}(xs, y) \quad (1)$$

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

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

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

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

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

(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 $\sim_{\mathcal{R}}$ -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 $id(f) = \{1, \dots, n\}$ 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 $\sim_{\mathcal{R}}$ -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 $id(f) = \{1, \dots, n\}$ 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 $\sim_{\mathcal{R}}$ -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 $id(f) = \{1, \dots, n\}$ 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

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

$$\pi = \{\text{append} \mapsto \{1\}, \text{reverse} \mapsto \{1\}\} \quad (\pi \text{ is safe for } t^\alpha)$$

The argument filtering processor returns:

Dependency pairs:
$$\left\{ \begin{array}{l} \text{APPEND}(\text{cons}(x, xs)) \rightarrow \text{APPEND}(xs) \\ \text{REVERSE}(\text{cons}(x, xs)) \rightarrow \text{REVERSE}(xs) \\ \text{REVERSE}(\text{cons}(x, xs)) \rightarrow \text{APPEND}(\text{reverse}(xs)) \end{array} \right.$$

Rewrite system:
$$\left\{ \begin{array}{l} \text{append}(\text{nil}) \rightarrow \perp \\ \text{append}(\text{cons}(x, xs)) \rightarrow \text{cons}(x, \text{append}(xs)) \\ \text{reverse}(\text{nil}) \rightarrow \text{nil} \\ \text{reverse}(\text{cons}(x, xs)) \rightarrow \text{append}(\text{reverse}(xs)) \end{array} \right.$$

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

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 $\text{AFT}_\pi(\mathcal{R})$

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

- a filtered rule $\pi(l) \rightarrow \pi_{rhs}(r)$ and
- an additional rule $\pi(l) \rightarrow \pi(t)$, for each subterm t of r that is filtered away in $\pi_{rhs}(r)$ 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 $\text{AFT}_{\pi}(\mathcal{R})$

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

- a filtered rule $\pi(l) \rightarrow \pi_{rhs}(r)$ and
- an additional rule $\pi(l) \rightarrow \pi(t)$, for each subterm t of r that is filtered away in $\pi_{rhs}(r)$ 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 $\text{AFT}_\pi(\mathcal{R})$ is terminating*

Therefore,

- $\text{AFT}_\pi(\mathcal{R})$ can be analyzed using standard techniques and tools for proving the termination of TRSs
(no data generator is involved in the derivations of $\text{AFT}_\pi(\mathcal{R})$)

Example

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

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

$$\pi = \{\text{append} \mapsto \{1\}, \text{reverse} \mapsto \{1\}\}$$

$$(\pi \text{ is safe for } t^\alpha)$$

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

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

which is clearly **not** terminating

Example

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

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

$$\pi = \{\text{append} \mapsto \{1\}, \text{reverse} \mapsto \{1\}\}$$

$$(\pi \text{ is safe for } t^\alpha)$$

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

$$\begin{aligned} \text{append}(\text{nil}) &\rightarrow \color{red}{y} \perp \quad (\perp \text{ is a fresh constant}) \\ \text{append}(\text{cons}(x, xs)) &\rightarrow \text{cons}(x, \text{append}(xs)) \\ \text{reverse}(\text{nil}) &\rightarrow \text{nil} \\ \text{reverse}(\text{cons}(x, xs)) &\rightarrow \text{append}(\text{reverse}(xs)) \end{aligned}$$

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

$$g \sqcup g = g \quad g \sqcup v = v \quad v \sqcup g = v \quad v \sqcup v = v$$

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

$$\begin{aligned} \{f \mapsto (g, v), g \mapsto (g, v)\} \sqcup \{f \mapsto (g, g), g \mapsto (v, g)\} \\ = \{f \mapsto (g, v), g \mapsto (v, v)\} \end{aligned}$$

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

Auxiliary functions

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

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

Roughly speaking,

- $(B_v[[t]] \text{ g/n } \rho)$ 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

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

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

Roughly speaking,

- $(B_v[[t]] \text{ g/n } \rho)$ 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

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

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

Roughly speaking,

- $(B_v[[t]] \text{ g/n } \rho)$ 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 $f_1(m_1, \dots, m_{n_1})$, the initial division is

$$div_0 = \{f_1 \mapsto (m_1, \dots, m_{n_1}), f_2 \mapsto (g, \dots, g), \dots, f_k \mapsto (g, \dots, g)\}$$

where $f_1/n_1, \dots, f_k/n_k$ are the defined functions of the TRS

Iterative process

$$div_i = \{f_1 \mapsto b_1, \dots, f_k \mapsto b_k\}$$

$$\Downarrow$$

$$div_{i+1} = \{ f_1 \mapsto b_1 \sqcup B_v[[r_1]] f_1/n_1 e(b_1, l_1) \sqcup \dots \sqcup B_v[[r_j]] f_1/n_1 e(b_j, l_j), \\ \dots, \\ f_k \mapsto b_k \sqcup B_v[[r_1]] f_k/n_k e(b_1, l_1) \sqcup \dots \sqcup B_v[[r_j]] f_k/n_k e(b_j, l_j) \}$$

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

$$e((m_1, \dots, m_n), f(t_1, \dots, t_n)) = \{x \mapsto m_1 \mid x \in \text{Var}(t_1)\} \\ \cup \dots \\ \cup \{x \mapsto m_n \mid x \in \text{Var}(t_n)\}$$

BTA algorithm

Given an abstract term $f_1(m_1, \dots, m_{n_1})$, the initial division is

$$div_0 = \{f_1 \mapsto (m_1, \dots, m_{n_1}), f_2 \mapsto (g, \dots, g), \dots, f_k \mapsto (g, \dots, g)\}$$

where $f_1/n_1, \dots, f_k/n_k$ are the defined functions of the TRS

Iterative process

$$div_i = \{f_1 \mapsto b_1, \dots, f_k \mapsto b_k\}$$

$$\Downarrow$$

$$div_{i+1} = \{ f_1 \mapsto b_1 \sqcup B_v[[r_1]] f_1/n_1 e(b_1, l_1) \sqcup \dots \sqcup B_v[[r_j]] f_1/n_1 e(b_j, l_j), \\ \dots, \\ f_k \mapsto b_k \sqcup B_v[[r_1]] f_k/n_k e(b_1, l_1) \sqcup \dots \sqcup B_v[[r_j]] f_k/n_k e(b_j, l_j) \}$$

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

$$e((m_1, \dots, m_n), f(t_1, \dots, t_n)) = \{x \mapsto m_1 \mid x \in \text{Var}(t_1)\} \\ \cup \dots \\ \cup \{x \mapsto m_n \mid x \in \text{Var}(t_n)\}$$

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

Given the division

$$div = \{f_1 \mapsto (m_1^1, \dots, m_{n_1}^1), \dots, f_k \mapsto (m_1^k, \dots, m_{n_k}^k)\}$$

we have

$$\pi(div) = \{f_1 \mapsto \{i \mid m_i^1 = g\}, \dots, f_k \mapsto \{i \mid m_i^k = g\}\}$$

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

Example

$$\begin{array}{ll}
 \text{mult}(z, y) \rightarrow z & \text{add}(z, y) \rightarrow y \\
 \text{mult}(s(x), y) \rightarrow \text{add}(\text{mult}(x, y), y) & \text{add}(s(x), y) \rightarrow s(\text{add}(x, y))
 \end{array}$$

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

$$\text{div}_0 = \{\text{mult} \mapsto (g, v), \text{add} \mapsto (g, g)\}$$

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

$$\begin{array}{l}
 \text{div}_1 = \{ \text{mult} \mapsto (g, v) \quad \sqcup \quad B_v[[z]] \text{mult}/2 \{y \mapsto v\} \\
 \quad \quad \quad \sqcup \quad B_v[[\text{add}(\text{mult}(x, y), y)]] \text{mult}/2 \{x \mapsto g, y \mapsto v\} \\
 \quad \quad \quad \sqcup \quad B_v[[y]] \text{mult}/2 \{y \mapsto g\} \\
 \quad \quad \quad \sqcup \quad B_v[[s(\text{add}(x, y))]] \text{mult}/2 \{x \mapsto g, y \mapsto g\}, \\
 \text{add} \mapsto (g, g) \quad \sqcup \quad B_v[[z]] \text{add}/2 \{y \mapsto v\} \\
 \quad \quad \quad \sqcup \quad B_v[[\text{add}(\text{mult}(x, y), y)]] \text{add}/2 \{x \mapsto g, y \mapsto v\} \\
 \quad \quad \quad \sqcup \quad B_v[[y]] \text{add}/2 \{y \mapsto g\} \\
 \quad \quad \quad \sqcup \quad B_v[[s(\text{add}(x, y))]] \text{add}/2 \{x \mapsto g, y \mapsto g\} \quad \quad \quad \}
 \end{array}$$

Example

$$\begin{array}{ll} \text{mult}(z, y) \rightarrow z & \text{add}(z, y) \rightarrow y \\ \text{mult}(s(x), y) \rightarrow \text{add}(\text{mult}(x, y), y) & \text{add}(s(x), y) \rightarrow s(\text{add}(x, y)) \end{array}$$

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

$$\text{div}_0 = \{ \text{mult} \mapsto (g, v), \text{add} \mapsto (g, g) \}$$

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

$$\begin{array}{l} \text{div}_1 = \{ \text{mult} \mapsto (g, v) \quad \sqcup \quad B_v[[z]] \text{mult}/2 \{y \mapsto v\} \\ \quad \sqcup \quad B_v[[\text{add}(\text{mult}(x, y), y)]] \text{mult}/2 \{x \mapsto g, y \mapsto v\} \\ \quad \sqcup \quad B_v[[y]] \text{mult}/2 \{y \mapsto g\} \\ \quad \sqcup \quad B_v[[s(\text{add}(x, y))]] \text{mult}/2 \{x \mapsto g, y \mapsto g\}, \\ \text{add} \mapsto (g, g) \quad \sqcup \quad B_v[[z]] \text{add}/2 \{y \mapsto v\} \\ \quad \sqcup \quad B_v[[\text{add}(\text{mult}(x, y), y)]] \text{add}/2 \{x \mapsto g, y \mapsto v\} \\ \quad \sqcup \quad B_v[[y]] \text{add}/2 \{y \mapsto g\} \\ \quad \sqcup \quad B_v[[s(\text{add}(x, y))]] \text{add}/2 \{x \mapsto g, y \mapsto g\} \quad \} \end{array}$$

Example (cont'd)

Therefore, by evaluating the calls to B_v , we get

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

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)

⇒ If we compute div_2 we get $div_1 = div_2 \implies$ div₁ is a fixpoint

From this division, the associated safe argument filtering is

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

Example (cont'd)

Therefore, by evaluating the calls to B_v , we get

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

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 $div_1 = div_2 \Rightarrow$** **$div_1$ is a fixpoint**

From this division, the associated safe argument filtering is

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

Example (cont'd)

Therefore, by evaluating the calls to B_v , we get

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

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 $div_1 = div_2 \Rightarrow$** **$div_1$ is a fixpoint**

From this division, the associated safe argument filtering is

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

Some refinements

Multiple abstract terms

Consider, e.g.,

$$\begin{aligned} \text{eq}(z, z) &\rightarrow \text{true} \\ \text{eq}(s(x), s(y)) &\rightarrow \text{eq}(x, y) \end{aligned}$$

and the set

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

Here, starting from

$$\text{div}_0 = \{ \text{eq} \mapsto (g, v) \sqcup (v, g) \} = \{ \text{eq} \mapsto (v, v) \}$$

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 $\rightsquigarrow_{\mathcal{R}}$ -terminating iff $\gamma(t^\alpha)$ is $\rightsquigarrow_{\mathcal{R}}$ -terminating for all $t^\alpha \in T^\alpha$.

Some refinements

Multiple abstract terms

Consider, e.g.,

$$\begin{aligned} \text{eq}(z, z) &\rightarrow \text{true} \\ \text{eq}(s(x), s(y)) &\rightarrow \text{eq}(x, y) \end{aligned}$$

and the set

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

Here, starting from

$$\text{div}_0 = \{ \text{eq} \mapsto (g, v) \sqcup (v, g) \} = \{ \text{eq} \mapsto (v, v) \}$$

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 $\rightsquigarrow_{\mathcal{R}}$ -terminating iff $\gamma(t^\alpha)$ is $\rightsquigarrow_{\mathcal{R}}$ -terminating for all $t^\alpha \in T^\alpha$.

Some refinements (cont'd)

Non well-moded programs

Consider, e.g.,

$$\begin{aligned} \text{eq}(z, z) &\rightarrow \text{true} \\ \text{eq}(s(x), s(y)) &\rightarrow \text{eq}(y, x) \end{aligned}$$

If we start with

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

the only safe argument filtering is

$$\pi = \{\text{eq} \mapsto \{\}\}$$

Solution

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

Some refinements (cont'd)

Non well-moded programs

Consider, e.g.,

$$\begin{aligned} \text{eq}(z, z) &\rightarrow \text{true} \\ \text{eq}(s(x), s(y)) &\rightarrow \text{eq}(y, x) \end{aligned}$$

If we start with

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

the only safe argument filtering is

$$\pi = \{\text{eq} \mapsto \{\}\}$$

Solution

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

Some refinements (cont'd)

Removing non-reachable functions

Consider, e.g.,

$$a \rightarrow a$$
$$b \rightarrow c$$
$$c \rightarrow d$$

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., $a \rightarrow a$)

Some refinements (cont'd)

Removing non-reachable functions

Consider, e.g.,

$$a \rightarrow a$$
$$b \rightarrow c$$
$$c \rightarrow d$$

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., $a \rightarrow a$)

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 and Miura \[NM06\]](#) adapted the dependency pair method for proving the termination of narrowing:

- direct approach (not based on using generators & rewriting)
- allow extra variables in TRSs
- not comparable





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
- application to (offline) partial evaluation

-  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.
-  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.