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)**

\[
\begin{align*}
\text{add}(z, y) & \rightarrow y \quad (R_1) \\
\text{add}(s(x), y) & \rightarrow s(\text{add}(x, y)) \quad (R_2)
\end{align*}
\]

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

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

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

\( \text{s(add}(y, z)) \)

\( \text{s}(z) \)

\( \text{add}(s(y), z) \)

\( R_2 \)

\( \text{s}(\text{add}(z, z)) \)

\( R_1 \)

“guess”

(many other non-deterministic reductions possible...)
What is narrowing?

Standard definition of addition (TRS)

\[
\begin{align*}
\text{add}(z, y) & \rightarrow y \quad (R_1) \\
\text{add}(s(x), y) & \rightarrow s(\text{add}(x, y)) \quad (R_2)
\end{align*}
\]

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

(many other non-deterministic reductions possible... )
What is narrowing?

Standard definition of addition (TRS)

\[
\begin{align*}
\text{add}(z, y) & \rightarrow y \quad (R_1) \\
\text{add}(s(x), y) & \rightarrow s(\text{add}(x, y)) \quad (R_2)
\end{align*}
\]

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: \( \text{add}(x, z) \)

\[
\begin{align*}
\text{add}(s(y), z) & \quad \text{“guess”} \\
\text{add}(s(y), z) & \rightarrow_{R_2} s(\text{add}(y, z)) \\
\text{add}(s(y), z) & \rightarrow_{R_1} s(\text{add}(z, z)) \\
\text{s}(z) & \quad \text{“guess”}
\end{align*}
\]

(many other non-deterministic reductions possible. . . )
What is narrowing?

Standard definition of addition (TRS)

\[
\begin{align*}
\text{add}(z, y) & \rightarrow y & (R_1) \\
\text{add}(s(x), y) & \rightarrow s(\text{add}(x, y)) & (R_2)
\end{align*}
\]

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

With **narrowing**:
\[
\text{add}(s(z), z) \rightharpoonup_{R_2} s(\text{add}(z, z)) \rightharpoonup_{R_1} s(z)
\]

but also:
\[
\text{add}(x, z) \rightharpoonup_{R_2, \{x \mapsto s(y)\}} s(\text{add}(y, z)) \rightharpoonup_{\{R_1, y \mapsto z\}} s(z)
\]

(\text{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 \( \sigma \) such that \( s|_p = l\sigma \)

\[ \downarrow \]

**Definition (narrowing)**

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

- a **nonvariable** position \( p \) of \( s \)
- a **variant** \( R = (l \rightarrow r) \) of a rule in \( \mathcal{R} \)
- a substitution \( \sigma \) 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!

\[ \text{add}(x, y) \xrightarrow{R_2, \{x \mapsto s(x')\}} \text{add}(x', y) \xrightarrow{R_2, \{x' \mapsto s(x'')\}} \ldots \]

In this work

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

For instance, \( \{ \text{add}(s, t) \mid s \text{ is ground} \} \) is \( \xrightarrow{} R \)-terminating
Termination of narrowing

The termination problem

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

Too strong!

\[ \text{add}(x, y) \leadsto_{R_2, \{x \mapsto s(x')\}} \text{add}(x', y) \leadsto_{R_2, \{x' \mapsto s(x'')\}} \ldots \]

In this work

- given a TRS \( R \) 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 \( \leadsto_R \)-terminating)

For instance, \( \{ \text{add}(s, t) \mid s \text{ is ground} \} \) is \( \leadsto_R \)-terminating
Termination of narrowing

The termination problem

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

Too strong!

\[ \text{add}(x, y) \sim_{R_2, \{x \mapsto s(x')\}} \text{add}(x', y) \sim_{R_2, \{x' \mapsto s(x'')\}} \ldots \]

In this work

- given a TRS \( R \) and a set of terms \( T \), are all possible narrowing derivations \( t_1 \sim t_2 \sim \ldots \) for \( t_1 \in T \) finite? (in symbols: \( T \) is \( \sim_R \)-terminating)

For instance, \( \{ \text{add}(s, t) \mid s \text{ is ground} \} \) is \( \sim_R \)-terminating
Termination of narrowing via termination of rewriting

Theorem

\( T \) is \( \leadsto_R \)-terminating if \( \{ t\sigma \mid t \in T \text{ and } t \leadsto^* s \text{ in } R \} \) is finite and \( \rightarrow_R \)-terminating

Drawbacks:

- very difficult to approximate
- sufficient but not necessary:

\[
\begin{align*}
  f(a) & \rightarrow b \\
  a & \rightarrow a
\end{align*}
\]

The set \( \{ f(x) \} \) is \( \leadsto_R \)-terminating
however \( \{ f(a) \} \) is finite but not \( \rightarrow_R \)-terminating:

\[
\begin{align*}
  f(a) & \rightarrow f(a) \rightarrow f(a) \rightarrow \ldots
\end{align*}
\]
Termination of narrowing via termination of rewriting

**Theorem**

\[ T \text{ is } \leadsto_R \text{-terminating} \]

if \(\{ t\sigma \mid t \in T \text{ and } t \leadsto^* \sigma s \text{ in } \mathcal{R} \}\) is finite and \(\rightarrow_R\)-terminating

**Drawbacks:**

- very difficult to approximate
- sufficient but not necessary:

\[
\begin{align*}
 f(a) & \rightarrow b \\
 a & \rightarrow a
\end{align*}
\]

The set \(\{f(x)\}\) is \(\leadsto_R\)-terminating

however \(\{f(a)\}\) is finite but not \(\rightarrow_R\)-terminating:

\[
\begin{align*}
 f(a) & \rightarrow f(a) \rightarrow f(a) \rightarrow \ldots
\end{align*}
\]
Termination of narrowing via termination of rewriting

**Theorem**

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

**Drawbacks:**

- very difficult to approximate
- sufficient but not necessary:

  \[
  f(a) \rightarrow b \\
  a \rightarrow a
  \]

  The set $\{f(x)\}$ is $\leadsto_R$-terminating
  however $\{f(a)\}$ is finite but not $\rightarrow_R$-terminating:

  \[
  f(a) \rightarrow f(a) \rightarrow f(a) \rightarrow \ldots
  \]
A first solution

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

Therefore:

\[ \{ t\sigma \mid t \in T \text{ and } t \xrightarrow{\sigma}^* s \text{ in } \mathcal{R} \} \]

\[ \Downarrow \]

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

Why infinite terms?

Example:

\[
\begin{align*}
\text{add}(z, y) & \to y \quad (R_1) \\
\text{add}(s(x), y) & \to s(\text{add}(x, y)) \quad (R_2)
\end{align*}
\]

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

\[
\text{add}(s(s(\ldots)), z) \to_\mathcal{R} s(\text{add}(s(s(\ldots)), z)) \to_\mathcal{R} \ldots
\]
A first solution

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

Therefore

\[ \{ t \sigma \mid t \in T \text{ and } t \xrightarrow{\sigma}^* s \text{ in } \mathcal{R} \} \]

\[ \Downarrow \]

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

Why infinite terms?

Example

\[
\begin{align*}
\text{add}(z, y) & \rightarrow y & (R_1) \\
\text{add}(s(x), y) & \rightarrow s(\text{add}(x, y)) & (R_2)
\end{align*}
\]

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

\[
\text{add}(s(s(\ldots)), z) \rightarrow_\mathcal{R} s(\text{add}(s(s(\ldots)), z)) \rightarrow_\mathcal{R} \ldots
\]
A first solution

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

Therefore
\[ \{ t\sigma \mid t \in T \text{ and } t \xrightarrow{\sigma}^* s \text{ in } \mathcal{R} \} \]
\[ \Downarrow \]
\[ \{ t\sigma \mid t \in T \text{ and } \sigma \text{ maps variables to possibly infinite terms} \} \]

Why infinite terms?

Example

\[
\begin{align*}
\text{add}(z, y) & \rightarrow y \quad (R_1) \\
\text{add}(s(x), y) & \rightarrow s(\text{add}(x, y)) \quad (R_2)
\end{align*}
\]

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

\[
\text{add}(s(s(\ldots)), z) \xrightarrow{\mathcal{R}} s(\text{add}(s(s(\ldots)), z)) \rightarrow_{\mathcal{R}} \ldots
\]
**Problem**

proving that the set

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

is \( \rightarrow_R \)-terminating is often too strong...

**Example**  Given the TRS

\[
\begin{align*}
    a & \rightarrow a \\
    f(x) & \rightarrow x
\end{align*}
\]

f(x) is clearly \( \sim_R \)-terminating

but \( \exists \sigma \) such that \( f(x)\sigma \) is not \( \rightarrow_R \)-terminating \hspace{1cm} (e.g., \( \sigma = \{ x \mapsto a \} \))

\( \Rightarrow \) an infinite computation \( f(a) \rightarrow_R f(a) \rightarrow_R \ldots \) is introduced by \( \sigma \)!!
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

\[
\begin{align*}
a &\rightarrow a \\
f(x) &\rightarrow x
\end{align*}
\]

$f(x)$ is clearly $\rightsquigarrow_\mathcal{R}$-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 $f(a) \rightarrow_\mathcal{R} f(a) \rightarrow_\mathcal{R} \ldots$ is introduced by $\sigma$!!
Problem

proving that the set

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

is \( \rightarrow_R \)-terminating is often too strong. . .

Example Given the TRS

\[
\begin{align*}
a & \rightarrow a \\
f(x) & \rightarrow x
\end{align*}
\]

\( f(x) \) is clearly \( \sim_R \)-terminating

but \( \exists \sigma \) such that \( f(x)\sigma \) is not \( \rightarrow_R \)-terminating \hspace{1cm} (e.g., \( \sigma = \{ x \mapsto a \} \))

\Rightarrow an \ infinite \ computation \( f(a) \rightarrow_R f(a) \rightarrow_R \ldots \) is introduced by \( \sigma \) !!
A second solution

⇒ forbid the reduction of redexes introduced by $\sigma$ ...

A second problem...

...this restriction makes the condition unsound!

Example

Given the TRS

\[
\begin{align*}
a & \rightarrow a \\
f(a) & \rightarrow c(b, b)
\end{align*}
\]

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

(e.g., $c(y, f(y)) \sim_{\{y \mapsto a\}} c(a, c(b, b)) \sim_{id} c(a, c(b, b)) \sim_{id} \ldots$)
A second solution

⇒ forbid the reduction of redexes introduced by $\sigma$...

A second problem...

...this restriction makes the condition unsound!

Example

Given the TRS

\[
\begin{align*}
a & \rightarrow a \\
f(a) & \rightarrow c(b, b)
\end{align*}
\]

- $c(y, f(y))\sigma$ is $\rightarrow_R$-terminating if the reduction of the terms introduced by $\sigma$ is forbidden
- but $c(y, f(y))$ is not $\sim_R$-terminating!!

(e.g., $c(y, f(y)) \xrightarrow{\{y \mapsto a\}} c(a, c(b, b)) \xrightarrow{id} c(a, c(b, b)) \xrightarrow{id} ...$)
A second solution

⇒ forbid the reduction of redexes introduced by σ...

A second problem...

...this restriction makes the condition unsound!

Example

Given the TRS

\[
\begin{align*}
  a & \rightarrow a \\
  f(a) & \rightarrow c(b, b)
\end{align*}
\]

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

\((e.g., \quad c(y, f(y)) \leadsto_{\{y \mapsto a\}} c(a, c(b, b)) \leadsto_{id} c(a, c(b, b)) \leadsto_{id} \ldots)\)
A second solution

⇒ forbid the reduction of redexes introduced by $\sigma$...

A second problem...

...this restriction makes the condition unsound!

Example Given the TRS

\[
\begin{align*}
  a & \rightarrow a \\
  f(a) & \rightarrow c(b, b)
\end{align*}
\]

- $c(y, f(y))\sigma$ is $\rightarrow_R$-terminating if the reduction of the terms introduced by $\sigma$ is forbidden
- but $c(y, f(y))$ is not $\leadsto_R$-terminating!!

(e.g., $c(y, f(y)) \leadsto_{\{y \mapsto a\}} c(a, c(b, b)) \leadsto_{id} c(a, c(b, b)) \leadsto_{id} \ldots$)
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
In the following, we consider **left-linear constructor** TRSs:

\[
f_1(t_{11}, \ldots, t_{1m_1}) \rightarrow r_1
\]

\[\ldots\]

\[
f_n(t_{n1}, \ldots, t_{nm_n}) \rightarrow r_n
\]

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

\[\Rightarrow\]

**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 $R$, we define $R_{\text{gen}}$ as $R$ augmented with

$$\begin{align*}
gen & \rightarrow \underbrace{c(gen, \ldots, gen)}_{n \text{ times}} \\
\text{for all constructor } c/n \in C, \ n \geq 0
\end{align*}$$

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

$$R_{\text{gen}} = R \cup \left\{ \begin{array}{c}
gen \rightarrow z \\
gen \rightarrow s(gen)
\end{array} \right\}$$

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 $R$, we define $R_{\text{gen}}$ as $R$ augmented with

$$\begin{array}{c}
gen \rightarrow n \text{ times } c(\text{gen}, \ldots, \text{gen}) \\
\end{array}$$

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

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

$$R_{\text{gen}} = R \cup \left\{ \begin{array}{c}
gen \rightarrow z \\
gen \rightarrow s(\text{gen}) \\
\end{array} \right\}$$

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

$$\begin{align*}
gen & \rightarrow c(gen, \ldots, gen) \\
& \text{for all constructor } c/n \in \mathcal{C}, \ n \geq 0
\end{align*}$$

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

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

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

**Completeness**

\[
\text{If } s \sim_{\sigma} t \text{ in } \mathcal{R} \quad \text{then} \quad \hat{s} \rightarrow^*_{\text{gen}} \hat{s}_{\sigma} \rightarrow \hat{t} \text{ in } \mathcal{R}_{\text{gen}}
\]

**Generally unsound**

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

but

\[
\begin{align*}
\text{add}(x, x) & \sim_{\{x \mapsto z\}} z \\
\text{add}(x, x) & \sim_{\{x \mapsto \text{s}(x')\}} \text{s}(\text{add}(x', \text{s}(x'))) \sim_{\{x' \mapsto z\}} \text{s}(\text{s}(z))
\end{align*}
\]

... 

**Soundness is preserved for admissible derivations**

- a derivation is admissible iff all the occurrences of \text{gen} originating from the replacement of the same variable are reduced to the same term.
Correctness of data generators

Completeness

\[
\text{If } s \xrightarrow{\sigma} t \text{ in } \mathcal{R} \text{ then } \hat{s} \xrightarrow{*_{\text{gen}}} \hat{s}\sigma \rightarrow \hat{t} \text{ in } \mathcal{R}_{\text{gen}}
\]

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

\[
\text{add}(x, x) \xrightarrow{\{x \mapsto z\}} z
\]
\[
\text{add}(x, x) \xrightarrow{\{x \mapsto s(x')\}} s(\text{add}(x', s(x'))) \xrightarrow{\{x' \mapsto z\}} s(s(z))
\]

... 

Soundness is preserved for admissible derivations

- a derivation is admissible iff all the occurrences of \( \text{gen} \) originating from the replacement of the same variable are reduced to the same term
Correctness of data generators

Completeness

\[
\text{If } s \leadsto^\sigma t \text{ in } \mathcal{R} \text{ then } \hat{s} \rightarrow^*_\text{gen} \hat{s}\sigma \rightarrow \hat{t} \text{ in } \mathcal{R}_\text{gen}
\]

Generally unsound

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

but

\[
\begin{align*}
\text{add}(x, x) & \leadsto^{\{x \mapsto z\}} z \\
\text{add}(x, x) & \leadsto^{\{x \mapsto \text{s}(x')\}} \text{s}(\text{add}(x', \text{s}(x'))) \leadsto^{\{x' \mapsto z\}} \text{s}(\text{s}(z))
\end{align*}
\]

... 

Soundness is preserved for admissible derivations

- a derivation is admissible iff all the occurrences of \text{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 $\text{gen}$ terminates!

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

- $T$ is relatively $\mathcal{R}_{\text{gen}}$-terminating to $\mathcal{R}$ if every derivation $t_1 \rightarrow t_2 \ldots$ 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 $\sim_{\mathcal{R}}$-terminating

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

$\Rightarrow$ sufficient condition
What about termination in $R_{gen}$?

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

Fortunately, relative termination of $R_{gen}$ suffices:

- $T$ is relatively $R_{gen}$-terminating to $R$ if every derivation $t_1 \rightarrow t_2 \ldots$ for $t_1 \in T$ contains finitely many $\rightarrow_R$ steps.

**Theorem (termination of narrowing via termination of rewriting)**

Let $R$ be a left-linear constructor TRS

$T$ is $\leadsto_R$-terminating

$\hat{T}$ is relatively $\rightarrow_{R_{gen}}$-terminating to $R$

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

Clearly, no term with occurrences of $\text{gen}$ terminates!

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

- $T$ is relatively $\mathcal{R}_{\text{gen}}$-terminating to $\mathcal{R}$ if every derivation $t_1 \rightarrow t_2 \ldots$ 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 $\sim_{\mathcal{R}}$-terminating

iff

$\hat{T}$ is relatively $\rightarrow_{\mathcal{R}_{\text{gen}}}$-terminating to $\mathcal{R}$ w.r.t. admissible derivations

$\implies$ **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 $t_1 \rightarrow t_2 \ldots$ 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 $\sim_{\mathcal{R}}$-terminating

iff

$\hat{T}$ is relatively $\rightarrow_{\mathcal{R}_{\text{gen}}}$-terminating to $\mathcal{R}$ w.r.t. admissible derivations

$\implies$ sufficient condition
automating the termination analysis
Proving termination automatically

The problem

Given $\mathcal{R}$ and $\mathcal{T}$,

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

Drawback

- the set $\mathcal{T}$ 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) ground
  - $t_2$ (possibly) variable

- concretization function $\gamma$,

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

Drawback

- the set $\mathcal{T}$ 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) ground
  - $t_2$ (possibly) variable
- concretization function $\gamma$,
  
  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 $R$ and $T$, $T$ is $\sim R$-terminating if $\hat{T}$ is relatively $\rightarrow R_{\text{gen}}$-terminating to $R$

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) ground
  - $t_2$ (possibly) variable
- concretization function $\gamma$
  e.g., $\gamma(\text{add}(g, v)) = \{\text{add}(z, x), \text{add}(z, z), \text{add}(s(z), x), \text{add}(s(z), z), \ldots\}$
The problem (revised)

Given $\mathcal{R}$ and $t^\alpha$,

$\gamma(t^\alpha)$ is $\sim_{\mathcal{R}}$-terminating if $\gamma(t^\alpha)$ 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 $t^{\alpha}$,

$\gamma(t^{\alpha})$ 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 $t^\alpha$, $\gamma(t^\alpha)$ is $\sim_{\mathcal{R}}$-terminating if $\gamma(t^\alpha)$ 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 \[\text{[Kusakari, Nakamura, Toyama 1999]}\]

\[
\pi(f) \subseteq \{1, \ldots, n\} \text{ for every defined function } f/n
\]

Argument filterings over terms & TRSs:

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

\[
\pi(l \rightarrow r) = \pi(l) \rightarrow \pi(r)
\]

From \(t^\alpha\) we infer a \textbf{safe argument filtering} \(\pi\) for \(t^\alpha\)

- \(\pi(t^\alpha) = f(g, g, \ldots, g)\)
- for all \(s \sim t\), if \(\pi(s|_p)\) are ground then \(\pi(t|_q)\) are ground too
Argument filterings [Kusakari, Nakamura, Toyama 1999]

\[ \pi(f) \subseteq \{1, \ldots, n\} \text{ for every defined function } f/n \]

Argument filterings over terms & TRSs:

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

\[ \pi(l \rightarrow r) = \pi(l) \rightarrow \pi(r) \]

From \( t^\alpha \) we infer a safe argument filtering \( \pi \) for \( t^\alpha \)

- \( \pi(t^\alpha) = f(g, g, \ldots, g) \)
- for all \( s \leadsto t \), 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** \( \mathcal{R} \)

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

where \( F, G \) are tuple symbols

**Example**

\[
\begin{align*}
\text{append}(\text{nil}, y) & \rightarrow y \\
\text{append}(\text{cons}(x, xs), y) & \rightarrow \text{cons}(x, \text{append}(xs, y)) \\
\text{reverse}(\text{nil}) & \rightarrow \text{nil} \\
\text{reverse}(\text{cons}(x, xs)) & \rightarrow \text{append}(\text{reverse}(xs), \text{cons}(x, \text{nil}))
\end{align*}
\]

\[
\begin{align*}
\text{APPEND}(\text{cons}(x, xs), y) & \rightarrow \text{APPEND}(xs, y) \quad (1) \\
\text{REVERSE}(\text{cons}(x, xs)) & \rightarrow \text{REVERSE}(xs) \quad (2) \\
\text{REVERSE}(\text{cons}(x, xs)) & \rightarrow \text{APPEND}(\text{reverse}(xs), \text{cons}(x, \text{nil})) \quad (3)
\end{align*}
\]
Dependency pairs approach

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

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

where $F, G$ are tuple symbols

**Example**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{append}(\text{nil}, y) \rightarrow y$</td>
<td></td>
</tr>
<tr>
<td>$\text{append}(\text{cons}(x, xs), y) \rightarrow \text{cons}(x, \text{append}(xs, y))$</td>
<td></td>
</tr>
<tr>
<td>$\text{reverse}(\text{nil}) \rightarrow \text{nil}$</td>
<td></td>
</tr>
<tr>
<td>$\text{reverse}(\text{cons}(x, xs)) \rightarrow \text{append}(\text{reverse}(xs), \text{cons}(x, \text{nil}))$</td>
<td></td>
</tr>
</tbody>
</table>

APPEND($\text{cons}(x, xs), y$) $\rightarrow$ APPEND($xs, y$)  
(1)
REVERSE($\text{cons}(x, xs)$) $\rightarrow$ REVERSE($xs$)  
(2)
REVERSE($\text{cons}(x, xs)$) $\rightarrow$ APPEND($\text{reverse}(xs), \text{cons}(x, \text{nil})$)  
(3)
Dependency pairs approach

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

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

where $F$, $G$ are tuple symbols

**Example**

append(nil, y) $\rightarrow$ y
append(cons(x, xs), y) $\rightarrow$ cons(x, append(xs, y))
reverse(nil) $\rightarrow$ nil
reverse(cons(x, xs)) $\rightarrow$ append(reverse(xs), cons(x, nil))

APPEND(cons(x, xs), y) $\rightarrow$ APPEND(xs, y) (1)
REVERSE(cons(x, xs)) $\rightarrow$ REVERSE(xs) (2)
REVERSE(cons(x, xs)) $\rightarrow$ APPEND(reverse(xs), cons(x, nil)) (3)
Dependency pairs approach

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

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

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($xs$, $y$) \hspace{1cm} (1)

REVERSE(cons($x$, $xs$)) → REVERSE($xs$) \hspace{1cm} (2)

REVERSE(cons($x$, $xs$)) → APPEND(reverse($xs$), cons($x$, nil)) \hspace{1cm} (3)
Dependency pairs approach

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

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

where $F, G$ are tuple symbols

**Example**

- $\text{append}(\text{nil}, y) \rightarrow y$
- $\text{append}(\text{cons}(x, xs), y) \rightarrow \text{cons}(x, \text{append}(xs, y))$
- $\text{reverse}(\text{nil}) \rightarrow \text{nil}$
- $\text{reverse}(\text{cons}(x, xs)) \rightarrow \text{append}(\text{reverse}(xs), \text{cons}(x, \text{nil}))$

\[
\begin{align*}
\text{APPEND}(\text{cons}(x, xs), y) & \rightarrow \text{APPEND}(xs, y) \quad (1) \\
\text{REVERSE}(\text{cons}(x, xs)) & \rightarrow \text{REVERSE}(xs) \quad (2) \\
\text{REVERSE}(\text{cons}(x, xs)) & \rightarrow \text{APPEND}(\text{reverse}(xs), \text{cons}(x, \text{nil})) \quad (3)
\end{align*}
\]
Dependency pairs approach: differences

**Definition (chain)**

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

- $\exists$ (constructor) substitution $\sigma$ such that $\hat{t}_i \sigma \rightarrow^*_{\mathcal{R}_{gen}} \hat{s}_{i+1} \sigma$ for $i \geq 1$

- $\pi(\hat{s}_i \sigma), \pi(\hat{t}_i \sigma)$ contain no occurrences of gen

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

- It is parameterized by $\pi$

- Variables are replaced by gen and reductions w.r.t. $\mathcal{R}_{gen}$

- $\pi$ should filter away all occurrences of gen
Dependency pairs approach: differences

**Definition (chain)**

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

- $\exists$ (constructor) substitution $\sigma$ such that $\hat{t}_i \sigma \rightarrow^*_{R_{gen}} \hat{s}_{i+1} \sigma$ for $i \geq 1$
- $\pi(\hat{s}_i \sigma), \pi(\hat{t}_i \sigma)$ contain no occurrences of gen

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

- it is parameterized by $\pi$
- variables are replaced by gen and reductions w.r.t. $R_{gen}$
- $\pi$ should filter away all occurrences of gen
Example

Given the dependency pair

\[
\text{APPEND}(\text{cons}(x, xs), y) \rightarrow \text{APPEND}(xs, y)
\]  

(1)

we have an infinite \((\text{DP}(R), R, \pi)-\text{chain}, (1),(1),\ldots,\) for

\[
\pi(\text{append}) = \pi(\text{APPEND}) = \{2\}
\]

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

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

\[
\overset{\tilde{t}\sigma}{\downarrow} \quad \overset{\tilde{t}\sigma}{\uparrow}
\]

\[
\text{APPEND}(\text{cons}(\text{gen, gen}), \text{nil}) \quad \leftarrow \quad \text{APPEND}(\text{gen, nil})
\]

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

(not a chain in the standard framework of rewriting)
Theorem

Let $\pi$ be a safe argument filtering for $t^\alpha$ 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

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

where $id(f) = \{1, \ldots, n\}$ 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 $\pi$ be a safe argument filtering for $t^\alpha$ 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

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

where $id(f) = \{1, \ldots, n\}$ 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 $\pi$ be a safe argument filtering for $t^\alpha$ 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

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

where $id(f) = \{1, \ldots, n\}$ 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

\[ t^\alpha = \text{append}(g, v) \]
\[ \pi = \{ \text{append} \mapsto \{1\}, \text{reverse} \mapsto \{1\} \} \]

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

The argument filtering processor returns:

**Dependency pairs:**
\[
\begin{align*}
\text{APPEND}(\text{cons}(x, xs)) & \rightarrow \text{APPEND}(xs) \\
\text{REVERSE}(\text{cons}(x, xs)) & \rightarrow \text{REVERSE}(xs) \\
\text{REVERSE}(\text{cons}(x, xs)) & \rightarrow \text{APPEND}(\text{reverse}(xs))
\end{align*}
\]

**Rewrite system:**
\[
\begin{align*}
\text{append}(\text{nil}) & \rightarrow y \\
\text{append}(\text{cons}(x, xs)) & \rightarrow \text{cons}(x, \text{append}(xs)) \\
\text{reverse}(\text{nil}) & \rightarrow \text{nil} \\
\text{reverse}(\text{cons}(x, xs)) & \rightarrow \text{append}(\text{reverse}(xs))
\end{align*}
\]

**Argument filtering:**
\[ id = \{ \text{append} \mapsto \{1\}, \text{reverse} \mapsto \{1\} \} \]
Example

\[ t^\alpha = \text{append}(g, v) \]
\[ \pi = \{\text{append} \mapsto \{1\}, \ \text{reverse} \mapsto \{1\}\} \]  
\[ (\pi \text{ is safe for } t^\alpha) \]

The argument filtering processor returns:

**Dependency pairs:**
\[
\begin{align*}
\text{APPEND}(\text{cons}(x, xs)) & \rightarrow \text{APPEND}(xs) \\
\text{REVERSE}(\text{cons}(x, xs)) & \rightarrow \text{REVERSE}(xs) \\
\text{REVERSE}(\text{cons}(x, xs)) & \rightarrow \text{APPEND}(\text{reverse}(xs))
\end{align*}
\]

**Rewrite system:**
\[
\begin{align*}
\text{append}(\text{nil}) & \rightarrow y \quad \text{PROBLEM!} \\
\text{append}(\text{cons}(x, xs)) & \rightarrow \text{cons}(x, \text{append}(xs)) \\
\text{reverse}(\text{nil}) & \rightarrow \text{nil} \\
\text{reverse}(\text{cons}(x, xs)) & \rightarrow \text{append}(\text{reverse}(xs))
\end{align*}
\]

**Argument filtering:**
\[ id = \{\text{append} \mapsto \{1\}, \ \text{reverse} \mapsto \{1\}\}\]
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 \( s_1 \rightarrow t_1, s_2 \rightarrow t_2, \ldots \) is a chain in \( \mathcal{R} \)
  then \( \pi(s_1) \rightarrow \pi(t_1), \pi(s_2) \rightarrow \pi(t_2), \ldots \) should be a chain in \( \pi(\mathcal{R}) \)

- For this purpose, it suffices to consider extra-vars in those functions
  - that are reachable from \( t^\alpha \) 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 \( \bot \))
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 $s_1 \rightarrow t_1, s_2 \rightarrow t_2, \ldots$ is a chain in $\mathcal{R}$
  then $\pi(s_1) \rightarrow \pi(t_1), \pi(s_2) \rightarrow \pi(t_2), \ldots$ should be a chain in $\pi(\mathcal{R})$

- For this purpose, it suffices to consider extra-vars in those functions that are reachable from $t^\alpha$ 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 $_\bot$).
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 $s_1 \rightarrow t_1, s_2 \rightarrow t_2, \ldots$ is a chain in $\mathcal{R}$
  - then $\pi(s_1) \rightarrow \pi(t_1), \pi(s_2) \rightarrow \pi(t_2), \ldots$ should be a chain in $\pi(\mathcal{R})$

- For this purpose, it suffices to consider extra-vars in those functions
  - that are **reachable** from $t^\alpha$ 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 ⊥)*
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 $s_1 \rightarrow t_1$, $s_2 \rightarrow t_2$, ... is a chain in $\mathcal{R}$
  
  then $\pi(s_1) \rightarrow \pi(t_1)$, $\pi(s_2) \rightarrow \pi(t_2)$, ... should be a chain in $\pi(\mathcal{R})$

- For this purpose, it suffices to consider extra-vars in those functions
  
  - that are *reachable* from $t^\alpha$ 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 $\bot$)
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:

\textit{Dependency pairs:}
\{ 
\begin{align*}
\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{align*}
\}

\textit{Rewrite system:}
\{ 
\begin{align*}
\text{append}(\text{nil}) & \rightarrow \bot \\
\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{align*}
\}

\textit{Argument filtering:} \quad 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 $\pi(l) \rightarrow \pi(r)$ and
- an additional rule $\pi(l) \rightarrow \pi(t)$, 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** $\text{AFT}_\pi(\mathcal{R})$

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

- a filtered rule $\pi(l) \rightarrow \pi(r)$ and
- an additional rule $\pi(l) \rightarrow \pi(t)$, 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 $\pi$ be a safe argument filtering for $t^\alpha$ in $\mathcal{R}$

$\gamma(t^\alpha)$ is $\sim_{\mathcal{R}}$-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

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))

\[ t^\alpha = \text{append}(g, v) \]
\[ \pi = \{\text{append} \mapsto \{1\}, \text{reverse} \mapsto \{1\}\} \]

(\(\pi\) is safe for \(t^\alpha\))

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

append(nil) → y \hspace{1cm} (y is an extra variable)
append(cons(x, xs)) → cons(x, append(xs))
reverse(nil) → nil
reverse(cons(x, xs)) → append(reverse(xs))

which is clearly not terminating
Example

\[
\begin{align*}
\text{append}(\text{nil}, y) & \rightarrow y \\
\text{append}(\text{cons}(x, xs), y) & \rightarrow \text{cons}(x, \text{append}(xs, y)) \\
\text{reverse}(\text{nil}) & \rightarrow \text{nil} \\
\text{reverse}(\text{cons}(x, xs)) & \rightarrow \text{append}(\text{reverse}(xs), \text{cons}(x, \text{nil}))
\end{align*}
\]

\[
t^\alpha = \text{append}(g, v)
\]
\[
\pi = \{\text{append} \mapsto \{1\}, \; \text{reverse} \mapsto \{1\}\}
\]

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

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

\[
\begin{align*}
\text{append}(\text{nil}) & \rightarrow y \perp \\
\text{append}(\text{cons}(x, xs)) & \rightarrow \text{cons}(x, \text{append}(xs)) \\
\text{reverse}(\text{nil}) & \rightarrow \text{nil} \\
\text{reverse}(\text{cons}(x, xs)) & \rightarrow \text{append}(\text{reverse}(xs))
\end{align*}
\]

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 $\text{AFT}_\pi$

Website: [http://german.dsic.upv.es/filtering.html](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 ground / possibly variable

\[
\begin{align*}
g \sqcup g &= g & g \sqcup v &= v & v \sqcup g &= v & v \sqcup v &= v \\
(g, v, g) \sqcup (g, g, v) &= (g, v, v) \\
\{f \mapsto (g, v), \ g \mapsto (g, v)\} \sqcup \{f \mapsto (g, g), \ g \mapsto (v, g)\} &= \{f \mapsto (g, v), \ g \mapsto (v, v)\}
\end{align*}
\]

- **binding-time environment**: a substitution mapping variables to binding-times

- **division**: a mapping \(f/n \mapsto (m_1, \ldots, m_n)\) for every defined function, where each \(m_i\) is a binding-time
Auxiliary functions

\[ B_v[[x]] \ g/n \ \rho = (g, \ldots, g) \quad \text{(if } x \in \mathcal{V}) \]
\[ B_v[[c(t_1, \ldots, t_n)]] \ g/n \ \rho = B_v[[t_1]] \ g/n \ \rho \sqcup \ldots \sqcup B_v[[t_n]] \ g/n \ \rho \quad \text{(if } c \in \mathcal{C}) \]
\[ B_v[[f(t_1, \ldots, t_n)]] \ g/n \ \rho = \textbf{bt} \sqcup (B_e[[t_1]] \ \rho, \ldots, B_e[[t_n]] \ \rho) \quad \text{(if } f = g, \ f \in \mathcal{D}) \]
\[ \quad \textbf{bt} \quad \text{(if } f \neq g, \ f \in \mathcal{D}) \]
\[ \quad \text{where } \textbf{bt} = B_v[[t_1]] \ g/n \ \rho \sqcup \ldots \sqcup B_v[[t_n]] \ g/n \ \rho \]

\[ B_e[[x]] \ \rho = x \rho \quad \text{(if } x \in \mathcal{V}) \]
\[ B_e[[h(t_1, \ldots, t_n)]] \ \rho = B_e[[t_1]] \ \rho \sqcup \ldots \sqcup B_e[[t_n]] \ \rho \quad \text{(if } h \in \mathcal{C} \cup \mathcal{D}) \]

Roughly speaking,

- \((B_v[[t]] \ g/n \ \rho)\) 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 \(\rho\)
- \((B_e[[t]] \ \rho)\) then returns \(g\) if \(t\) contains no variable which is bound to \(v\) in \(\rho\), and \(v\) otherwise
Auxiliary functions

\[ B_v[[x]] \ g/n \ \rho = (g, \ldots, g) \quad (\text{if } x \in V) \]

\[ B_v[[c(t_1, \ldots, t_n)]] \ g/n \ \rho = B_v[[t_1]] \ g/n \ \rho \sqcup \ldots \sqcup B_v[[t_n]] \ g/n \ \rho \quad (\text{if } c \in C) \]

\[ B_v[[f(t_1, \ldots, t_n)]] \ g/n \ \rho = bt \sqcup (B_e[[t_1]] \ \rho, \ldots, B_e[[t_n]] \ \rho) \quad (\text{if } f = g, \ f \in D) \]

\[ bt = B_v[[t_1]] \ g/n \ \rho \sqcup \ldots \sqcup B_v[[t_n]] \ g/n \ \rho \quad (\text{if } f \neq g, \ f \in D) \]

where \( bt = B_v[[t_1]] \ g/n \ \rho \sqcup \ldots \sqcup B_v[[t_n]] \ g/n \ \rho \)

\[ B_e[[x]] \ \rho = x \ \rho \quad (\text{if } x \in V) \]

\[ B_e[[h(t_1, \ldots, t_n)]] \ \rho = B_e[[t_1]] \ \rho \sqcup \ldots \sqcup B_e[[t_n]] \ \rho \quad (\text{if } h \in C \cup D) \]

Roughly speaking,

- \((B_v[[t]] \ g/n \ \rho)\) 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 \(\rho\)

- \((B_e[[t]] \ \rho)\) then returns \(g\) if \(t\) contains no variable which is bound to \(v\) in \(\rho\), and \(v\) otherwise
### Auxiliary functions

\[ B_v[[x]] \frac{g}{n} \rho = (g, \ldots, g) \]  
\[ B_v[[c(t_1, \ldots, t_n)]] \frac{g}{n} \rho = B_v[[t_1]] \frac{g}{n} \rho \sqcup \ldots \sqcup B_v[[t_n]] \frac{g}{n} \rho \]  
\[ B_v[[f(t_1, \ldots, t_n)]] \frac{g}{n} \rho = \text{bt} \sqcup (B_e[[t_1]] \rho, \ldots, B_e[[t_n]] \rho) \]  
\[ B_e[[x]] \rho = x\rho \]  
\[ B_e[[h(t_1, \ldots, t_n)]] \rho = B_e[[t_1]] \rho \sqcup \ldots \sqcup B_e[[t_n]] \rho \]

where \( \text{bt} = B_v[[t_1]] \frac{g}{n} \rho \sqcup \ldots \sqcup B_v[[t_n]] \frac{g}{n} \rho \)

Roughly speaking,

- \((B_v[[t]] \frac{g}{n} \rho)\) 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 \(\rho\).
- \((B_e[[t]] \rho)\) then returns \(g\) if \(t\) contains no variable which is bound to \(v\) in \(\rho\), and \(v\) otherwise.
BTA algorithm

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

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

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

Iterative process

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

$$\downarrow$$

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

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

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

$$\cup \ldots$$

$$\cup \{ x \mapsto m_n \mid x \in \text{Var}(t_n) \}$$
Given an abstract term $f_1(m_1, \ldots, m_{n_1})$, the initial division is

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

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

**Iterative process**

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

$$\downarrow$$

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

$$\ldots,$$

$$f_k \mapsto b_k \sqcup B_v[[r_1]] f_k/n_k e(b_1, l_1) \sqcup \ldots \sqcup B_v[[r_j]] f_k/n_k e(b_j, l_j) \}$$

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

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

$$\cup \ldots$$

$$\cup \{ x \mapsto m_n \mid x \in \text{Var}(t_n) \}$$
When \( \text{div}_i = \text{div}_{i+1} \) (fixpoint), the corresponding safe argument filtering \( \pi \) is obtained as follows:

Given the division

\[
\text{div} = \{ f_1 \leftrightarrow (m^1_1, \ldots, m^1_{n_1}), \ldots, f_k \leftrightarrow (m^k_1, \ldots, m^k_{n_k}) \}
\]

we have

\[
\pi(\text{div}) = \{ f_1 \leftrightarrow \{ i \mid m^1_i = g \}, \ldots, f_k \leftrightarrow \{ i \mid m^k_i = g \} \}
\]

\( \pi(\text{div}) \) is a safe argument filtering since the computed division \( \text{div} \) is congruent [JGS93]
Example

\[
\begin{align*}
\text{mult}(z, y) & \rightarrow z \\
\text{mult}(s(x), y) & \rightarrow \text{add}(\text{mult}(x, y), y) \\
\text{add}(z, y) & \rightarrow y \\
\text{add}(s(x), y) & \rightarrow s(\text{add}(x, y))
\end{align*}
\]

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

\[
div_0 = \{ \text{mult} \mapsto (g, v), \text{add} \mapsto (g, g) \} \]

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

\[
div_1 = \{ \text{mult} \mapsto (g, v) \} \cup \begin{cases} 
\text{add} \mapsto (g, g) \\
B_v[[z]] \text{ mult/2 } \{ y \mapsto v \} \\
B_v[[\text{add}(\text{mult}(x, y), y)]] \text{ mult/2 } \{ x \mapsto g, y \mapsto v \} \\
B_v[[y]] \text{ mult/2 } \{ y \mapsto g \} \\
B_v[[s(\text{add}(x, y))]] \text{ mult/2 } \{ x \mapsto g, y \mapsto g \}, \\
B_v[[z]] \text{ add/2 } \{ y \mapsto v \} \\
B_v[[\text{add}(\text{mult}(x, y), y)]] \text{ add/2 } \{ x \mapsto g, y \mapsto v \} \\
B_v[[y]] \text{ add/2 } \{ y \mapsto g \} \\
B_v[[s(\text{add}(x, y))]] \text{ add/2 } \{ x \mapsto g, y \mapsto g \} \\
\end{cases}
\]
Example

\[
\begin{align*}
\text{mult}(z, y) & \rightarrow z \\
\text{mult}(s(x), y) & \rightarrow \text{add(\text{mult}(x, y), y)} \\
\text{add}(z, y) & \rightarrow y \\
\text{add}(s(x), y) & \rightarrow \text{s(\text{add}(x, y))}
\end{align*}
\]

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

\[
div_0 = \{ \text{mult} \mapsto (g, v), \; \text{add} \mapsto (g, g) \}
\]

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

\[
div_1 = \{ \text{mult} \mapsto (g, v), \; \text{add} \mapsto (g, g) \}
\]

- \(B_v[[z]] \text{ mult/2 } \{ y \mapsto v \}\)
- \(B_v[[\text{add(\text{mult}(x, y), y)}]] \text{ mult/2 } \{ x \mapsto g, \ y \mapsto v \}\)
- \(B_v[[y]] \text{ mult/2 } \{ y \mapsto g \}\)
- \(B_v[[\text{s(\text{add}(x, y))}]] \text{ mult/2 } \{ x \mapsto g, \ y \mapsto g \}\)
- \(B_v[[z]] \text{ add/2 } \{ y \mapsto v \}\)
- \(B_v[[\text{add(\text{mult}(x, y), y)}]] \text{ add/2 } \{ x \mapsto g, \ y \mapsto v \}\)
- \(B_v[[y]] \text{ add/2 } \{ y \mapsto g \}\)
- \(B_v[[\text{s(\text{add}(x, y))}]] \text{ add/2 } \{ x \mapsto g, \ y \mapsto g \}\)
Example (cont’d)

Therefore, by evaluating the calls to $B_v$, we get

$$\text{div}_1 = \{ \text{mult} \mapsto (g, v), \text{add} \mapsto (v, v) \}$$

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 div₂ we get div₁ = div₂  \implies \text{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

$$\text{div}_1 = \{\text{mult} \mapsto (g, v), \text{add} \mapsto (v, v)\}$$

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 $\text{div}_2$ we get $\text{div}_1 = \text{div}_2 \implies \text{div}_1$ 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

$$\text{div}_1 = \{ \text{mult} \mapsto (g, v), \text{add} \mapsto (v, v) \}$$

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 $\text{div}_2$ we get $\text{div}_1 = \text{div}_2 \implies \text{div}_1$ 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.,

\[\begin{align*}
\text{eq}(z, z) & \rightarrow \text{true} \\
\text{eq}(s(x), s(y)) & \rightarrow \text{eq}(x, y)
\end{align*}\]

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 \( T^\alpha \) 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.,

- \( \text{eq}(z, z) \rightarrow \text{true} \)
- \( \text{eq}(s(x), s(y)) \rightarrow \text{eq}(x, y) \)

and the set

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

Here, starting from

\[
div_0 = \{ \text{eq} \mapsto (g, v) \sqcup (v, g) \} = \{ \text{eq} \mapsto (v, v) \}
\]

is not a good idea . . .

**Solution**

**Lemma**

Let \( \mathcal{R} \) be a TRS and \( T^\alpha \) be a finite set of abstract terms. \( \gamma(T^\alpha) \) is \( \rightarrow_{\mathcal{R}} \)-terminating iff \( \gamma(t^\alpha) \) is \( \rightarrow_{\mathcal{R}} \)-terminating for all \( t^\alpha \in T^\alpha \).
Some refinements (cont’d)

**Non well-moded programs**

Consider, e.g.,

\[
\begin{align*}
eq(z, z) & \rightarrow \text{true} \\
eq(s(x), s(y)) & \rightarrow \text{eq}(y, x)
\end{align*}
\]

If we start with

\[\text{eq}(g, v)\]

the only safe argument filtering is

\[\pi = \{ \text{eq} \mapsto \{ \} \}\]

**Solution**

\[
\begin{align*}
eq_{gg}(z, z) & \rightarrow \text{true} & \quad \text{eq}_{gg}(z, z) & \rightarrow \text{true} \\
eq_{gg}(s(x), s(y)) & \rightarrow \text{eq}_{gg}(y, x) & \quad \text{eq}_{gv}(s(x), s(y)) & \rightarrow \text{eq}_{vg}(y, x) \\
eq_{vg}(z, z) & \rightarrow \text{true} & \quad \text{eq}_{vv}(z, z) & \rightarrow \text{true} \\
eq_{vg}(s(x), s(y)) & \rightarrow \text{eq}_{vg}(y, x) & \quad \text{eq}_{vv}(s(x), s(y)) & \rightarrow \text{eq}_{vv}(y, x)
\end{align*}
\]
Some refinements (cont’d)

Non well-moded programs

Consider, e.g.,

\[
\begin{align*}
\text{eq}(z, z) & \rightarrow \text{true} \\
\text{eq}(s(x), s(y)) & \rightarrow \text{eq}(y, x)
\end{align*}
\]

If we start with

\[
\text{eq}(g, v)
\]

the only safe argument filtering is

\[
\pi = \{ \text{eq} \mapsto \{ \} \}
\]

Solution

\[
\begin{align*}
\text{eq}_{gg}(z, z) & \rightarrow \text{true} & \text{eq}_{gg}(z, z) & \rightarrow \text{true} \\
\text{eq}_{gg}(s(x), s(y)) & \rightarrow \text{eq}_{gg}(y, x) & \text{eq}_{gv}(s(x), s(y)) & \rightarrow \text{eq}_{vg}(y, x) \\
\text{eq}_{vg}(z, z) & \rightarrow \text{true} & \text{eq}_{vv}(z, z) & \rightarrow \text{true} \\
\text{eq}_{vg}(s(x), s(y)) & \rightarrow \text{eq}_{gg}(y, x) & \text{eq}_{vv}(s(x), s(y)) & \rightarrow \text{eq}_{vv}(y, x)
\end{align*}
\]
Removing non-reachable functions

Consider, e.g.,

\[
\begin{align*}
    a & \rightarrow a \\
    b & \rightarrow c \\
    c & \rightarrow d
\end{align*}
\]

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.,

\[
\begin{align*}
\text{a} & \rightarrow \text{a} \\
\text{b} & \rightarrow \text{c} \\
\text{c} & \rightarrow \text{d}
\end{align*}
\]

Although narrowing terminates for the abstract term \( \text{b} \),
we get the argument filtering

\[
\pi = \{ \text{a} \mapsto \{ \}, \text{b} \mapsto \{ \}, \text{c} \mapsto \{ \} \}
\]

and then we fail to prove its termination...

**Solution**

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

- 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

- 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.  

J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke.  
Mechanizing and Improving Dependency Pairs.  

N.D. Jones, C.K. Gomard, and P. Sestoft.  
*Partial Evaluation and Automatic Program Generation*.  

N. Nishida and K. Miura.  
Dependency Graph Method for Proving Termination of Narrowing.  

Narrowing-Based Simulation of Term Rewriting Systems with Extra Variables.


P. Schneider-Kamp, J. Giesl, A. Serebrenik, and R. Thiemann. Automated Termination Analysis for Logic Programs by Term Rewriting.