Termination and Complexity in Higher-Order Term Rewriting

Part 4. Termination: modular termination proofs using dependency pairs

Cynthia Kop

ISR 2024

Motivation

Goal:

We want to prove termination of large higher-order term rewriting systems.

Motivation

Goal:

We want to prove termination of large higher-order term rewriting systems.

Secondary goal:

We want to prove termination properties of part of a higher-order TRS.

Running example

$$
\begin{array}{rcl}
\mathtt{I}(x) & \to & x \\
\mathtt{minus}(x, \mathtt{0}) & \to & x \\
\mathtt{minus}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{minus}(x, y) \\
\mathtt{quot}(\mathtt{0}, \mathtt{s}(y)) & \to & \mathtt{0} \\
\mathtt{quot}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{s}(\mathtt{quot}(\mathtt{minus}(x, y), \mathtt{s}(y))) \\
\mathtt{ack}(\mathtt{0}, y) & \to & \mathtt{s}(y) \\
\mathtt{ack}(\mathtt{s}(x), \mathtt{0}) & \to & \mathtt{ack}(x, \mathtt{s}(\mathtt{0})) \\
\mathtt{ack}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{ack}(x, \mathtt{ack}(\mathtt{s}(x), y)) \\
\mathtt{inc}(\mathtt{0}) & \to & \mathtt{s}(\mathtt{inc}(\mathtt{s}(\mathtt{0}))) \\
\mathtt{fexp}(\mathtt{0}, y) & \to & y \\
\mathtt{fexp}(\mathtt{s}(x), y) & \to & \mathtt{double}(x, y, \mathtt{0}) \\
\mathtt{double}(x, \mathtt{0}, z) & \to & \mathtt{fexp}(x, z) \\
\mathtt{double}(x, \mathtt{s}(y), z) & \to & \mathtt{double}(x, y, \mathtt{s}(\mathtt{s}(z))) \\
\mathtt{hd}(\mathtt{cons}(x, l)) & \to & x \\
\mathtt{len}([]) & \to & \mathtt{0} \\
\mathtt{len}(\mathtt{cons}(x, l)) & \to & \mathtt{s}(\mathtt{len}(l)) \\
\mathtt{map}(F, []) & \to & [] \\
\mathtt{map}(F, \mathtt{cons}(x, l)) & \to & \mathtt{cons}(F \cdot x, \mathtt{map}(F, l)) \\
\mathtt{fold}(F, x, []) & \to & x \\
\mathtt{fold}(F, x, \mathtt{cons}(y, l)) & \to & \mathtt{fold}(F, F \cdot x \cdot y, l) \\
\mathtt{mkbig}(l, x) & \to & \mathtt{map}(\mathtt{ack}(x), l) \\
\mathtt{mkdiv}(l, x) & \to & \mathtt{map}(\lambda y.\mathtt{quot}(y, x), l) \\
\mathtt{sma}(b, F, \mathtt{0}) & \to & \mathtt{0} \\
\mathtt{sma}(\mathtt{true}, F, \mathtt{s}(x)) & \to & \mathtt{s}(x) \\
\mathtt{sma}(\mathtt{false}, F, \mathtt{s}(x)) & \to & \mathtt{sma}(F \cdot x, F, \mathtt{quot}(x, \mathtt{s}\ (\mathtt{s}\ \mathtt{0}))) \\
\mathtt{twice}(F, x) & \to & F \cdot (F \cdot x) \\
\mathtt{H}(\mathtt{s}(x)) & \to & \mathtt{H}(\mathtt{twice}(\mathtt{I}, x))
\end{array}
$$

Modularity

Ideal situation:

- split $\mathcal{R}$ into $\mathcal{R} = A \cup B$ (signatures share only constructors)
- prove termination of A and B separately
- conclude termination of $\mathcal{R}$

Modularity

Ideal situation:

- split $\mathcal{R}$ into $\mathcal{R} = A \cup B$ (signatures share only constructors)
- prove termination of A and B separately
- conclude termination of $\mathcal{R}$

Modularity

Ideal situation:

- split $\mathcal{R}$ into $\mathcal{R} = A \cup B$ (signatures share only constructors)
- prove termination of A and B separately
- conclude termination of $\mathcal{R}$

Toyama's counterexample:

- $A = \{ \mathtt{f}(\mathtt{a}, \mathtt{b}, x) \to \mathtt{f}(x, x, x) \}$
- $B = \{ \pi(x, y) \to x \; ; \pi(x, y) \to y \}$
- non-termination of $A \cup B$ due to $\mathtt{f}(\mathtt{a}, \mathtt{b}, \pi(\mathtt{a}, \mathtt{b}))$

Modularity

Pretty good situation:

- split $\mathcal{R}$ into $\mathcal{R} = A \cup B$ (signatures share only constructors)
- prove termination of $A \cup \mathcal{C}_\epsilon$ and $B \cup \mathcal{C}_\epsilon$ separately (here, $\mathcal{C}_\epsilon = \{ \pi\ x\ y \to x\ ;\ \pi\ x\ y \to y \}$)
- conclude termination of $\mathcal{R} \cup \mathcal{C}_\epsilon$

Modularity

Pretty good situation:

- split $\mathcal{R}$ into $\mathcal{R} = A \cup B$ (signatures share only constructors)
- prove termination of $A \cup \mathcal{C}_\epsilon$ and $B \cup \mathcal{C}_\epsilon$ separately
(here, $\mathcal{C}_\epsilon = \{\ \pi\ x\ y \to x\ ;\ \pi\ x\ y \to y\ \}$)
- conclude termination of $\mathcal{R} \cup \mathcal{C}_\epsilon$

Modularity

~~Pretty good situation:~~

- ~~split $\mathcal{R}$ into $\mathcal{R} = A \cup B$ (signatures share only constructors)~~
- ~~prove termination of $A \cup \mathcal{C}_\epsilon$ and $B \cup \mathcal{C}_\epsilon$ separately (here, $\mathcal{C}_\epsilon = \{\ \pi\ x\ y \to x\ ;\ \pi\ x\ y \to y\ \}$)~~
- ~~conclude termination of $\mathcal{R} \cup \mathcal{C}_\epsilon$~~

My counterexample:

$$A = \left\{ \begin{array}{rcl} \mathtt{comp2}(\mathtt{0}, \mathtt{s}(y)) & \to & \mathtt{false} \\ \mathtt{comp2}(\mathtt{s}(\mathtt{0}), \mathtt{s}(y)) & \to & \mathtt{false} \\ \mathtt{comp2}(x, \mathtt{0}) & \to & \mathtt{true} \\ \mathtt{comp2}(\mathtt{s}(\mathtt{s}(x)), \mathtt{s}(y)) & \to & \mathtt{comp2}(x, y) \\ \mathtt{find}(F, x, \mathtt{false}) & \to & \mathtt{end}(x) \\ \mathtt{find}(F, x, \mathtt{true}) & \to & \mathtt{find}(F, \mathtt{s}(x), \mathtt{comp2}(F \cdot x, x)) \end{array} \right\}$$

$$B = \{\ \mathtt{double}(\mathtt{0}) \to \mathtt{0} \qquad \mathtt{double}(\mathtt{s}(x)) \to \mathtt{s}(\mathtt{s}(\mathtt{double}(x)))\ \}$$

Modularity

Pretty good situation:

- split $\mathcal{R}$ into $\mathcal{R} = A \cup B$ (signatures share only constructors)
- prove termination of $A \cup \mathcal{C}_\epsilon$ and $B \cup \mathcal{C}_\epsilon$ separately (here, $\mathcal{C}_\epsilon = \{ \pi\ x\ y \to x\ ;\ \pi\ x\ y \to y \}$)
- conclude termination of $\mathcal{R} \cup \mathcal{C}_\epsilon$

My counterexample:

$$A = \left\{ \begin{array}{rcl} \mathtt{comp2}(x, y) & \to & \text{“if } x \geq 2y \\ & & \text{ then } \mathtt{true} \\ & & \text{ else } \mathtt{false}\text{”} \\ \\ \mathtt{find}(F, x, \mathtt{false}) & \to & \mathtt{end}(x) \\ \mathtt{find}(F, x, \mathtt{true}) & \to & \mathtt{find}(F, \mathtt{s}(x), \mathtt{comp2}(F \cdot x, x)) \end{array} \right\}$$

$$B = \{ \mathtt{double}(\mathtt{0}) \to \mathtt{0} \qquad \mathtt{double}(\mathtt{s}(x)) \to \mathtt{s}(\mathtt{s}(\mathtt{double}(x))) \}$$

Higher-order Modularity is hard!

Appel, Oostrom, Simonsen (2010):

Almost no modularity properties hold for higher-order rewriting!
(Even when they do hold for first-order rewriting.)

Higher-order Modularity is hard!

Appel, Oostrom, Simonsen (2010):

Almost no modularity properties hold for higher-order rewriting!
(Even when they do hold for first-order rewriting.)

Property	TRS	STTRS	CRS	PRS
Confluence	Yes	No	No	No
Normalization	Yes	No (†)	No (†)	No (†)
Termination	No	No	No	No
Completeness	No	No	No	No
Confluence, for left-linear systems	Yes	Yes	Yes	Yes
Completeness, for left-linear systems	Yes	No (†)	No (†)	No (†)
Unique normal forms	Yes	No (†)	No (†)	No (†)
Normalization, non-duplicating pattern systems	Yes	Yes (†)	?	?
Termination, non-duplicating pattern systems	Yes	Yes (†)	?	No (†)

Dependency Pairs

Idea:

- isolate function calls in reduction rules
- determine groups of recursive calls
- prove for each group of recursive calls that it doesn't lead to an infinite loop

Dependency Pairs

Idea:

- isolate function calls in reduction rules
- determine groups of recursive calls
- prove for each group of recursive calls that it doesn't lead to an infinite loop

```
le(0, x) => true
le(s(x), 0) => false
le(s(x), s(y)) => le(x, y)
eq(0, 0) => true
eq(0, s(x)) => false
eq(s(x), 0) => false
eq(s(x), s(y)) => eq(x, y)
if(true, x, y) => x
if(false, x, y) => y
minsort(nil) => nil
minsort(cons(x, y)) => cons(min(x, y), minsort(del(min(x, y), cons(x, y))))
min(x, nil) => x
min(x, cons(y, z)) => if(le(x, y), min(x, z), min(y, z))
del(x, nil) => nil
del(x, cons(y, z)) => if(eq(x, y), z, cons(y, del(x, z)))
map(f, nil) => nil
map(f, cons(x, y)) => cons(f x, map(f, y))
filter(f, nil) => nil
filter(f, cons(x, y)) => filter2(f x, f, x, y)
filter2(true, f, x, y) => cons(x, filter(f, y))
filter2(false, f, x, y) => filter(f, y)
```

Dependency Pairs

Idea:

- isolate function calls in reduction rules
- determine groups of recursive calls
- prove for each group of recursive calls that it doesn't lead to an infinite loop

```
0] le#(s(X), s(Y)) =#> le#(X, Y)
1] eq#(s(X), s(Y)) =#> eq#(X, Y)
2] minsort#(cons(X, Y)) =#> min#(X, Y)
3] minsort#(cons(X, Y)) =#> minsort#(del(min(X, Y), cons(X, Y)))
4] minsort#(cons(X, Y)) =#> del#(min(X, Y), cons(X, Y))
5] minsort#(cons(X, Y)) =#> min#(X, Y)
6] min#(X, cons(Y, Z)) =#> if#(le(X, Y), min(X, Z), min(Y, Z))
7] min#(X, cons(Y, Z)) =#> le#(X, Y)
8] min#(X, cons(Y, Z)) =#> min#(X, Z)
9] min#(X, cons(Y, Z)) =#> min#(Y, Z)
10] del#(X, cons(Y, Z)) =#> if#(eq(X, Y), Z, cons(Y, del(X, Z)))
11] del#(X, cons(Y, Z)) =#> eq#(X, Y)
12] del#(X, cons(Y, Z)) =#> del#(X, Z)
13] map#(F, cons(X, Y)) =#> map#(F, Y)
14] filter#(F, cons(X, Y)) =#> filter2#(F X, F, X, Y)
15] filter2#(true, F, X, Y) =#> filter#(F, Y)
16] filter2#(false, F, X, Y) =#> filter#(F, Y)
```

Dependency Pairs

Idea:

- isolate function calls in reduction rules
- determine groups of recursive calls
- prove for each group of recursive calls that it doesn't lead to an infinite loop

```
0] le#(s(X), s(Y)) =#> le#(X, Y)

1] eq#(s(X), s(Y)) =#> eq#(X, Y)

3] minsort#(cons(X, Y)) =#> minsort#(del(min(X, Y), cons(X, Y)))

8] min#(X, cons(Y, Z)) =#> min#(X, Z)
9] min#(X, cons(Y, Z)) =#> min#(Y, Z)

12] del#(X, cons(Y, Z)) =#> del#(X, Z)

13] map#(F, cons(X, Y)) =#> map#(F, Y)

14] filter#(F, cons(X, Y)) =#> filter2#(F X, F, X, Y)
15] filter2#(true, F, X, Y) =#> filter#(F, Y)
16] filter2#(false, F, X, Y) =#> filter#(F, Y)
```

Dependency Pairs

Idea:

- isolate function calls in reduction rules
- determine groups of recursive calls
- prove for each group of recursive calls that it doesn't lead to an infinite loop

Practice:

- "dependency pair" ≈ "function call"
- "dependency pair problem" ≈ "group of calls"

Dependency Pairs

Idea:

- isolate function calls in reduction rules
- determine groups of recursive calls
- prove for each group of recursive calls that it doesn't lead to an infinite loop

Practice:

- "dependency pair" $\approx$ "function call"
- "dependency pair problem" $\approx$ "group of calls"
- each dependency pair problem can be finite or infinite:

Dependency Pairs

Idea:

- isolate function calls in reduction rules
- determine groups of recursive calls
- prove for each group of recursive calls that it doesn't lead to an infinite loop

Practice:

- "dependency pair" $\approx$ "function call"
- "dependency pair problem" $\approx$ "group of calls"
- each dependency pair problem can be finite or infinite:
  - finite: harmless; this group of calls does not lead to non-termination

Dependency Pairs

Idea:

- isolate function calls in reduction rules
- determine groups of recursive calls
- prove for each group of recursive calls that it doesn't lead to an infinite loop

Practice:

- "dependency pair" $\approx$ "function call"
- "dependency pair problem" $\approx$ "group of calls"
- each dependency pair problem can be finite or infinite:
  - finite: harmless; this group of calls does not lead to non-termination
  - infinite: harmful: this group of calls *does* lead to non-termination

First-order dependency pairs

$$\begin{aligned}
\mathtt{minus}(x, \mathtt{0}) &\rightarrow x \\
\mathtt{minus}(\mathtt{s}(x), \mathtt{s}(y)) &\rightarrow \mathtt{minus}(x, y) \\
\mathtt{quot}(\mathtt{0}, \mathtt{s}(y)) &\rightarrow \mathtt{0} \\
\mathtt{quot}(\mathtt{s}(x), \mathtt{s}(y)) &\rightarrow \mathtt{s}(\mathtt{quot}(\mathtt{minus}(x, y), \mathtt{s}(y)))
\end{aligned}$$

First-order dependency pairs

$$\begin{array}{rcl} \mathtt{minus}(x, \mathtt{0}) & \to & x \\ \mathtt{minus}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{minus}(x, y) \\ \mathtt{quot}(\mathtt{0}, \mathtt{s}(y)) & \to & \mathtt{0} \\ \mathtt{quot}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{s}(\mathtt{quot}(\mathtt{minus}(x, y), \mathtt{s}(y))) \end{array}$$

Question: what is a “function call”?

First-order dependency pairs

$$\begin{array}{rcl} \mathtt{minus}(x, \mathtt{0}) & \to & x \\ \mathtt{minus}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{minus}(x, y) \\ \mathtt{quot}(\mathtt{0}, \mathtt{s}(y)) & \to & \mathtt{0} \\ \mathtt{quot}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{s}(\mathtt{quot}(\underline{\mathtt{minus}(x, y)}, \mathtt{s}(y))) \end{array}$$

Question: what is a “function call”?

First-order dependency pairs

$$\begin{array}{rcl} \mathtt{minus}(x, \mathtt{0}) & \to & x \\ \mathtt{minus}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{minus}(x, y) \\ \mathtt{quot}(\mathtt{0}, \mathtt{s}(y)) & \to & \mathtt{0} \\ \mathtt{quot}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{s}(\underline{\mathtt{quot}(\mathtt{minus}(x, y), \mathtt{s}(y))}) \end{array}$$

Question: what is a “function call”?

First-order dependency pairs

$$\begin{array}{rcl} \mathtt{minus}(x, \mathtt{0}) & \to & x \\ \mathtt{minus}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{minus}(x, y) \\ \mathtt{quot}(\mathtt{0}, \mathtt{s}(y)) & \to & \mathtt{0} \\ \mathtt{quot}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{s}(\mathtt{quot}(\mathtt{minus}(x, y), \mathtt{s}(y))) \end{array}$$

Question: what is a "function call"?

Answer: subterms whose root symbol is a defined symbol.

First-order dependency pairs

$$\begin{array}{rcl} \mathtt{minus}(x, \mathtt{0}) & \to & x \\ \mathtt{minus}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{minus}(x, y) \\ \mathtt{quot}(\mathtt{0}, \mathtt{s}(y)) & \to & \mathtt{0} \\ \mathtt{quot}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{s}(\mathtt{quot}(\mathtt{minus}(x, y), \mathtt{s}(y))) \end{array}$$

Question: what is a “function call”?

Answer: subterms whose root symbol is a $\mathtt{defined}$ symbol.

Dependency pairs:

$$\begin{array}{rcl} \mathtt{minus}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{minus}^\sharp(x, y) \\ \mathtt{quot}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{minus}^\sharp(x, y), \mathtt{s}(y)) \\ \mathtt{quot}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{quot}^\sharp(\mathtt{minus}(x, y), \mathtt{s}(y)) \end{array}$$

First-order dependency pair chain

Definition: a minimal DP chain over $(\mathcal{P}, \mathcal{R})$ is a reduction chain:

$$s_1 \rightarrow_{\mathcal{P}} t_1 \rightarrow^*_{\mathcal{R}} s_2 \rightarrow_{\mathcal{P}} t_2 \rightarrow^*_{\mathcal{R}} \dots$$

First-order dependency pair chain

Definition: a minimal DP chain over $(\mathcal{P}, \mathcal{R})$ is a reduction chain:

$$s_1 \rightarrow_{\mathcal{P}} t_1 \rightarrow^*_{\mathcal{R}} s_2 \rightarrow_{\mathcal{P}} t_2 \rightarrow^*_{\mathcal{R}} \dots$$

Such that:

First-order dependency pair chain

Definition: a minimal DP chain over $(\mathcal{P}, \mathcal{R})$ is a reduction chain:

$$s_1 \rightarrow_{\mathcal{P}} t_1 \rightarrow^*_{\mathcal{R}} s_2 \rightarrow_{\mathcal{P}} t_2 \rightarrow^*_{\mathcal{R}} \dots$$

Such that:

- each reduction $s_i \rightarrow_{\mathcal{P}} t_i$ is at the root
  (so $s_i = \ell\gamma$ and $t_i = r\gamma$ for some $\ell \rightarrow r \in \mathcal{P}$)

First-order dependency pair chain

Definition: a minimal DP chain over $(\mathcal{P}, \mathcal{R})$ is a reduction chain:

$$s_1 \rightarrow_{\mathcal{P}} t_1 \rightarrow^*_{\mathcal{R}} s_2 \rightarrow_{\mathcal{P}} t_2 \rightarrow^*_{\mathcal{R}} \dots$$

Such that:

- each reduction $s_i \rightarrow_{\mathcal{P}} t_i$ is at the root
  (so $s_i = \ell\gamma$ and $t_i = r\gamma$ for some $\ell \rightarrow r \in \mathcal{P}$)
- each reduction $s_i \rightarrow^*_{\mathcal{R}} t_i$ occurs below the root

First-order dependency pair chain

Definition: a minimal DP chain over $(\mathcal{P}, \mathcal{R})$ is a reduction chain:

$$s_1 \rightarrow_{\mathcal{P}} t_1 \rightarrow^*_{\mathcal{R}} s_2 \rightarrow_{\mathcal{P}} t_2 \rightarrow^*_{\mathcal{R}} \dots$$

Such that:

- each reduction $s_i \rightarrow_{\mathcal{P}} t_i$ is at the root (so $s_i = \ell\gamma$ and $t_i = r\gamma$ for some $\ell \rightarrow r \in \mathcal{P}$)
- each reduction $s_i \rightarrow^*_{\mathcal{R}} t_i$ occurs below the root (this is actually automatic: root symbols are constructors)

First-order dependency pair chain

Definition: a minimal DP chain over $(\mathcal{P}, \mathcal{R})$ is a reduction chain:

$$s_1 \to_{\mathcal{P}} t_1 \to_{\mathcal{R}}^* s_2 \to_{\mathcal{P}} t_2 \to_{\mathcal{R}}^* \dots$$

Such that:

- each reduction $s_i \to_{\mathcal{P}} t_i$ is at the root (so $s_i = \ell\gamma$ and $t_i = r\gamma$ for some $\ell \to r \in \mathcal{P}$)
- each reduction $s_i \to_{\mathcal{R}}^* t_i$ occurs below the root (this is actually automatic: root symbols are constructors)
- each t_i is terminating with respect to $\to_{\mathcal{R}}$

First-order dependency pair chain

Definition: a minimal DP chain over $(\mathcal{P}, \mathcal{R})$ is a reduction chain:

$$s_1 \rightarrow_{\mathcal{P}} t_1 \rightarrow_{\mathcal{R}}^* s_2 \rightarrow_{\mathcal{P}} t_2 \rightarrow_{\mathcal{R}}^* \dots$$

Such that:

- each reduction $s_i \rightarrow_{\mathcal{P}} t_i$ is at the root (so $s_i = \ell\gamma$ and $t_i = r\gamma$ for some $\ell \rightarrow r \in \mathcal{P}$)
- each reduction $s_i \rightarrow_{\mathcal{R}}^* t_i$ occurs below the root (this is actually automatic: root symbols are constructors)
- each t_i is terminating with respect to $\rightarrow_{\mathcal{R}}$

Claim:

there is an infinite minimal $(\mathtt{DP}(\mathcal{R}), \mathcal{R})$-chain
if and only if
$\rightarrow_{\mathcal{R}}$ is non-terminating

Dependency chain claim

Claim:

there is an infinite minimal $(\mathtt{DP}(\mathcal{R}), \mathcal{R})$-chain
if and only if $\rightarrow_{\mathcal{R}}$ is non-terminating

Dependency chain claim

Claim:

there is an infinite minimal $(\mathtt{DP}(\mathcal{R}), \mathcal{R})$-chain
if and only if $\rightarrow_{\mathcal{R}}$ is non-terminating

Proof:

Dependency chain claim

Claim:

there is an infinite minimal $(\mathtt{DP}(\mathcal{R}), \mathcal{R})$-chain
if and only if $\rightarrow_{\mathcal{R}}$ is non-terminating

Proof:

$\Rightarrow$ If $s \rightarrow_{\mathtt{DP}(\mathcal{R})} t$ then $|s| \rightarrow_{\mathcal{R}} \cdot \trianglerighteq |t|$.

Dependency chain claim

Claim:

there is an infinite minimal $(\mathtt{DP}(\mathcal{R}), \mathcal{R})$-chain
if and only if $\rightarrow_{\mathcal{R}}$ is non-terminating

Proof:

$\Rightarrow$ If $s \rightarrow_{\mathtt{DP}(\mathcal{R})} t$ then $|s| \rightarrow_{\mathcal{R}} \cdot \trianglerighteq |t|$.

$\Leftarrow$ If $\rightarrow_{\mathcal{R}}$ is non-terminating, there is a minimal non-terminating term s.

Dependency chain claim

Claim:

there is an infinite minimal $(\mathtt{DP}(\mathcal{R}), \mathcal{R})$-chain
if and only if $\rightarrow_{\mathcal{R}}$ is non-terminating

Proof:

$\Rightarrow$ If $s \rightarrow_{\mathtt{DP}(\mathcal{R})} t$ then $|s| \rightarrow_{\mathcal{R}} \cdot \trianglerighteq |t|$.

$\Leftarrow$ If $\rightarrow_{\mathcal{R}}$ is non-terminating, there is a minimal non-terminating term s.
Hence, there is an infinite reduction

$$s = \mathtt{f}(s_1, \ldots, s_k) \rightarrow^*_{\mathcal{R},in} \mathtt{f}(s'_1, \ldots, s'_k) = \ell\gamma \rightarrow_{\mathcal{R}} r\gamma \rightarrow_{\mathcal{R}} \ldots$$

Dependency chain claim

Claim:

there is an infinite minimal $(\mathtt{DP}(\mathcal{R}), \mathcal{R})$-chain
if and only if $\to_{\mathcal{R}}$ is non-terminating

Proof:

$\Rightarrow$ If $s \to_{\mathtt{DP}(\mathcal{R})} t$ then $|s| \to_{\mathcal{R}} \cdot \trianglerighteq |t|$.

$\Leftarrow$ If $\to_{\mathcal{R}}$ is non-terminating, there is a minimal non-terminating term s.
Hence, there is an infinite reduction

$$s = \mathtt{f}(s_1, \dots, s_k) \to^*_{\mathcal{R},in} \mathtt{f}(s'_1, \dots, s'_k) = \ell\gamma \to_{\mathcal{R}} r\gamma \to_{\mathcal{R}} \dots$$

Let p be the smallest subterm of r such that $p\gamma$ is non-terminating.

Dependency chain claim

Claim:

there is an infinite minimal $(\mathtt{DP}(\mathcal{R}), \mathcal{R})$-chain
if and only if $\rightarrow_{\mathcal{R}}$ is non-terminating

Proof:

$\Rightarrow$ If $s \rightarrow_{\mathtt{DP}(\mathcal{R})} t$ then $|s| \rightarrow_{\mathcal{R}} \cdot \trianglerighteq |t|$.

$\Leftarrow$ If $\rightarrow_{\mathcal{R}}$ is non-terminating, there is a minimal non-terminating term s.
Hence, there is an infinite reduction

$$s = \mathtt{f}(s_1, \ldots, s_k) \rightarrow^*_{\mathcal{R},in} \mathtt{f}(s'_1, \ldots, s'_k) = \ell\gamma \rightarrow_{\mathcal{R}} r\gamma \rightarrow_{\mathcal{R}} \ldots$$

Let p be the smallest subterm of r such that $p\gamma$ is non-terminating.

We easily see: $\ell^\sharp \rightarrow p^\sharp$ is a dependency pair!

Proving termination using dependency pairs

Rules:

$$\begin{array}{rcl}
\mathtt{minus}(x, \mathtt{0}) & \to & x \\
\mathtt{minus}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{minus}(x, y) \\
\mathtt{quot}(\mathtt{0}, \mathtt{s}(y)) & \to & \mathtt{0} \\
\mathtt{quot}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{s}(\mathtt{quot}(\mathtt{minus}(x, y), \mathtt{s}(y)))
\end{array}$$

Proving termination using dependency pairs

Rules:

$$\begin{array}{rcl} \mathtt{minus}(x, \mathtt{0}) & \to & x \\ \mathtt{minus}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{minus}(x, y) \\ \mathtt{quot}(\mathtt{0}, \mathtt{s}(y)) & \to & \mathtt{0} \\ \mathtt{quot}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{s}(\mathtt{quot}(\mathtt{minus}(x, y), \mathtt{s}(y))) \end{array}$$

Dependency pairs:

$$\begin{array}{rcl} \mathtt{minus}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{minus}^\sharp(x, y) \\ \mathtt{quot}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{minus}^\sharp(x, y), \mathtt{s}(y) \\ \mathtt{quot}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{quot}^\sharp(\mathtt{minus}(x, y), \mathtt{s}(y)) \end{array}$$

Proving termination using dependency pairs

Rules:

$$\begin{array}{rcl} \mathtt{minus}(x, \mathtt{0}) & \to & x \\ \mathtt{minus}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{minus}(x, y) \\ \mathtt{quot}(\mathtt{0}, \mathtt{s}(y)) & \to & \mathtt{0} \\ \mathtt{quot}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{s}(\mathtt{quot}(\mathtt{minus}(x, y), \mathtt{s}(y))) \end{array}$$

Dependency pairs:

$$\begin{array}{rcl} \mathtt{minus}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{minus}^\sharp(x, y) \\ \mathtt{quot}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{minus}^\sharp(x, y), \mathtt{s}(y) \\ \mathtt{quot}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{quot}^\sharp(\mathtt{minus}(x, y), \mathtt{s}(y)) \end{array}$$

Observation: In an infinite chain, if ever we encounter a root symbol $\mathtt{minus}^\sharp$ the root symbol never becomes $\mathtt{quot}^\sharp$ again!

Modularity using dependency pairs

Idea:

$\mathcal{R}_{\texttt{quot}}$ is non-terminating

Modularity using dependency pairs

Idea:

$\mathcal{R}_{\mathtt{quot}}$ is non-terminating

if and only if

Modularity using dependency pairs

Idea:

$\mathcal{R}_{\texttt{quot}}$ is non-terminating

if and only if

there is an infinite minimal $(\texttt{DP}(\mathcal{R}_{\texttt{quot}}), \mathcal{R}_{\texttt{quot}})$-chain

Modularity using dependency pairs

Idea:

$\mathcal{R}_{\mathtt{quot}}$ is non-terminating

if and only if

there is an infinite minimal $(\mathtt{DP}(\mathcal{R}_{\mathtt{quot}}), \mathcal{R}_{\mathtt{quot}})$-chain

if and only if

Modularity using dependency pairs

Idea:

$\mathcal{R}_{\mathtt{quot}}$ is non-terminating

if and only if

there is an infinite minimal $(\mathtt{DP}(\mathcal{R}_{\mathtt{quot}}), \mathcal{R}_{\mathtt{quot}})$-chain

if and only if

there is an infinite minimal
$(\{\mathtt{quot}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) \to \mathtt{quot}^\sharp(\mathtt{minus}(x, y), \mathtt{s}(y))\}, \mathcal{R}_{\mathtt{quot}})$-chain
or
there is an infinite minimal
$(\{\mathtt{minus}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) \to \mathtt{minus}^\sharp(x, y)\}, \mathcal{R}_{\mathtt{quot}})$-chain

Exercises

1. Identify the dependency pairs of:

$$
\begin{array}{rcl}
\mathtt{ack}(\mathtt{0}, y) & \rightarrow & \mathtt{s}(y) \\
\mathtt{ack}(\mathtt{s}(x), \mathtt{0}) & \rightarrow & \mathtt{ack}(x, \mathtt{s}(\mathtt{0})) \\
\mathtt{ack}(\mathtt{s}(x), \mathtt{s}(y)) & \rightarrow & \mathtt{ack}(x, \mathtt{ack}(\mathtt{s}(x), y)) \\
\mathtt{inc}(\mathtt{0}) & \rightarrow & \mathtt{s}(\mathtt{inc}(\mathtt{s}(\mathtt{0}))) \\
\mathtt{fexp}(\mathtt{0}, y) & \rightarrow & y \\
\mathtt{fexp}(\mathtt{s}(x), y) & \rightarrow & \mathtt{double}(x, y, \mathtt{0}) \\
\mathtt{double}(x, \mathtt{0}, z) & \rightarrow & \mathtt{fexp}(x, z) \\
\mathtt{double}(x, \mathtt{s}(y), z) & \rightarrow & \mathtt{double}(x, y, \mathtt{s}(\mathtt{s}(z)))
\end{array}
$$

Exercises

1. Identify the dependency pairs of:

$$
\begin{array}{rcl}
\mathtt{ack}(\mathtt{0}, y) & \to & \mathtt{s}(y) \\
\mathtt{ack}(\mathtt{s}(x), \mathtt{0}) & \to & \mathtt{ack}(x, \mathtt{s}(\mathtt{0})) \\
\mathtt{ack}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{ack}(x, \mathtt{ack}(\mathtt{s}(x), y)) \\
\mathtt{inc}(\mathtt{0}) & \to & \mathtt{s}(\mathtt{inc}(\mathtt{s}(\mathtt{0}))) \\
\mathtt{fexp}(\mathtt{0}, y) & \to & y \\
\mathtt{fexp}(\mathtt{s}(x), y) & \to & \mathtt{double}(x, y, \mathtt{0}) \\
\mathtt{double}(x, \mathtt{0}, z) & \to & \mathtt{fexp}(x, z) \\
\mathtt{double}(x, \mathtt{s}(y), z) & \to & \mathtt{double}(x, y, \mathtt{s}(\mathtt{s}(z)))
\end{array}
$$

2. Can you split up the resulting problem whether an infinite minimal $(\mathtt{DP}(\mathcal{R}), \mathcal{R})$-chain exists?

Exercises

1. Identify the dependency pairs of:

$$
\begin{array}{rcl}
\mathtt{ack}(\mathtt{0}, y) & \to & \mathtt{s}(y) \\
\mathtt{ack}(\mathtt{s}(x), \mathtt{0}) & \to & \mathtt{ack}(x, \mathtt{s}(\mathtt{0})) \\
\mathtt{ack}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{ack}(x, \mathtt{ack}(\mathtt{s}(x), y)) \\
\mathtt{inc}(\mathtt{0}) & \to & \mathtt{s}(\mathtt{inc}(\mathtt{s}(\mathtt{0}))) \\
\mathtt{fexp}(\mathtt{0}, y) & \to & y \\
\mathtt{fexp}(\mathtt{s}(x), y) & \to & \mathtt{double}(x, y, \mathtt{0}) \\
\mathtt{double}(x, \mathtt{0}, z) & \to & \mathtt{fexp}(x, z) \\
\mathtt{double}(x, \mathtt{s}(y), z) & \to & \mathtt{double}(x, y, \mathtt{s}(\mathtt{s}(z)))
\end{array}
$$

2. Can you split up the resulting problem whether an infinite minimal $(\mathtt{DP}(\mathcal{R}), \mathcal{R})$-chain exists?
3. This should result in multiple problems “is there an infinite minimal $(\mathcal{P}, \mathcal{R})$-chain?” Can you prove for some of them that the answer is no (such a chain does not exist)?

Higher-order challenges

Higher-order challenges

Discussion: what should be the dependency pairs of $\mathcal{R}_{\mathtt{map}}$?

$$
\begin{array}{rcl}
\mathtt{map}(F, []) & \to & [] \\
\mathtt{map}(F, \mathtt{cons}(x, l)) & \to & \mathtt{cons}(F \cdot x, \mathtt{map}(F, l))
\end{array}
$$

Higher-order challenges

Discussion: what should be the dependency pairs of $\mathcal{R}_{\mathtt{map}}$?

$$\begin{array}{rcl} \mathtt{map}(F, []) & \to & [] \\ \mathtt{map}(F, \mathtt{cons}(x, l)) & \to & \mathtt{cons}(F \cdot x, \mathtt{map}(F, l)) \end{array}$$

Two approaches:

- dynamic dependency pairs: include collapsing DPs like $\mathtt{map}^\sharp(F, \mathtt{cons}(x, l)) \to F \cdot x$
- static dependency pairs: only include non-collapsing DPs like $\mathtt{map}^\sharp(F, \mathtt{cons}(x, l)) \to \mathtt{map}^\sharp(F, l)$.

Higher-order challenges

Discussion: what should be the dependency pairs of $\mathcal{R}_{\mathtt{map}}$?

$$\begin{array}{rcl} \mathtt{map}(F, []) & \to & [] \\ \mathtt{map}(F, \mathtt{cons}(x, l)) & \to & \mathtt{cons}(\underline{F \cdot x}, \underline{\mathtt{map}(F, l)}) \end{array}$$

Two approaches:

- dynamic dependency pairs: include collapsing DPs like $\mathtt{map}^\sharp(F, \mathtt{cons}(x, l)) \to F \cdot x$
- static dependency pairs: only include non-collapsing DPs like $\mathtt{map}^\sharp(F, \mathtt{cons}(x, l)) \to \mathtt{map}^\sharp(F, l)$.

Underlying proof idea:

Higher-order challenges

Discussion: what should be the dependency pairs of $\mathcal{R}_{\mathtt{map}}$?

$$\begin{array}{rcl} \mathtt{map}(F, []) & \to & [] \\ \mathtt{map}(F, \mathtt{cons}(x, l)) & \to & \mathtt{cons}(F \cdot x, \mathtt{map}(F, l)) \end{array}$$

Two approaches:

- dynamic dependency pairs: include collapsing DPs like $\mathtt{map}^\sharp(F, \mathtt{cons}(x, l)) \to F \cdot x$
- static dependency pairs: only include non-collapsing DPs like $\mathtt{map}^\sharp(F, \mathtt{cons}(x, l)) \to \mathtt{map}^\sharp(F, l)$.

Underlying proof idea:

- dynamic DPs: in a DP $\mathtt{f}^\sharp(\ell_1, \dots, \ell_k) \to r$, all (instances of each) ℓ_i are assumed to be terminating.

Higher-order challenges

Discussion: what should be the dependency pairs of $\mathcal{R}_{\mathtt{map}}$?

$$\begin{array}{rcl} \mathtt{map}(F, []) & \to & [] \\ \mathtt{map}(F, \mathtt{cons}(x, l)) & \to & \mathtt{cons}(F \cdot x, \mathtt{map}(F, l)) \end{array}$$

Two approaches:

- dynamic dependency pairs: include collapsing DPs like $\mathtt{map}^\sharp(F, \mathtt{cons}(x, l)) \to F \cdot x$
- static dependency pairs: only include non-collapsing DPs like $\mathtt{map}^\sharp(F, \mathtt{cons}(x, l)) \to \mathtt{map}^\sharp(F, l)$.

Underlying proof idea:

- dynamic DPs: in a DP $\mathtt{f}^\sharp(\ell_1, \dots, \ell_k) \to r$, all (instances of each) ℓ_i are assumed to be terminating.
- static DPs: in a DP $\mathtt{f}^\sharp(\ell_1, \dots, \ell_k) \to r$, all (instances of each) ℓ_i are assumed to be compuable.

Higher-order challenges

Discussion: what should be the dependency pairs of $\mathcal{R}_{\mathtt{map}}$?

$$\begin{array}{rcl} \mathtt{map}(F, []) & \to & [] \\ \mathtt{map}(F, \mathtt{cons}(x, l)) & \to & \mathtt{cons}(F \cdot x, \mathtt{map}(F, l)) \end{array}$$

Two approaches:

- static dependency pairs: only include non-collapsing DPs like $\mathtt{map}^\sharp(F, \mathtt{cons}(x, l)) \to \mathtt{map}^\sharp(F, l)$.

Underlying proof idea:

- static DPs: in a DP $\mathtt{f}^\sharp(\ell_1, \dots, \ell_k) \to r$, all (instances of each) ℓ_i are assumed to be compuable.

Higher-order challenges

Discussion: what should be the dependency pairs of:

$$\mathtt{up}(l) \to \mathtt{map}(\lambda x.\mathtt{double}(x), l)$$

Higher-order challenges

Discussion: what should be the dependency pairs of:

$$\mathtt{up}(l) \to \mathtt{map}(\lambda x.\mathtt{double}(x), l)$$

Likely answer:

$$\mathtt{up}^\sharp(l) \to \mathtt{map}^\sharp(\lambda x.\mathtt{double}(x), l)$$
$$\mathtt{up}^\sharp(l) \to \mathtt{double}^\sharp(x)$$

Higher-order challenges

Discussion: what should be the dependency pairs of:

$$\mathtt{up}(l) \to \mathtt{map}(\lambda x.\mathtt{double}(x), l)$$

Likely answer:

$$\mathtt{up}^\sharp(l) \to \mathtt{map}^\sharp(\lambda x.\mathtt{double}(x), l)$$

$$\mathtt{up}^\sharp(l) \to \mathtt{double}^\sharp(x) \quad \Leftarrow \text{ fresh variable } x$$

Higher-order challenges

Discussion: what should be the dependency pairs of:

$$\mathtt{up}(l) \to \mathtt{map}(\lambda x.\mathtt{double}(x), l)$$

Likely answer:

$$\mathtt{up}^\sharp(l) \to \mathtt{map}^\sharp(\lambda x.\mathtt{double}(x), l)$$
$$\mathtt{up}^\sharp(l) \to \mathtt{double}^\sharp(x) \quad \Leftarrow \text{ fresh variable } x$$

But: we may assume x is computable.

Higher-order challenges

Discussion: what should be the dependency pairs of:

$$\mathtt{up}(l) \to \mathtt{map}(\mathtt{double}, l)$$

Higher-order challenges

Discussion: what should be the dependency pairs of:

$$\mathtt{up}(l) \to \mathtt{map}(\mathtt{double}, l)$$

Likely answer:

$$\mathtt{up}^\sharp(l) \to \mathtt{map}^\sharp(\mathtt{double}, l)$$
$$\mathtt{up}^\sharp(l) \to \mathtt{double}^\sharp(x)$$

Higher-order challenges

Discussion: what should be the dependency pairs of:

$$\mathtt{up}(l) \to \mathtt{map}(\mathtt{double}, l)$$

Likely answer:

$$\mathtt{up}^\sharp(l) \to \mathtt{map}^\sharp(\mathtt{double}, l)$$
$$\mathtt{up}^\sharp(l) \to \mathtt{double}^\sharp(x)$$

Again: we may assume that x is computable.

Higher-order challenges

Discussion: what should be the dependency pairs of:

$$\mathtt{up} \to \mathtt{map}(\mathtt{double})$$

Higher-order challenges

Discussion: what should be the dependency pairs of:

$$\mathtt{up} \to \mathtt{map}(\mathtt{double})$$

Likely answer:

$$\mathtt{up}^\sharp(l) \to \mathtt{map}^\sharp(\mathtt{double}, l)$$
$$\mathtt{up}^\sharp(l) \to \mathtt{double}^\sharp(x)$$

Higher-order challenges

Discussion: what should be the dependency pairs of:

$$\mathtt{f}(F, x) \to F \cdot x$$

Higher-order challenges

Discussion: what should be the dependency pairs of:

$$\mathtt{f}(F, x) \rightarrow F \cdot x$$

Likely answer: this should not have any dependency pairs!

Higher-order challenges

Discussion: what should be the dependency pairs of:

$$\mathtt{f}(F, x) \to F \cdot x$$

Likely answer: this should not have any dependency pairs!

Discussion: what should be the dependency pairs of:

$$\mathtt{app}(\mathtt{lam}(F), x) \to F \cdot x$$

Higher-order challenges

Discussion: what should be the dependency pairs of:

$$\mathtt{f}(F, x) \rightarrow F \cdot x$$

Likely answer: this should not have any dependency pairs!

Discussion: what should be the dependency pairs of:

$$\mathtt{app}(\mathtt{lam}(F), x) \rightarrow F \cdot x$$

Likely answer: This should not be allowed!

Plain function passing

Definition

A HTRS is plain function passing if:

Plain function passing

Definition

A HTRS is plain function passing if:
for all rules $\mathtt{f}(\ell_1, \ldots, \ell_k) \to r$:

Plain function passing

Definition

A HTRS is plain function passing if:
for all rules $\mathtt{f}(\ell_1, \ldots, \ell_k) \to r$:
if $\ell_i \trianglerighteq F$ with F a variable of higher type

Plain function passing

Definition

A HTRS is plain function passing if:
for all rules $\mathtt{f}(\ell_1, \ldots, \ell_k) \to r$:
if $\ell_i \trianglerighteq F$ with F a variable of higher type
then $\ell_i = F$ or F does not occur in r

Plain function passing

$$\begin{array}{rcl} [] & :: & \text{list} \\ \texttt{cons} & :: & \text{nat} \Rightarrow \text{list} \Rightarrow \text{list} \\ \texttt{double} & :: & \text{nat} \Rightarrow \text{nat} \\ \texttt{map} & :: & (\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{list} \Rightarrow \text{list} \\ \texttt{up} & :: & \text{list} \Rightarrow \text{list} \end{array}$$

$$\begin{array}{rcl} \texttt{map}(F, []) & \rightarrow & [] \\ \texttt{map}(F, \texttt{cons}(x, l)) & \rightarrow & \texttt{cons}(F \cdot x, \texttt{map}(F, l)) \\ \texttt{up}(l) & \rightarrow & \texttt{map}(\lambda x.\texttt{double}(x), l) \end{array}$$

Plain function passing

$$
\begin{array}{rcl}
[] & :: & \text{list} \\
\texttt{cons} & :: & \text{nat} \Rightarrow \text{list} \Rightarrow \text{list} \\
\texttt{double} & :: & \text{nat} \Rightarrow \text{nat} \\
\texttt{map} & :: & (\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{list} \Rightarrow \text{list} \\
\texttt{up} & :: & \text{list} \Rightarrow \text{list}
\end{array}
$$

$$
\begin{array}{rcl}
\texttt{map}(F, []) & \to & [] \\
\texttt{map}(F, \texttt{cons}(x, l)) & \to & \texttt{cons}(F \cdot x, \texttt{map}(F, l)) \\
\texttt{up}(l) & \to & \texttt{map}(\lambda x.\texttt{double}(x), l)
\end{array}
$$

✓

Plain function passing

$$\texttt{app} :: \text{term} \Rightarrow \text{term} \Rightarrow \text{term}$$
$$\texttt{lam} :: (\text{term} \Rightarrow \text{term}) \Rightarrow \text{term}$$

$$\texttt{app}(\texttt{lam}(F)) \rightarrow F$$

Plain function passing

$$\texttt{app} :: \text{term} \Rightarrow \text{term} \Rightarrow \text{term}$$
$$\texttt{lam} :: (\text{term} \Rightarrow \text{term}) \Rightarrow \text{term}$$

$$\texttt{app}(\texttt{lam}(F)) \rightarrow F$$

✗

Plain function passing

$$
\begin{array}{rcl}
[] & :: & \text{list} \\
\mathtt{cons} & :: & (\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{list} \Rightarrow \text{list} \\
\mathtt{map} & :: & ((\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{nat} \Rightarrow \text{nat}) \Rightarrow \text{list} \Rightarrow \text{list} \\
\mathtt{up} & :: & \text{list} \Rightarrow \text{list}
\end{array}
$$

$$
\begin{array}{rcl}
\mathtt{map}(F, []) & \rightarrow & [] \\
\mathtt{map}(F, \mathtt{cons}(x, l)) & \rightarrow & \mathtt{cons}(F \cdot x, \mathtt{map}(F, l)) \\
\mathtt{up}(l) & \rightarrow & \mathtt{map}(\lambda x.x, l)
\end{array}
$$

Plain function passing

$$
\begin{array}{rcl}
[] & :: & \text{list} \\
\texttt{cons} & :: & (\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{list} \Rightarrow \text{list} \\
\texttt{map} & :: & ((\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{nat} \Rightarrow \text{nat}) \Rightarrow \text{list} \Rightarrow \text{list} \\
\texttt{up} & :: & \text{list} \Rightarrow \text{list}
\end{array}
$$

$$
\begin{array}{rcl}
\texttt{map}(F, []) & \to & [] \\
\texttt{map}(F, \texttt{cons}(x, l)) & \to & \texttt{cons}(F \cdot x, \texttt{map}(F, l)) \\
\texttt{up}(l) & \to & \texttt{map}(\lambda x.x, l)
\end{array}
$$

✗

Plain function passing

$$
\begin{array}{rcl}
\mathtt{I}(x) & \to & x \\
\mathtt{minus}(x, \mathtt{0}) & \to & x \\
\mathtt{minus}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{minus}(x, y) \\
\mathtt{quot}(\mathtt{0}, \mathtt{s}(y)) & \to & \mathtt{0} \\
\mathtt{quot}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{s}(\mathtt{quot}(\mathtt{minus}(x, y), \mathtt{s}(y))) \\
\mathtt{ack}(\mathtt{0}, y) & \to & \mathtt{s}(y) \\
\mathtt{ack}(\mathtt{s}(x), \mathtt{0}) & \to & \mathtt{ack}(x, \mathtt{s}(\mathtt{0})) \\
\mathtt{ack}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{ack}(x, \mathtt{ack}(\mathtt{s}(x), y)) \\
\mathtt{inc}(\mathtt{0}) & \to & \mathtt{s}(\mathtt{inc}(\mathtt{s}(\mathtt{0}))) \\
\mathtt{fexp}(\mathtt{0}, y) & \to & y \\
\mathtt{fexp}(\mathtt{s}(x), y) & \to & \mathtt{double}(x, y, \mathtt{0}) \\
\mathtt{double}(x, \mathtt{0}, z) & \to & \mathtt{fexp}(x, z) \\
\mathtt{double}(x, \mathtt{s}(y), z) & \to & \mathtt{double}(x, y, \mathtt{s}(\mathtt{s}(z))) \\
\mathtt{hd}(\mathtt{cons}(x, l)) & \to & x \\
\mathtt{len}([]) & \to & \mathtt{0} \\
\mathtt{len}(\mathtt{cons}(x, l)) & \to & \mathtt{s}(\mathtt{len}(l)) \\
\mathtt{map}(F, []) & \to & [] \\
\mathtt{map}(F, \mathtt{cons}(x, l)) & \to & \mathtt{cons}(F \cdot x, \mathtt{map}(F, l)) \\
\mathtt{fold}(F, x, []) & \to & x \\
\mathtt{fold}(F, x, \mathtt{cons}(y, l)) & \to & \mathtt{fold}(F, F \cdot x \cdot y, l) \\
\mathtt{mkbig}(l, x) & \to & \mathtt{map}(\mathtt{ack}(x), l) \\
\mathtt{mkdiv}(l, x) & \to & \mathtt{map}(\lambda y.\mathtt{quot}(y, x), l) \\
\mathtt{sma}(b, F, \mathtt{0}) & \to & \mathtt{0} \\
\mathtt{sma}(\mathtt{true}, F, \mathtt{s}(x)) & \to & \mathtt{s}(x) \\
\mathtt{sma}(\mathtt{false}, F, \mathtt{s}(x)) & \to & \mathtt{sma}(F \cdot x, F, \mathtt{quot}(x, \mathtt{s}\ (\mathtt{s}\ \mathtt{0}))) \\
\mathtt{twice}(F, x) & \to & F \cdot (F \cdot x) \\
\mathtt{H}(\mathtt{s}(x)) & \to & \mathtt{H}(\mathtt{twice}(\mathtt{I}, x))
\end{array}
$$

Plain function passing

$$
\begin{array}{rcl}
\mathtt{I}(x) & \to & x \\
\mathtt{minus}(x, \mathtt{0}) & \to & x \\
\mathtt{minus}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{minus}(x, y) \\
\mathtt{quot}(\mathtt{0}, \mathtt{s}(y)) & \to & \mathtt{0} \\
\mathtt{quot}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{s}(\mathtt{quot}(\mathtt{minus}(x, y), \mathtt{s}(y))) \\
\mathtt{ack}(\mathtt{0}, y) & \to & \mathtt{s}(y) \\
\mathtt{ack}(\mathtt{s}(x), \mathtt{0}) & \to & \mathtt{ack}(x, \mathtt{s}(\mathtt{0})) \\
\mathtt{ack}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{ack}(x, \mathtt{ack}(\mathtt{s}(x), y)) \\
\mathtt{inc}(\mathtt{0}) & \to & \mathtt{s}(\mathtt{inc}(\mathtt{s}(\mathtt{0}))) \\
\mathtt{fexp}(\mathtt{0}, y) & \to & y \\
\mathtt{fexp}(\mathtt{s}(x), y) & \to & \mathtt{double}(x, y, \mathtt{0}) \\
\mathtt{double}(x, \mathtt{0}, z) & \to & \mathtt{fexp}(x, z) \\
\mathtt{double}(x, \mathtt{s}(y), z) & \to & \mathtt{double}(x, y, \mathtt{s}(\mathtt{s}(z))) \\
\mathtt{hd}(\mathtt{cons}(x, l)) & \to & x \\
\mathtt{len}([]) & \to & \mathtt{0} \\
\mathtt{len}(\mathtt{cons}(x, l)) & \to & \mathtt{s}(\mathtt{len}(l)) \\
\mathtt{map}(F, []) & \to & [] \\
\mathtt{map}(F, \mathtt{cons}(x, l)) & \to & \mathtt{cons}(F \cdot x, \mathtt{map}(F, l)) \\
\mathtt{fold}(F, x, []) & \to & x \\
\mathtt{fold}(F, x, \mathtt{cons}(y, l)) & \to & \mathtt{fold}(F, F \cdot x \cdot y, l) \\
\mathtt{mkbig}(l, x) & \to & \mathtt{map}(\mathtt{ack}(x), l) \\
\mathtt{mkdiv}(l, x) & \to & \mathtt{map}(\lambda y.\mathtt{quot}(y, x), l) \\
\mathtt{sma}(b, F, \mathtt{0}) & \to & \mathtt{0} \\
\mathtt{sma}(\mathtt{true}, F, \mathtt{s}(x)) & \to & \mathtt{s}(x) \\
\mathtt{sma}(\mathtt{false}, F, \mathtt{s}(x)) & \to & \mathtt{sma}(F \cdot x, F, \mathtt{quot}(x, \mathtt{s}\ (\mathtt{s}\ \mathtt{0}))) \\
\mathtt{twice}(F, x) & \to & F \cdot (F \cdot x) \\
\mathtt{H}(\mathtt{s}(x)) & \to & \mathtt{H}(\mathtt{twice}(\mathtt{I}, x))
\end{array}
$$

✓

Definition

Definition

For a term s the candidates of s are given by:

$$
\begin{array}{rcl}
\mathsf{Cand}(\mathtt{f}(s_1,\ldots,s_n)) & = & \{\mathtt{f}(s_1,\ldots,s_n)\} \cup \bigcup_{i=1}^{n} \mathsf{Cand}(s_i) \\
\mathsf{Cand}(\mathtt{c}(s_1,\ldots,s_n)) & = & \bigcup_{i=1}^{n} \mathsf{Cand}(s_i) \\
\mathsf{Cand}(x \cdot s_1 \cdots s_n) & = & \bigcup_{i=1}^{n} \mathsf{Cand}(s_i) \\
\mathsf{Cand}(\lambda x.s) & = & \mathsf{Cand}(s[x := y]) \\
\mathsf{Cand}((\lambda x.t) \cdot s_0 \cdots s_n & = & \mathsf{Cand}(t[x := s_1] \cdot s_1 \cdots s_n) \cup \mathsf{Cand}(s_1)
\end{array}
$$

Definition

Definition

For a term s the candidates of s are given by:

$$\begin{array}{rcl}
\mathsf{Cand}(\mathtt{f}(s_1,\dots,s_n)) & = & \{\mathtt{f}(s_1,\dots,s_n)\} \cup \bigcup_{i=1}^n \mathsf{Cand}(s_i) \\
\mathsf{Cand}(\mathtt{c}(s_1,\dots,s_n)) & = & \bigcup_{i=1}^n \mathsf{Cand}(s_i) \\
\mathsf{Cand}(x \cdot s_1 \cdots s_n) & = & \bigcup_{i=1}^n \mathsf{Cand}(s_i) \\
\mathsf{Cand}(\lambda x.s) & = & \mathsf{Cand}(s[x := y]) \\
\mathsf{Cand}((\lambda x.t) \cdot s_0 \cdots s_n & = & \mathsf{Cand}(t[x := s_1] \cdot s_1 \cdots s_n) \cup \mathsf{Cand}(s_1)
\end{array}$$

Dependency pairs of a rule $\mathtt{f}(\ell_1,\dots,\ell_k) \to r$:

Definition

Definition

For a term s the candidates of s are given by:

$$
\begin{array}{rcl}
\mathrm{Cand}(\mathtt{f}(s_1,\dots,s_n)) & = & \{\mathtt{f}(s_1,\dots,s_n)\} \cup \bigcup_{i=1}^{n} \mathrm{Cand}(s_i) \\
\mathrm{Cand}(\mathtt{c}(s_1,\dots,s_n)) & = & \bigcup_{i=1}^{n} \mathrm{Cand}(s_i) \\
\mathrm{Cand}(x \cdot s_1 \cdots s_n) & = & \bigcup_{i=1}^{n} \mathrm{Cand}(s_i) \\
\mathrm{Cand}(\lambda x.s) & = & \mathrm{Cand}(s[x := y]) \\
\mathrm{Cand}((\lambda x.t) \cdot s_0 \cdots s_n & = & \mathrm{Cand}(t[x := s_1] \cdot s_1 \cdots s_n) \cup \mathrm{Cand}(s_1)
\end{array}
$$

Dependency pairs of a rule $\mathtt{f}(\ell_1,\dots,\ell_k) \to r$:

- if $r :: \sigma_1 \Rightarrow \dots \Rightarrow \sigma_m \Rightarrow \iota$

Definition

Definition

For a term s the candidates of s are given by:

$$
\begin{array}{rcl}
\mathsf{Cand}(\mathtt{f}(s_1,\ldots,s_n)) & = & \{\mathtt{f}(s_1,\ldots,s_n)\} \cup \bigcup_{i=1}^{n} \mathsf{Cand}(s_i) \\
\mathsf{Cand}(\mathtt{c}(s_1,\ldots,s_n)) & = & \bigcup_{i=1}^{n} \mathsf{Cand}(s_i) \\
\mathsf{Cand}(x \cdot s_1 \cdots s_n) & = & \bigcup_{i=1}^{n} \mathsf{Cand}(s_i) \\
\mathsf{Cand}(\lambda x.s) & = & \mathsf{Cand}(s[x := y]) \\
\mathsf{Cand}((\lambda x.t) \cdot s_0 \cdots s_n & = & \mathsf{Cand}(t[x := s_1] \cdot s_1 \cdots s_n) \cup \mathsf{Cand}(s_1)
\end{array}
$$

Dependency pairs of a rule $\mathtt{f}(\ell_1,\ldots,\ell_k) \to r$:

- if $r :: \sigma_1 \Rightarrow \ldots \Rightarrow \sigma_m \Rightarrow \iota$
- and $\mathtt{g}(t_1,\ldots,t_n) \in \mathsf{Cand}(r \cdot x_1 \cdots x_m)$ (fresh $\vec{x}$)

Definition

Definition

For a term s the candidates of s are given by:

$$
\begin{array}{rcl}
\mathsf{Cand}(\mathtt{f}(s_1,\dots,s_n)) & = & \{\mathtt{f}(s_1,\dots,s_n)\} \cup \bigcup_{i=1}^{n} \mathsf{Cand}(s_i) \\
\mathsf{Cand}(\mathtt{c}(s_1,\dots,s_n)) & = & \bigcup_{i=1}^{n} \mathsf{Cand}(s_i) \\
\mathsf{Cand}(x \cdot s_1 \cdots s_n) & = & \bigcup_{i=1}^{n} \mathsf{Cand}(s_i) \\
\mathsf{Cand}(\lambda x.s) & = & \mathsf{Cand}(s[x := y]) \\
\mathsf{Cand}((\lambda x.t) \cdot s_0 \cdots s_n & = & \mathsf{Cand}(t[x := s_1] \cdot s_1 \cdots s_n) \cup \mathsf{Cand}(s_1)
\end{array}
$$

Dependency pairs of a rule $\mathtt{f}(\ell_1,\dots,\ell_k) \to r$:

- if $r :: \sigma_1 \Rightarrow \dots \Rightarrow \sigma_m \Rightarrow \iota$
- and $\mathtt{g}(t_1,\dots,t_n) \in \mathsf{Cand}(r \cdot x_1 \cdots x_m)$ (fresh $\vec{x}$)
- and $\mathtt{g}(t_1,\dots,t_n) :: \tau_1 \Rightarrow \dots \Rightarrow \tau_p \Rightarrow \kappa$

Definition

Definition

For a term s the candidates of s are given by:

$$
\begin{array}{rcl}
\text{Cand}(\mathtt{f}(s_1,\dots,s_n)) & = & \{\mathtt{f}(s_1,\dots,s_n)\} \cup \bigcup_{i=1}^{n} \text{Cand}(s_i) \\
\text{Cand}(\mathtt{c}(s_1,\dots,s_n)) & = & \bigcup_{i=1}^{n} \text{Cand}(s_i) \\
\text{Cand}(x \cdot s_1 \cdots s_n) & = & \bigcup_{i=1}^{n} \text{Cand}(s_i) \\
\text{Cand}(\lambda x.s) & = & \text{Cand}(s[x := y]) \\
\text{Cand}((\lambda x.t) \cdot s_0 \cdots s_n & = & \text{Cand}(t[x := s_1] \cdot s_1 \cdots s_n) \cup \text{Cand}(s_1)
\end{array}
$$

Dependency pairs of a rule $\mathtt{f}(\ell_1,\dots,\ell_k) \to r$:

- if $r :: \sigma_1 \Rightarrow \dots \Rightarrow \sigma_m \Rightarrow \iota$
- and $\mathtt{g}(t_1,\dots,t_n) \in \text{Cand}(r \cdot x_1 \cdots x_m)$ (fresh $\vec{x}$)
- and $\mathtt{g}(t_1,\dots,t_n) :: \tau_1 \Rightarrow \dots \Rightarrow \tau_p \Rightarrow \kappa$
- then $\mathtt{f}^\sharp(\ell_1,\dots,\ell_k,x_1,\dots,x_m) \to \mathtt{g}^\sharp(t_1,\dots,t_n,y_1,\dots,y_p)$ is in a dependency pair of this rule (for fresh $\vec{y}$)

Exercise

Compute the dependency pairs of:

$$\begin{array}{rcl}
\mathtt{0} & :: & \text{nat} \\
\mathtt{s} & :: & \text{nat} \Rightarrow \text{nat} \\
\mathtt{a} & :: & \text{o} \\
\mathtt{c} & :: & \text{o} \Rightarrow \text{o} \\
\mathtt{rec} & :: & \text{nat} \Rightarrow \text{nat} \Rightarrow (\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat}) \Rightarrow \text{nat} \\
\mathtt{add} & :: & \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \\
\mathtt{mul} & :: & \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \\
\mathtt{f} & :: & \text{o} \Rightarrow \text{o}
\end{array}$$

$$\begin{array}{rcl}
\mathtt{rec}(\mathtt{0}, F, y) & \to & y \\
\mathtt{rec}(\mathtt{s}(x), F, y) & \to & F \cdot x \cdot \mathtt{rec}(x, F, y) \\
\mathtt{add}(x) & \to & \mathtt{rec}(x, \lambda z.\mathtt{s}) \\
\mathtt{mul}(x) & \to & \mathtt{rec}(x, \lambda z.\mathtt{add}(z)) \\
\mathtt{f}(\mathtt{b}) & \to & \mathtt{c}(\ (\lambda x.\mathtt{f}(x)) \cdot \mathtt{a}\ ) \\
\mathtt{f}(\mathtt{a}) & \to & \mathtt{c}(\ (\lambda x.\mathtt{a}) \cdot \mathtt{f}(\mathtt{b})\ )
\end{array}$$

Higher-order-order dependency pair chain

Definition: a computable DP chain over $(\mathcal{P}, \mathcal{R})$ is a reduction chain:

$$s_1 \to_{\mathcal{P}} t_1 \to_{\mathcal{R}}^* s_2 \to_{\mathcal{P}} t_2 \to_{\mathcal{R}}^* \dots$$

Such that:

- each reduction $s_i \to_{\mathcal{P}} t_i$ is at the root
- each reduction $s_i \to_{\mathcal{R}}^* t_i$ occurs below the root
- each t_i is computable with respect to $\to_{\mathcal{R}}$

Claim:

there is an infinite computable $(\mathtt{DP}(\mathcal{R}), \mathcal{R})$-chain
if $\to_{\mathcal{R}}$ is non-terminating

Higher-order-order dependency pair chain

Definition: a computable DP chain over $(\mathcal{P}, \mathcal{R})$ is a reduction chain:

$$s_1 \to_{\mathcal{P}} t_1 \to_{\mathcal{R}}^* s_2 \to_{\mathcal{P}} t_2 \to_{\mathcal{R}}^* \dots$$

Such that:

- each reduction $s_i \to_{\mathcal{P}} t_i$ is at the root
- each reduction $s_i \to_{\mathcal{R}}^* t_i$ occurs below the root
- each t_i is computable with respect to $\to_{\mathcal{R}}$

Claim:

there is an infinite computable $(\mathtt{DP}(\mathcal{R}), \mathcal{R})$-chain
if $\to_{\mathcal{R}}$ is non-terminating

if there is an infinite computable $(\mathtt{DP}(\mathcal{R}), \mathcal{R})$-chain
using only dependency pairs $\ell \to r$ with $FV(r) \subseteq FV(\ell)$
then $\to_{\mathcal{R}}$ is non-terminating

Dependency chain claim: proof sketch

Claim:

there is an infinite computable $(\mathtt{DP}(\mathcal{R}), \mathcal{R})$-chain
if $\rightarrow_{\mathcal{R}}$ is non-terminating

Dependency chain claim: proof sketch

Claim:

there is an infinite computable $(\mathtt{DP}(\mathcal{R}), \mathcal{R})$-chain
if $\rightarrow_{\mathcal{R}}$ is non-terminating

Proof sketch:

Dependency chain claim: proof sketch

Claim:

there is an infinite computable $(\mathtt{DP}(\mathcal{R}), \mathcal{R})$-chain
if $\rightarrow_{\mathcal{R}}$ is non-terminating

Proof sketch:

- If $\rightarrow_{\mathcal{R}}$ is non-terminating, there is a non-terminating base-type term s whose strict subterms are comptable.

Dependency chain claim: proof sketch

Claim:

there is an infinite computable $(\mathtt{DP}(\mathcal{R}), \mathcal{R})$-chain
if $\rightarrow_{\mathcal{R}}$ is non-terminating

Proof sketch:

- If $\rightarrow_{\mathcal{R}}$ is non-terminating, there is a non-terminating base-type term s whose strict subterms are comptable.
- Consider an infinite reduction

$$s \rightarrow^{*}_{\mathcal{R},in} \mathtt{f}(s'_1, \ldots, s'_k) = \ell\gamma \rightarrow_{\mathcal{R}} r\gamma \rightarrow_{\mathcal{R}} \ldots$$

Dependency chain claim: proof sketch

Claim:

there is an infinite computable $(\mathtt{DP}(\mathcal{R}), \mathcal{R})$-chain
if $\rightarrow_{\mathcal{R}}$ is non-terminating

Proof sketch:

- If $\rightarrow_{\mathcal{R}}$ is non-terminating, there is a non-terminating base-type term s whose strict subterms are comptable.
- Consider an infinite reduction

$$s \rightarrow^*_{\mathcal{R},in} \mathtt{f}(s'_1, \dots, s'_k) = \ell\gamma \rightarrow_{\mathcal{R}} r\gamma \rightarrow_{\mathcal{R}} \dots$$

- Identify a smallest subterm p of r such that $p\gamma$ is non-computable.

Dependency chain claim: proof sketch

Claim:

there is an infinite computable $(\mathtt{DP}(\mathcal{R}), \mathcal{R})$-chain
if $\rightarrow_{\mathcal{R}}$ is non-terminating

Proof sketch:

- If $\rightarrow_{\mathcal{R}}$ is non-terminating, there is a non-terminating base-type term s whose strict subterms are comptable.
- Consider an infinite reduction

$$s \rightarrow^{*}_{\mathcal{R},in} \mathtt{f}(s'_1, \ldots, s'_k) = \ell\gamma \rightarrow_{\mathcal{R}} r\gamma \rightarrow_{\mathcal{R}} \ldots$$

- Identify a smallest subterm p of r such that $p\gamma$ is non-computable.
- Then $r \cdot y_1 \cdots y_p$ is a candidate.

Discussion:

Discussion:

Plain-function passingness:

Discussion:

Plain-function passingness:

- admits most (terminating) common examples

Discussion:

Plain-function passingness:

- admits most (terminating) common examples
- performs poorly with polymorphism (e.g., `cons` $:: \alpha \Rightarrow \mathsf{list}(\alpha) \Rightarrow \mathsf{list}(\alpha)$)

Discussion:

Plain-function passingness:

- admits most (terminating) common examples
- performs poorly with polymorphism (e.g., $\texttt{cons} :: \alpha \Rightarrow \text{list}(\alpha) \Rightarrow \text{list}(\alpha)$)
- but: requirement can be weakened to avoid this problem!

Discussion:

Plain-function passingness:

- admits most (terminating) common examples
- performs poorly with polymorphism (e.g., `cons` $:: \alpha \Rightarrow \text{list}(\alpha) \Rightarrow \text{list}(\alpha)$)
- but: requirement can be weakened to avoid this problem!

Polymorphism overall:

Discussion:

Plain-function passingness:

- admits most (terminating) common examples
- performs poorly with polymorphism (e.g., $\mathtt{cons} :: \alpha \Rightarrow \mathsf{list}(\alpha) \Rightarrow \mathsf{list}(\alpha)$)
- but: requirement can be weakened to avoid this problem!

Polymorphism overall:

- forcing rules into base-type before computing dependency pairs. . .

Discussion:

Plain-function passingness:

- admits most (terminating) common examples
- performs poorly with polymorphism (e.g., $\texttt{cons} :: \alpha \Rightarrow \mathsf{list}(\alpha) \Rightarrow \mathsf{list}(\alpha)$)
- but: requirement can be weakened to avoid this problem!

Polymorphism overall:

- forcing rules into base-type before computing dependency pairs. . .
- can be done with slightly different definitions

Dependency Pair Processors

Splitting by root symbol

Recall:

$$\begin{array}{rcl} \mathtt{minus}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{minus}^\sharp(x, y) \\ \mathtt{quot}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{minus}^\sharp(x, y), \mathtt{s}(y) \\ \mathtt{quot}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{quot}^\sharp(\mathtt{minus}(x, y), \mathtt{s}(y)) \end{array}$$

Splitting by root symbol

Recall:

$$\begin{aligned}
\mathtt{minus}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) &\to \mathtt{minus}^\sharp(x, y) \\
\mathtt{quot}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) &\to \mathtt{minus}^\sharp(x, y), \mathtt{s}(y) \\
\mathtt{quot}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) &\to \mathtt{quot}^\sharp(\mathtt{minus}(x, y), \mathtt{s}(y))
\end{aligned}$$

Observation: In an infinite chain, if ever we encounter a root symbol $\mathtt{minus}^\sharp$ the root symbol never becomes $\mathtt{quot}^\sharp$ again!

Splitting by root symbol

Recall:

$$\begin{array}{rcl} \mathtt{minus}^\sharp(\mathtt{s}(x),\mathtt{s}(y)) & \to & \mathtt{minus}^\sharp(x,y) \\ \mathtt{quot}^\sharp(\mathtt{s}(x),\mathtt{s}(y)) & \to & \mathtt{minus}^\sharp(x,y),\mathtt{s}(y)) \\ \mathtt{quot}^\sharp(\mathtt{s}(x),\mathtt{s}(y)) & \to & \mathtt{quot}^\sharp(\mathtt{minus}(x,y),\mathtt{s}(y)) \end{array}$$

Observation: In an infinite chain, if ever we encounter a root symbol $\mathtt{minus}^\sharp$ the root symbol never becomes $\mathtt{quot}^\sharp$ again!

More general:

Splitting by root symbol

Recall:

$$\begin{aligned} \mathtt{minus}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) &\rightarrow \mathtt{minus}^\sharp(x, y) \\ \mathtt{quot}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) &\rightarrow \mathtt{minus}^\sharp(x, y), \mathtt{s}(y)) \\ \mathtt{quot}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) &\rightarrow \mathtt{quot}^\sharp(\mathtt{minus}(x, y), \mathtt{s}(y)) \end{aligned}$$

Observation: In an infinite chain, if ever we encounter a root symbol $\mathtt{minus}^\sharp$ the root symbol never becomes $\mathtt{quot}^\sharp$ again!

More general:

- Consider: which pairs can follow each other in a chain?

Splitting by root symbol

Recall:

$$\begin{array}{rcl} \mathtt{minus}^\sharp(\mathtt{s}(x),\mathtt{s}(y)) & \to & \mathtt{minus}^\sharp(x,y) \\ \mathtt{quot}^\sharp(\mathtt{s}(x),\mathtt{s}(y)) & \to & \mathtt{minus}^\sharp(x,y),\mathtt{s}(y)) \\ \mathtt{quot}^\sharp(\mathtt{s}(x),\mathtt{s}(y)) & \to & \mathtt{quot}^\sharp(\mathtt{minus}(x,y),\mathtt{s}(y)) \end{array}$$

Observation: In an infinite chain, if ever we encounter a root symbol $\mathtt{minus}^\sharp$ the root symbol never becomes $\mathtt{quot}^\sharp$ again!

More general:

- Consider: which pairs can follow each other in a chain?
- Split the DPs into groups that may follow each other!

Splitting call groups method

Splitting call groups method

Idea:

- Given: a set of dependency pairs
- Create: blue and red subsets $A_1, \ldots, A_n$ such that:

Splitting call groups method

Idea:

- Given: a set of dependency pairs
- Create: blue and red subsets $A_1, \dots, A_n$ such that:
  - a pair in A_i can only be followed in a chain by a pair in $A_0, A_1, \dots, A_i$

Splitting call groups method

Idea:

- Given: a set of dependency pairs
- Create: blue and red subsets $A_1, \dots, A_n$ such that:
  - a pair in A_i can only be followed in a chain by a pair in $A_0, A_1, \dots, A_i$
  - if A_i is red, then it cannot be followed by a pair in A_i either

Splitting call groups method

Idea:

- Given: a set of dependency pairs
- Create: blue and red subsets $A_1, \dots, A_n$ such that:
  - a pair in A_i can only be followed in a chain by a pair in $A_0, A_1, \dots, A_i$
  - if A_i is red, then it cannot be followed by a pair in A_i either
- Then: it suffices to prove that there is no chain over each blue subset!

Running example

$$
\begin{array}{rcl}
\mathtt{minus}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{minus}^\sharp(x, y) \\
\mathtt{quot}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{minus}^\sharp(x, y) \\
\mathtt{quot}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{quot}^\sharp(\mathtt{minus}(x, y), \mathtt{s}(y)) \\
\mathtt{ack}^\sharp(\mathtt{s}(x), \mathtt{0}) & \to & \mathtt{ack}^\sharp(x, \mathtt{s}(\mathtt{0})) \\
\mathtt{ack}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{ack}^\sharp(\mathtt{s}(x), y) \\
\mathtt{ack}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{ack}^\sharp(x, \mathtt{ack}(\mathtt{s}(x), y)) \\
\mathtt{inc}^\sharp(\mathtt{0}) & \to & \mathtt{inc}^\sharp(\mathtt{s}(\mathtt{0})) \\
\mathtt{fexp}^\sharp(\mathtt{s}(x), y) & \to & \mathtt{double}^\sharp(x, y, \mathtt{0}) \\
\mathtt{double}^\sharp(x, \mathtt{0}, z) & \to & \mathtt{fexp}^\sharp(x, z) \\
\mathtt{double}^\sharp(x, \mathtt{s}(y), z) & \to & \mathtt{double}^\sharp(x, y, \mathtt{s}(\mathtt{s}(z))) \\
\mathtt{len}^\sharp(\mathtt{cons}(x, l)) & \to & \mathtt{len}^\sharp(l) \\
\mathtt{map}^\sharp(F, \mathtt{cons}(x, l)) & \to & \mathtt{map}^\sharp(F, l) \\
\mathtt{fold}^\sharp(F, x, \mathtt{cons}(y, l)) & \to & \mathtt{fold}^\sharp(F, F \cdot x \cdot y, l) \\
\mathtt{mkbig}^\sharp(l, x) & \to & \mathtt{ack}^\sharp(x, y) \\
\mathtt{mkbig}^\sharp(l, x) & \to & \mathtt{map}^\sharp(\mathtt{ack}(x), l) \\
\mathtt{mkdiv}^\sharp(l, x) & \to & \mathtt{quot}^\sharp(y, x) \\
\mathtt{mkdiv}^\sharp(l, x) & \to & \mathtt{map}^\sharp(\lambda y.\mathtt{quot}(y, x), l) \\
\mathtt{sma}^\sharp(\mathtt{false}, F, \mathtt{s}(x)) & \to & \mathtt{quot}^\sharp(x, \mathtt{s}\ (\mathtt{s}\ \mathtt{0})) \\
\mathtt{sma}^\sharp(\mathtt{false}, F, \mathtt{s}(x)) & \to & \mathtt{sma}^\sharp(F \cdot x, F, \mathtt{quot}(x, \mathtt{s}\ (\mathtt{s}\ \mathtt{0}))) \\
\mathtt{H}^\sharp(\mathtt{s}(x)) & \to & \mathtt{I}^\sharp(y) \\
\mathtt{H}^\sharp(\mathtt{s}(x)) & \to & \mathtt{twice}^\sharp(\mathtt{I}, x) \\
\mathtt{H}^\sharp(\mathtt{s}(x)) & \to & \mathtt{H}^\sharp(\mathtt{twice}(\mathtt{I}, x))
\end{array}
$$

Running example

A_1	$\mathtt{minus}^\sharp(\mathtt{s}(x), \mathtt{s}(y))$	$\to$	$\mathtt{minus}^\sharp(x, y)$
A_2	$\mathtt{quot}^\sharp(\mathtt{s}(x), \mathtt{s}(y))$	$\to$	$\mathtt{minus}^\sharp(x, y)$
A_3	$\mathtt{quot}^\sharp(\mathtt{s}(x), \mathtt{s}(y))$	$\to$	$\mathtt{quot}^\sharp(\mathtt{minus}(x, y), \mathtt{s}(y))$
A_4	$\mathtt{ack}^\sharp(\mathtt{s}(x), \mathtt{0})$	$\to$	$\mathtt{ack}^\sharp(x, \mathtt{s}(\mathtt{0}))$
	$\mathtt{ack}^\sharp(\mathtt{s}(x), \mathtt{s}(y))$	$\to$	$\mathtt{ack}^\sharp(\mathtt{s}(x), y)$
	$\mathtt{ack}^\sharp(\mathtt{s}(x), \mathtt{s}(y))$	$\to$	$\mathtt{ack}^\sharp(x, \mathtt{ack}(\mathtt{s}(x), y))$
A_5	$\mathtt{inc}^\sharp(\mathtt{0})$	$\to$	$\mathtt{inc}^\sharp(\mathtt{s}(\mathtt{0}))$
A_6	$\mathtt{fexp}^\sharp(\mathtt{s}(x), y)$	$\to$	$\mathtt{double}^\sharp(x, y, \mathtt{0})$
	$\mathtt{double}^\sharp(x, \mathtt{0}, z)$	$\to$	$\mathtt{fexp}^\sharp(x, z)$
	$\mathtt{double}^\sharp(x, \mathtt{s}(y), z)$	$\to$	$\mathtt{double}^\sharp(x, y, \mathtt{s}(\mathtt{s}(z)))$
A_7	$\mathtt{len}^\sharp(\mathtt{cons}(x, l))$	$\to$	$\mathtt{len}^\sharp(l)$
A_8	$\mathtt{map}^\sharp(F, \mathtt{cons}(x, l))$	$\to$	$\mathtt{map}^\sharp(F, l)$
A_9	$\mathtt{fold}^\sharp(F, x, \mathtt{cons}(y, l))$	$\to$	$\mathtt{fold}^\sharp(F, F \cdot x \cdot y, l)$
A_{10}	$\mathtt{mkbig}^\sharp(l, x)$	$\to$	$\mathtt{ack}^\sharp(x, y)$
	$\mathtt{mkbig}^\sharp(l, x)$	$\to$	$\mathtt{map}^\sharp(\mathtt{ack}(x), l)$
	$\mathtt{mkdiv}^\sharp(l, x)$	$\to$	$\mathtt{quot}^\sharp(y, x)$
	$\mathtt{mkdiv}^\sharp(l, x)$	$\to$	$\mathtt{map}^\sharp(\lambda y.\mathtt{quot}(y, x), l)$
	$\mathtt{sma}^\sharp(\mathtt{false}, F, \mathtt{s}(x))$	$\to$	$\mathtt{quot}^\sharp(x, \mathtt{s}\ (\mathtt{s}\ \mathtt{0}))$
A_{11}	$\mathtt{sma}^\sharp(\mathtt{false}, F, \mathtt{s}(x))$	$\to$	$\mathtt{sma}^\sharp(F \cdot x, F, \mathtt{quot}(x, \mathtt{s}\ (\mathtt{s}\ \mathtt{0})))$
A_{12}	$\mathtt{H}^\sharp(\mathtt{s}(x))$	$\to$	$\mathtt{I}^\sharp(y)$
	$\mathtt{H}^\sharp(\mathtt{s}(x))$	$\to$	$\mathtt{twice}^\sharp(\mathtt{I}, x)$
A_{13}	$\mathtt{H}^\sharp(\mathtt{s}(x))$	$\to$	$\mathtt{H}^\sharp(\mathtt{twice}(\mathtt{I}, x))$

Running example

A_1	$\texttt{minus}^\sharp(\texttt{s}(x), \texttt{s}(y))$	$\rightarrow$	$\texttt{minus}^\sharp(x, y)$
A_3	$\texttt{quot}^\sharp(\texttt{s}(x), \texttt{s}(y))$	$\rightarrow$	$\texttt{quot}^\sharp(\texttt{minus}(x, y), \texttt{s}(y))$
A_4	$\texttt{ack}^\sharp(\texttt{s}(x), \texttt{0})$	$\rightarrow$	$\texttt{ack}^\sharp(x, \texttt{s}(\texttt{0}))$
	$\texttt{ack}^\sharp(\texttt{s}(x), \texttt{s}(y))$	$\rightarrow$	$\texttt{ack}^\sharp(\texttt{s}(x), y)$
	$\texttt{ack}^\sharp(\texttt{s}(x), \texttt{s}(y))$	$\rightarrow$	$\texttt{ack}^\sharp(x, \texttt{ack}(\texttt{s}(x), y))$
A_6	$\texttt{fexp}^\sharp(\texttt{s}(x), y)$	$\rightarrow$	$\texttt{double}^\sharp(x, y, \texttt{0})$
	$\texttt{double}^\sharp(x, \texttt{0}, z)$	$\rightarrow$	$\texttt{fexp}^\sharp(x, z)$
	$\texttt{double}^\sharp(x, \texttt{s}(y), z)$	$\rightarrow$	$\texttt{double}^\sharp(x, y, \texttt{s}(\texttt{s}(z)))$
A_7	$\texttt{len}^\sharp(\texttt{cons}(x, l))$	$\rightarrow$	$\texttt{len}^\sharp(l)$
A_8	$\texttt{map}^\sharp(F, \texttt{cons}(x, l))$	$\rightarrow$	$\texttt{map}^\sharp(F, l)$
A_9	$\texttt{fold}^\sharp(F, x, \texttt{cons}(y, l))$	$\rightarrow$	$\texttt{fold}^\sharp(F, F \cdot x \cdot y, l)$
A_{11}	$\texttt{sma}^\sharp(\texttt{false}, F, \texttt{s}(x))$	$\rightarrow$	$\texttt{sma}^\sharp(F \cdot x, F, \texttt{quot}(x, \texttt{s}\ (\texttt{s}\ \texttt{0})))$
A_{13}	$\texttt{H}^\sharp(\texttt{s}(x))$	$\rightarrow$	$\texttt{H}^\sharp(\texttt{twice}(\texttt{I}, x))$

Alternative formulation: DP graph

Alternative formulation: DP graph

- Make a graph whose vertices are the elements of $\mathcal{P}$

Alternative formulation: DP graph

- Make a graph whose vertices are the elements of $\mathcal{P}$
- Place an edge from ρ_1 to ρ_2 if ρ_2 may follow ρ_1 in a graph

Alternative formulation: DP graph

- Make a graph whose vertices are the elements of $\mathcal{P}$
- Place an edge from ρ_1 to ρ_2 if ρ_2 may follow ρ_1 in a graph
- Split up $\mathcal{P}$ into the strongly connected components of the graph

Alternative formulation: DP graph

- Make a graph whose vertices are the elements of $\mathcal{P}$
- Place an edge from ρ_1 to ρ_2 if ρ_2 may follow ρ_1 in a graph
- Split up $\mathcal{P}$ into the strongly connected components of the graph

Claim: This is the same method.

Alternative formulation: DP graph

- Make a graph whose vertices are the elements of $\mathcal{P}$
- Place an edge from ρ_1 to ρ_2 if ρ_2 may follow ρ_1 in a graph
- Split up $\mathcal{P}$ into the strongly connected components of the graph

Claim: This is the same method.

- The graph is natural for automation .

Alternative formulation: DP graph

- Make a graph whose vertices are the elements of $\mathcal{P}$
- Place an edge from ρ_1 to ρ_2 if ρ_2 may follow ρ_1 in a graph
- Split up $\mathcal{P}$ into the strongly connected components of the graph

Claim: This is the same method.

- The graph is natural for automation .
- The groups approach is natural for certification .

Example

$$\begin{array}{lrcl}
(1) & \mathtt{map}^\sharp(F, \mathtt{cons}(x, l)) & \rightarrow & \mathtt{map}^\sharp(F, l) \\
(2) & \mathtt{double}^\sharp(l) & \rightarrow & \mathtt{map}^\sharp(\lambda x.\mathtt{add}(x, x), l) \\
(3) & \mathtt{double}^\sharp(l) & \rightarrow & \mathtt{add}^\sharp(x, x) \\
(4) & \mathtt{add}^\sharp(\mathtt{s}(x), y) & \rightarrow & \mathtt{add}^\sharp(x, \mathtt{s}(y))
\end{array}$$

Example

$$
\begin{array}{llcl}
(1) & \mathtt{map}^\sharp(F, \mathtt{cons}(x, l)) & \to & \mathtt{map}^\sharp(F, l) \\
(2) & \mathtt{double}^\sharp(l) & \to & \mathtt{map}^\sharp(\lambda x.\mathtt{add}(x, x), l) \\
(3) & \mathtt{double}^\sharp(l) & \to & \mathtt{add}^\sharp(x, x) \\
(4) & \mathtt{add}^\sharp(\mathtt{s}(x), y) & \to & \mathtt{add}^\sharp(x, \mathtt{s}(y))
\end{array}
$$

Example

$$
\begin{array}{llcl}
(1) & \mathtt{map}^\sharp(F, \mathtt{cons}(x, l)) & \rightarrow & \mathtt{map}^\sharp(F, l) \\
(2) & \mathtt{double}^\sharp(l) & \rightarrow & \mathtt{map}^\sharp(\lambda x.\mathtt{add}(x, x), l) \\
(3) & \mathtt{double}^\sharp(l) & \rightarrow & \mathtt{add}^\sharp(x, x) \\
(4) & \mathtt{add}^\sharp(\mathtt{s}(x), y) & \rightarrow & \mathtt{add}^\sharp(x, \mathtt{s}(y))
\end{array}
$$

Example

(1)	$\mathtt{map}^\sharp(F, \mathtt{cons}(x, l))$	$\rightarrow$	$\mathtt{map}^\sharp(F, l)$
(2)	$\mathtt{double}^\sharp(l)$	$\rightarrow$	$\mathtt{map}^\sharp(\lambda x.\mathtt{add}(x, x), l)$
(3)	$\mathtt{double}^\sharp(l)$	$\rightarrow$	$\mathtt{add}^\sharp(x, x)$
(4)	$\mathtt{add}^\sharp(\mathtt{s}(x), y)$	$\rightarrow$	$\mathtt{add}^\sharp(x, \mathtt{s}(y))$

Example

$$
\begin{array}{llcl}
(1) & \mathtt{map}^\sharp(F, \mathtt{cons}(x, l)) & \to & \mathtt{map}^\sharp(F, l) \\
(2) & \mathtt{double}^\sharp(l) & \to & \mathtt{map}^\sharp(\lambda x.\mathtt{add}(x, x), l) \\
(3) & \mathtt{double}^\sharp(l) & \to & \mathtt{add}^\sharp(x, x) \\
(4) & \mathtt{add}^\sharp(\mathtt{s}(x), y) & \to & \mathtt{add}^\sharp(x, \mathtt{s}(y))
\end{array}
$$

Result: the DP problem $(\mathtt{DP}(\mathcal{R}), \mathcal{R})$ is finite if:

- the DP problem $(\{(1)\}, \mathcal{R})$ is finite;
- the DP problem $(\{(4)\}, \mathcal{R})$ is finite.

Exercises:

1. Compute the dependency pairs of the following HTRS, and divide them into call groups. (You may use a graph. Types are as expected, with sorts nat and bool.)

$$\begin{array}{rcl}
\mathtt{comp2}(\mathtt{0}, \mathtt{s}(y)) & \to & \mathtt{false} \\
\mathtt{comp2}(\mathtt{s}(\mathtt{0}), \mathtt{s}(y)) & \to & \mathtt{false} \\
\mathtt{comp2}(x, \mathtt{0}) & \to & \mathtt{true} \\
\mathtt{comp2}(\mathtt{s}(\mathtt{s}(x)), \mathtt{s}(y)) & \to & \mathtt{comp2}(x, y) \\
\mathtt{find}(F, x, \mathtt{false}) & \to & \mathtt{end}(x) \\
\mathtt{find}(F, x, \mathtt{true}) & \to & \mathtt{find}(F, \mathtt{s}(x), \mathtt{comp2}(F \cdot x, x)) \\
\mathtt{double}(\mathtt{0}) & \to & \mathtt{0} \\
\mathtt{double}(\mathtt{s}(x)) & \to & \mathtt{s}(\mathtt{s}(\mathtt{double}(x)))
\end{array}$$

2. Compute the dependency pairs, and call groups, for the HTRS consisting only of Toyama's example (with $\mathtt{a}, \mathtt{b} :: \mathrm{o}$):

$$\mathtt{f}(\mathtt{a}, \mathtt{b}, x) \to \mathtt{f}(x, x, x)$$

The subterm criterion: intuition

Recall:

$$A_8 \quad \mathtt{map}^\sharp(F, \mathtt{cons}(x, l)) \quad \rightarrow \quad \mathtt{map}^\sharp(F, l)$$

Question: what does an infinite chain over A_8 look like?

The subterm criterion: intuition

Recall:

$$A_8 \quad \mathtt{map}^\sharp(F, \mathtt{cons}(x, l)) \quad \to \quad \mathtt{map}^\sharp(F, l)$$

Question: what does an infinite chain over A_8 look like?

$$\begin{array}{lll} \mathtt{map}^\sharp(u_1, \mathtt{cons}(v_1, w_1)) & \to_{A_8} & \mathtt{map}^\sharp(u_1, w_1) \\ & \to_{\mathcal{R}}^* & \mathtt{map}^\sharp(u_2, \mathtt{cons}(v_2, w_2)) \\ & \to_{A_8} & \mathtt{map}^\sharp(u_2, w_2) \\ & \to_{\mathcal{R}}^* & \cdots \end{array}$$

The subterm criterion: intuition

Recall:

$$A_8 \quad \mathtt{map}^\sharp(F, \mathtt{cons}(x, l)) \;\rightarrow\; \mathtt{map}^\sharp(F, l)$$

Question: what does an infinite chain over A_8 look like?

$$\begin{array}{lll} \mathtt{map}^\sharp(u_1, \mathtt{cons}(v_1, w_1)) & \rightarrow_{A_8} & \mathtt{map}^\sharp(u_1, w_1) \\ & \rightarrow^*_{\mathcal{R}} & \mathtt{map}^\sharp(u_2, \mathtt{cons}(v_2, w_2)) \\ & \rightarrow_{A_8} & \mathtt{map}^\sharp(u_2, w_2) \\ & \rightarrow^*_{\mathcal{R}} & \dots \end{array}$$

Idea: look at the second argument of map

The subterm criterion: intuition

Recall:

$$A_8 \quad \mathtt{map}^\sharp(F, \mathtt{cons}(x, l)) \quad \rightarrow \quad \mathtt{map}^\sharp(F, l)$$

Question: what does an infinite chain over A_8 look like?

$$\begin{array}{lll} \mathtt{map}^\sharp(u_1, \mathtt{cons}(v_1, w_1)) & \rightarrow_{A_8} & \mathtt{map}^\sharp(u_1, w_1) \\ & \rightarrow^*_{\mathcal{R}} & \mathtt{map}^\sharp(u_2, \mathtt{cons}(v_2, w_2)) \\ & \rightarrow_{A_8} & \mathtt{map}^\sharp(u_2, w_2) \\ & \rightarrow^*_{\mathcal{R}} & \dots \end{array}$$

Idea: look at the second argument of `map` (which is computable by assumption).

The subterm criterion: intuition

Recall:

$$A_8 \quad \mathtt{map}^\sharp(F, \mathtt{cons}(x, l)) \quad \to \quad \mathtt{map}^\sharp(F, l)$$

Question: what does an infinite chain over A_8 look like?

$$\begin{array}{lll} \mathtt{map}^\sharp(u_1, \mathtt{cons}(v_1, w_1)) & \to_{A_8} & \mathtt{map}^\sharp(u_1, w_1) \\ & \to_{\mathcal{R}}^* & \mathtt{map}^\sharp(u_2, \mathtt{cons}(v_2, w_2)) \\ & \to_{A_8} & \mathtt{map}^\sharp(u_2, w_2) \\ & \to_{\mathcal{R}}^* & \dots \end{array}$$

Idea: look at the second argument of $\mathtt{map}$ (which is computable by assumption).

$$\mathtt{cons}(v_1, w_1)) \rhd w_1 \to_{\mathcal{R}}^* \mathtt{cons}(v_2, w_2) \rhd w_2 \to_{\mathcal{R}}^* \dots$$

The subterm criterion: intuition

Recall:

$$A_8 \quad \mathtt{map}^\sharp(F, \mathtt{cons}(x, l)) \quad \to \quad \mathtt{map}^\sharp(F, l)$$

Question: what does an infinite chain over A_8 look like?

$$\begin{array}{lll} \mathtt{map}^\sharp(u_1, \mathtt{cons}(v_1, w_1)) & \to_{A_8} & \mathtt{map}^\sharp(u_1, w_1) \\ & \to_{\mathcal{R}}^* & \mathtt{map}^\sharp(u_2, \mathtt{cons}(v_2, w_2)) \\ & \to_{A_8} & \mathtt{map}^\sharp(u_2, w_2) \\ & \to_{\mathcal{R}}^* & \dots \end{array}$$

Idea: look at the second argument of $\mathtt{map}$ (which is computable by assumption).

$$\mathtt{cons}(v_1, w_1)) \rhd w_1 \to_{\mathcal{R}}^* \mathtt{cons}(v_2, w_2) \rhd w_2 \to_{\mathcal{R}}^* \dots$$

Observation: this contradicts termination, and therefore computability!

The subterm criterion: definition

Given: $(\mathcal{P}, \mathcal{R})$ with marked symbols $\mathtt{f}_1^\sharp, \ldots, \mathtt{f}_n^\sharp$

The subterm criterion: definition

Given: $(\mathcal{P}, \mathcal{R})$ with marked symbols $\mathtt{f}_1^\sharp, \dots, \mathtt{f}_n^\sharp$

Choose: for each $\mathtt{f}_i^\sharp$, one argument position $\nu(\mathtt{f}_i^\sharp)$

The subterm criterion: definition

Given: $(\mathcal{P}, \mathcal{R})$ with marked symbols $\mathtt{f}_1^\sharp, \dots, \mathtt{f}_n^\sharp$

Choose: for each $\mathtt{f}_i^\sharp$, one argument position $\nu(\mathtt{f}_i^\sharp)$

Show: for every DP $\mathtt{f}_i^\sharp(\ell_1, \dots, \ell_k) \to \mathtt{f}_j^\sharp(r_1, \dots, r_n)$:

- either $\ell_{\nu(\mathtt{f}_i^\sharp)} \rhd r_{\nu(\mathtt{f}_j^\sharp)}$
- or $\ell_{\nu(\mathtt{f}_i^\sharp)} = r_{\nu(\mathtt{f}_j^\sharp)}$

The subterm criterion: definition

Given: $(\mathcal{P}, \mathcal{R})$ with marked symbols $\mathtt{f}_1^\sharp, \dots, \mathtt{f}_n^\sharp$

Choose: for each $\mathtt{f}_i^\sharp$, one argument position $\nu(\mathtt{f}_i^\sharp)$

Show: for every DP $\mathtt{f}_i^\sharp(\ell_1, \dots, \ell_k) \to \mathtt{f}_j^\sharp(r_1, \dots, r_n)$:

- either $\ell_{\nu(\mathtt{f}_i^\sharp)} \rhd r_{\nu(\mathtt{f}_j^\sharp)}$
- or $\ell_{\nu(\mathtt{f}_i^\sharp)} = r_{\nu(\mathtt{f}_j^\sharp)}$

Then: remove from $\mathcal{P}$ all the DPs where we used $\rhd$.

The subterm criterion: definition

Given: $(\mathcal{P}, \mathcal{R})$ with marked symbols $\mathtt{f}_1^\sharp, \dots, \mathtt{f}_n^\sharp$

Choose: for each $\mathtt{f}_i^\sharp$, one argument position $\nu(\mathtt{f}_i^\sharp)$

Show: for every DP $\mathtt{f}_i^\sharp(\ell_1, \dots, \ell_k) \to \mathtt{f}_j^\sharp(r_1, \dots, r_n)$:

- either $\ell_{\nu(\mathtt{f}_i^\sharp)} \rhd r_{\nu(\mathtt{f}_j^\sharp)}$
- or $\ell_{\nu(\mathtt{f}_i^\sharp)} = r_{\nu(\mathtt{f}_j^\sharp)}$

Then: remove from $\mathcal{P}$ all the DPs where we used $\rhd$.

Soundness proof: in any infinite computable chain, only finitely many $\rhd$ steps can be done. Hence, any such chain must have an infinite tail without $\rhd$ steps.

Examples

A_1	$\mathtt{minus}^\sharp(\mathtt{s}(x), \mathtt{s}(y))$	$\to$	$\mathtt{minus}^\sharp(x, y)$
A_3	$\mathtt{quot}^\sharp(\mathtt{s}(x), \mathtt{s}(y))$	$\to$	$\mathtt{quot}^\sharp(\mathtt{minus}(x, y), \mathtt{s}(y))$
A_4	$\mathtt{ack}^\sharp(\mathtt{s}(x), \mathtt{0})$	$\to$	$\mathtt{ack}^\sharp(x, \mathtt{s}(\mathtt{0}))$
	$\mathtt{ack}^\sharp(\mathtt{s}(x), \mathtt{s}(y))$	$\to$	$\mathtt{ack}^\sharp(\mathtt{s}(x), y)$
	$\mathtt{ack}^\sharp(\mathtt{s}(x), \mathtt{s}(y))$	$\to$	$\mathtt{ack}^\sharp(x, \mathtt{ack}(\mathtt{s}(x), y))$
A_6	$\mathtt{fexp}^\sharp(\mathtt{s}(x), y)$	$\to$	$\mathtt{double}^\sharp(x, y, \mathtt{0})$
	$\mathtt{double}^\sharp(x, \mathtt{0}, z)$	$\to$	$\mathtt{fexp}^\sharp(x, z)$
	$\mathtt{double}^\sharp(x, \mathtt{s}(y), z)$	$\to$	$\mathtt{double}^\sharp(x, y, \mathtt{s}(\mathtt{s}(z)))$
A_7	$\mathtt{len}^\sharp(\mathtt{cons}(x, l))$	$\to$	$\mathtt{len}^\sharp(l)$
A_8	$\mathtt{map}^\sharp(F, \mathtt{cons}(x, l))$	$\to$	$\mathtt{map}^\sharp(F, l)$
A_9	$\mathtt{fold}^\sharp(F, x, \mathtt{cons}(y, l))$	$\to$	$\mathtt{fold}^\sharp(F, F \cdot x \cdot y, l)$
A_{11}	$\mathtt{sma}^\sharp(\mathtt{false}, F, \mathtt{s}(x))$	$\to$	$\mathtt{sma}^\sharp(F \cdot x, F, \mathtt{quot}(x, \mathtt{s}\ (\mathtt{s}\ \mathtt{0})))$
A_{13}	$\mathtt{H}^\sharp(\mathtt{s}(x))$	$\to$	$\mathtt{H}^\sharp(\mathtt{twice}(\mathtt{I}, x))$

Examples: A_1

$$A_1 \quad \mathtt{minus}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) \quad \rightarrow \quad \mathtt{minus}^\sharp(x, y)$$

Examples: A_1

$$A_1 \quad \mathtt{minus}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) \quad \rightarrow \quad \mathtt{minus}^\sharp(x, y)$$

Argument position: $\nu(\mathtt{minus}^\sharp) = 2$

Examples: A_1

$$A_1 \quad \mathtt{minus}^\sharp(\mathtt{s}(x), \underline{\mathtt{s}(y)}) \;\to\; \mathtt{minus}^\sharp(x, \underline{y})$$

Argument position: $\nu(\mathtt{minus}^\sharp) = 2$

Examples: A_3

$$A_3 \quad \mathtt{quot}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) \quad \to \quad \mathtt{quot}^\sharp(\mathtt{minus}(x, y), \mathtt{s}(y))$$

Examples: A_3

$$A_3 \quad \mathtt{quot}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) \;\rightarrow\; \mathtt{quot}^\sharp(\mathtt{minus}(x, y), \mathtt{s}(y))$$

Argument position: method does not apply

Examples: A_4

$$
\begin{array}{lrcl}
A_4 & \mathtt{ack}^\sharp(\mathtt{s}(x), \mathtt{0}) & \to & \mathtt{ack}^\sharp(x, \mathtt{s}(\mathtt{0})) \\
 & \mathtt{ack}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{ack}^\sharp(\mathtt{s}(x), y) \\
 & \mathtt{ack}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{ack}^\sharp(x, \mathtt{ack}(\mathtt{s}(x), y))
\end{array}
$$

Examples: A_4

$$
\begin{array}{lrcl}
A_4 & \mathtt{ack}^\sharp(\mathtt{s}(x), \mathtt{0}) & \to & \mathtt{ack}^\sharp(x, \mathtt{s}(\mathtt{0})) \\
& \mathtt{ack}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{ack}^\sharp(\mathtt{s}(x), y) \\
& \mathtt{ack}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{ack}^\sharp(x, \mathtt{ack}(\mathtt{s}(x), y))
\end{array}
$$

Argument position: $\nu(\mathtt{ack}^\sharp) = 1$

Examples: A_4

$$
\begin{array}{lrcl}
A_4 & \mathtt{ack}^\sharp(\underline{\mathtt{s}(x)}, \mathtt{0}) & \to & \mathtt{ack}^\sharp(\underline{x}, \mathtt{s}(\mathtt{0})) \\
& \mathtt{ack}^\sharp(\underline{\mathtt{s}(x)}, \mathtt{s}(y)) & \to & \mathtt{ack}^\sharp(\underline{\mathtt{s}(x)}, y) \\
& \mathtt{ack}^\sharp(\underline{\mathtt{s}(x)}, \mathtt{s}(y)) & \to & \mathtt{ack}^\sharp(\underline{x}, \mathtt{ack}(\mathtt{s}(x), y))
\end{array}
$$

Argument position: $\nu(\mathtt{ack}^\sharp) = 1$

Examples: A_4

$$A_4 \quad \begin{array}{rcl} \mathtt{ack}^\sharp(\mathtt{s}(x), \mathtt{0}) & \to & \mathtt{ack}^\sharp(x, \mathtt{s}(\mathtt{0})) \\ \mathtt{ack}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{ack}^\sharp(\mathtt{s}(x), y) \\ \mathtt{ack}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{ack}^\sharp(x, \mathtt{ack}(\mathtt{s}(x), y)) \end{array}$$

Argument position: $\nu(\mathtt{ack}^\sharp) = 1$

Remaining:

$$\mathtt{ack}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) \to \mathtt{ack}^\sharp(\mathtt{s}(x), y)$$

Examples: A_4

$$
A_4 \quad
\begin{array}{rcl}
\mathtt{ack}^\sharp(\mathtt{s}(x), \mathtt{0}) & \to & \mathtt{ack}^\sharp(x, \mathtt{s}(\mathtt{0})) \\
\mathtt{ack}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{ack}^\sharp(\mathtt{s}(x), y) \\
\mathtt{ack}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{ack}^\sharp(x, \mathtt{ack}(\mathtt{s}(x), y))
\end{array}
$$

Argument position: $\nu(\mathtt{ack}^\sharp) = 1$

Remaining:

$$
\mathtt{ack}^\sharp(\mathtt{s}(x), \underline{\mathtt{s}(y)}) \to \mathtt{ack}^\sharp(\mathtt{s}(x), \underline{y})
$$

Argument position: $\nu(\mathtt{ack}^\sharp) = 2$

Examples: A_6

$$A_6 \qquad \begin{array}{rcl} \mathtt{fexp}^\sharp(\mathtt{s}(x), y) & \to & \mathtt{double}^\sharp(x, y, \mathtt{0}) \\ \mathtt{double}^\sharp(x, \mathtt{0}, z) & \to & \mathtt{fexp}^\sharp(x, z) \\ \mathtt{double}^\sharp(x, \mathtt{s}(y), z) & \to & \mathtt{double}^\sharp(x, y, \mathtt{s}(\mathtt{s}(z))) \end{array}$$

Examples: A_6

$$A_6 \quad \begin{array}{rcl} \mathtt{fexp}^\sharp(\mathtt{s}(x), y) & \to & \mathtt{double}^\sharp(x, y, \mathtt{0}) \\ \mathtt{double}^\sharp(x, \mathtt{0}, z) & \to & \mathtt{fexp}^\sharp(x, z) \\ \mathtt{double}^\sharp(x, \mathtt{s}(y), z) & \to & \mathtt{double}^\sharp(x, y, \mathtt{s}(\mathtt{s}(z))) \end{array}$$

Argument positions:

- $\nu(\mathtt{fexp}^\sharp) = 1$
- $\nu(\mathtt{double}^\sharp) = 1$

Examples: A_6

$$A_6 \qquad \begin{array}{rcl} \mathtt{fexp}^\sharp(\underline{\mathtt{s}(x)}, y) & \to & \mathtt{double}^\sharp(\underline{x}, y, \mathtt{0}) \\ \mathtt{double}^\sharp(\underline{x}, \mathtt{0}, z) & \to & \mathtt{fexp}^\sharp(\underline{x}, z) \\ \mathtt{double}^\sharp(\underline{x}, \mathtt{s}(y), z) & \to & \mathtt{double}^\sharp(\underline{x}, y, \mathtt{s}(\mathtt{s}(z))) \end{array}$$

Argument positions:

- $\nu(\mathtt{fexp}^\sharp) = 1$
- $\nu(\mathtt{double}^\sharp) = 1$

Examples: A_6

$$
\begin{array}{llcl}
A_6 & \mathtt{fexp}^\sharp(\mathtt{s}(x), y) & \to & \mathtt{double}^\sharp(x, y, \mathtt{0}) \\
 & \mathtt{double}^\sharp(x, \mathtt{0}, z) & \to & \mathtt{fexp}^\sharp(x, z) \\
 & \mathtt{double}^\sharp(x, \mathtt{s}(y), z) & \to & \mathtt{double}^\sharp(x, y, \mathtt{s}(\mathtt{s}(z)))
\end{array}
$$

Argument positions:

- $\nu(\mathtt{fexp}^\sharp) = 1$
- $\nu(\mathtt{double}^\sharp) = 1$

Remaining:

$$
\begin{array}{lcl}
\mathtt{double}^\sharp(x, \mathtt{0}, z) & \to & \mathtt{fexp}^\sharp(x, z) \\
\mathtt{double}^\sharp(x, \mathtt{s}(y), z) & \to & \mathtt{double}^\sharp(x, y, \mathtt{s}(\mathtt{s}(z)))
\end{array}
$$

Examples: A_6

$$A_6 \quad \begin{array}{rcl} \mathtt{fexp}^\sharp(\mathtt{s}(x), y) & \to & \mathtt{double}^\sharp(x, y, \mathtt{0}) \\ \mathtt{double}^\sharp(x, \mathtt{0}, z) & \to & \mathtt{fexp}^\sharp(x, z) \\ \mathtt{double}^\sharp(x, \mathtt{s}(y), z) & \to & \mathtt{double}^\sharp(x, y, \mathtt{s}(\mathtt{s}(z))) \end{array}$$

Argument positions:

- $\nu(\mathtt{fexp}^\sharp) = 1$
- $\nu(\mathtt{double}^\sharp) = 1$

Remaining:

$$\mathtt{double}^\sharp(x, \mathtt{s}(y), z) \to \mathtt{double}^\sharp(x, y, \mathtt{s}(\mathtt{s}(z)))$$

Examples: A_6

$$
A_6 \qquad
\begin{array}{rcl}
\mathtt{fexp}^\sharp(\mathtt{s}(x), y) & \to & \mathtt{double}^\sharp(x, y, \mathtt{0}) \\
\mathtt{double}^\sharp(x, \mathtt{0}, z) & \to & \mathtt{fexp}^\sharp(x, z) \\
\mathtt{double}^\sharp(x, \mathtt{s}(y), z) & \to & \mathtt{double}^\sharp(x, y, \mathtt{s}(\mathtt{s}(z)))
\end{array}
$$

Argument positions:

- $\nu(\mathtt{fexp}^\sharp) = 1$
- $\nu(\mathtt{double}^\sharp) = 1$

Remaining:

$$
\mathtt{double}^\sharp(x, \underline{\mathtt{s}(y)}, z) \to \mathtt{double}^\sharp(x, \underline{y}, \mathtt{s}(\mathtt{s}(z)))
$$

Argument position: $\nu(\mathtt{double}^\sharp) = 2$

Running example

$$A_3 = \{\mathtt{quot}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) \to \mathtt{quot}^\sharp(\mathtt{minus}(x, y), \mathtt{s}(y))\}$$

$$A_{11} = \{\mathtt{sma}^\sharp(\mathtt{false}, F, \mathtt{s}(x)) \to \mathtt{sma}^\sharp(F \cdot x, F, \mathtt{quot}(x, \mathtt{s}\ (\mathtt{s}\ \mathtt{0})))\}$$

$$A_{13} = \{\mathtt{H}^\sharp(\mathtt{s}(x)) \to \mathtt{H}^\sharp(\mathtt{twice}(\mathtt{I}, x))\}$$

First-order example

Consider:

$$
\begin{aligned}
\mathtt{minus}(x, \mathtt{0}) &\rightarrow x \\
\mathtt{minus}(\mathtt{s}(x), \mathtt{s}(y)) &\rightarrow \mathtt{minus}(x, y) \\
\mathtt{quot}(\mathtt{0}, \mathtt{s}(y)) &\rightarrow \mathtt{0} \\
\mathtt{quot}(\mathtt{s}(x), \mathtt{s}(y)) &\rightarrow \mathtt{s}(\mathtt{quot}(\mathtt{minus}(x, y), \mathtt{s}(y)))
\end{aligned}
$$

$$
\mathtt{quot}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) \rightarrow \mathtt{quot}^\sharp(\mathtt{minus}(x, y), \mathtt{s}(y))
$$

First-order example

Consider:

$$\begin{array}{rcl} \mathtt{minus}(x, \mathtt{0}) & \to & x \\ \mathtt{minus}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{minus}(x, y) \\ \mathtt{quot}(\mathtt{0}, \mathtt{s}(y)) & \to & \mathtt{0} \\ \mathtt{quot}(\mathtt{s}(x), \mathtt{s}(y)) & \to & \mathtt{s}(\mathtt{quot}(\mathtt{minus}(x, y), \mathtt{s}(y))) \end{array}$$

$$\mathtt{quot}^\sharp(\mathtt{s}(x), \mathtt{s}(y)) \to \mathtt{quot}^\sharp(\mathtt{minus}(x, y), \mathtt{s}(y))$$

Idea: look only at the first argument of each function symbol

First-order example

Consider:

$$
\begin{aligned}
\mathtt{minus}(x) &\to x \\
\mathtt{minus}(\mathtt{s}(x)) &\to \mathtt{minus}(x) \\
\mathtt{quot}(\mathtt{0}) &\to \mathtt{0} \\
\mathtt{quot}(\mathtt{s}(x)) &\to \mathtt{s}(\mathtt{quot}(\mathtt{minus}(x)))
\end{aligned}
$$

$$
\mathtt{quot}^\sharp(\mathtt{s}(x)) \to \mathtt{quot}^\sharp(\mathtt{minus}(x))
$$

Idea: look only at the first argument of each function symbol

First-order example

Consider:

$$\begin{array}{rcl} \mathtt{minus}(x) & \to & x \\ \mathtt{minus}(\mathtt{s}(x)) & \to & \mathtt{minus}(x) \\ \mathtt{quot}(\mathtt{0}) & \to & \mathtt{0} \\ \mathtt{quot}(\mathtt{s}(x)) & \to & \mathtt{s}(\mathtt{quot}(\mathtt{minus}(x))) \\ & & \\ \mathtt{quot}^\sharp(\mathtt{s}(x)) & \to & \mathtt{quot}^\sharp(\mathtt{minus}(x)) \end{array}$$

Idea: look only at the first argument of each function symbol

Observation: we can orient all rules and DPs together with LPO now!

Argument filtering

Argument filtering

Suppose:

Argument filtering

Suppose:

- Left-hand sides of rules have no subterm $x \cdot s_1 \cdots s_n$ or $(\lambda x.s_0) \cdot s_1 \cdots s_n$ with $n > 0$

Argument filtering

Suppose:

- Left-hand sides of rules have no subterm $x \cdot s_1 \cdots s_n$ or $(\lambda x.s_0) \cdot s_1 \cdots s_n$ with $n > 0$
- Each occurrence of $\mathtt{f}$ in $\mathcal{R}, \mathcal{P}$ has at least $N_{\mathtt{f}}$ arguments.

Argument filtering

Suppose:

- Left-hand sides of rules have no subterm $x \cdot s_1 \cdots s_n$ or $(\lambda x.s_0) \cdot s_1 \cdots s_n$ with $n > 0$
- Each occurrence of $\mathtt{f}$ in $\mathcal{R}, \mathcal{P}$ has at least $N_{\mathtt{f}}$ arguments.

Choose:

Argument filtering

Suppose:

- Left-hand sides of rules have no subterm $x \cdot s_1 \cdots s_n$ or $(\lambda x.s_0) \cdot s_1 \cdots s_n$ with $n > 0$
- Each occurrence of $\mathtt{f}$ in $\mathcal{R}, \mathcal{P}$ has at least $N_\mathtt{f}$ arguments.

Choose:

- a sequence $1 \leq i_1 < i_2 < \cdots < i_k \leq N_\mathtt{f}$ for each $\mathtt{f}$

Argument filtering

Suppose:

- Left-hand sides of rules have no subterm $x \cdot s_1 \cdots s_n$ or $(\lambda x.s_0) \cdot s_1 \cdots s_n$ with $n > 0$
- Each occurrence of $\mathtt{f}$ in $\mathcal{R}, \mathcal{P}$ has at least $N_{\mathtt{f}}$ arguments.

Choose:

- a sequence $1 \leq i_1 < i_2 < \cdots < i_k \leq N_{\mathtt{f}}$ for each $\mathtt{f}$

Define:

- $\overline{\nu}(\mathtt{f}(s_1, \ldots, s_n)) = \mathtt{f}(\overline{\nu}(s_{i_1}), \ldots, \overline{\nu}(s_{i_k}), \overline{\nu}(s_{N_{\mathtt{f}}+1}), \ldots, \overline{\nu}(s_n))$ if $n \geq N_{\mathtt{f}}$
- $\overline{\nu}(x \cdot s_1 \cdots s_n) = x \cdot \overline{\nu}(s_1) \cdots \overline{\nu}(s_n)$
- $\overline{\nu}((\lambda x.s_0) \cdot s_1 \cdots s_n) = (\lambda x.\overline{\nu}(s_0)) \cdot \overline{\nu}(s_1) \cdots \overline{\nu}(s_n)$

Argument filtering

Suppose:

- Left-hand sides of rules have no subterm $x \cdot s_1 \cdots s_n$ or $(\lambda x.s_0) \cdot s_1 \cdots s_n$ with $n > 0$
- Each occurrence of $\mathtt{f}$ in $\mathcal{R}, \mathcal{P}$ has at least $N_{\mathtt{f}}$ arguments.

Choose:

- a sequence $1 \leq i_1 < i_2 < \cdots < i_k \leq N_{\mathtt{f}}$ for each $\mathtt{f}$

Define:

- $\overline{\nu}(\mathtt{f}(s_1, \dots, s_n)) = \mathtt{f}(\overline{\nu}(s_{i_1}), \dots, \overline{\nu}(s_{i_k}), \overline{\nu}(s_{N_{\mathtt{f}}+1}), \dots, \overline{\nu}(s_n))$ if $n \geq N_{\mathtt{f}}$
- $\overline{\nu}(x \cdot s_1 \cdots s_n) = x \cdot \overline{\nu}(s_1) \cdots \overline{\nu}(s_n)$
- $\overline{\nu}((\lambda x.s_0) \cdot s_1 \cdots s_n) = (\lambda x.\overline{\nu}(s_0)) \cdot \overline{\nu}(s_1) \cdots \overline{\nu}(s_n)$

Find: a reduction ordering such that: $\overline{\nu}(\ell) \succ \overline{\nu}(r)$ or $\overline{\nu}(\ell) = \overline{\nu}(r)$ for all $\ell \to r \in \mathcal{P} \cup \mathcal{R}$

Argument filtering

Suppose:

- Left-hand sides of rules have no subterm $x \cdot s_1 \cdots s_n$ or $(\lambda x.s_0) \cdot s_1 \cdots s_n$ with $n > 0$
- Each occurrence of $\mathtt{f}$ in $\mathcal{R}, \mathcal{P}$ has at least $N_{\mathtt{f}}$ arguments.

Choose:

- a sequence $1 \leq i_1 < i_2 < \cdots < i_k \leq N_{\mathtt{f}}$ for each $\mathtt{f}$

Define:

- $\overline{\nu}(\mathtt{f}(s_1, \ldots, s_n)) = \mathtt{f}(\overline{\nu}(s_{i_1}), \ldots, \overline{\nu}(s_{i_k}), \overline{\nu}(s_{N_{\mathtt{f}}+1}), \ldots, \overline{\nu}(s_n))$ if $n \geq N_{\mathtt{f}}$
- $\overline{\nu}(x \cdot s_1 \cdots s_n) = x \cdot \overline{\nu}(s_1) \cdots \overline{\nu}(s_n)$
- $\overline{\nu}((\lambda x.s_0) \cdot s_1 \cdots s_n) = (\lambda x.\overline{\nu}(s_0)) \cdot \overline{\nu}(s_1) \cdots \overline{\nu}(s_n)$

Find: a reduction ordering such that: $\overline{\nu}(\ell) \succ \overline{\nu}(r)$ or $\overline{\nu}(\ell) = \overline{\nu}(r)$ for all $\ell \to r \in \mathcal{P} \cup \mathcal{R}$

Then: remove all $\ell \to r$ from $\mathcal{P}$ that were oriented with $\succ$

Exercise

Prove finiteness of the following DP problem using argument filterings and HORPO.

$$
\begin{array}{rcl}
\mathtt{minus}(x) & \to & x \\
\mathtt{minus}(\mathtt{s}(x)) & \to & \mathtt{minus}(x) \\
\mathtt{quot}(\mathtt{0}) & \to & \mathtt{0} \\
\mathtt{quot}(\mathtt{s}(x)) & \to & \mathtt{s}(\mathtt{quot}(\mathtt{minus}(x))) \\
\mathtt{sma}(b, F, \mathtt{0}) & \to & \mathtt{0} \\
\mathtt{sma}(\mathtt{true}, F, \mathtt{s}(x)) & \to & \mathtt{s}(x) \\
\mathtt{sma}(\mathtt{false}, F, \mathtt{s}(x)) & \to & \mathtt{sma}(F \cdot x, F, \mathtt{quot}(x, \mathtt{s}\ (\mathtt{s}\ \mathtt{0})))
\end{array}
$$

$$
\mathtt{sma}^{\sharp}(\mathtt{false}, F, \mathtt{s}(x)) \to \mathtt{sma}^{\sharp}(F \cdot x, F, \mathtt{quot}(x, \mathtt{s}\ (\mathtt{s}\ \mathtt{0})))
$$

Most important missing steps

Most important missing steps

- Using fully first-order techniques on first-order subsets of $(\mathcal{P}, \mathcal{R})$

Most important missing steps

- Using fully first-order techniques on first-order subsets of $(\mathcal{P}, \mathcal{R})$
- Reduction pairs in general (such as weakly monotonic algebras)

Most important missing steps

- Using fully first-order techniques on first-order subsets of $(\mathcal{P}, \mathcal{R})$
- Reduction pairs in general (such as weakly monotonic algebras)
- Usable rules (with respect to an argument filtering)

Most important missing steps

- Using fully first-order techniques on first-order subsets of $(\mathcal{P}, \mathcal{R})$
- Reduction pairs in general (such as weakly monotonic algebras)
- Usable rules (with respect to an argument filtering)
- Narrowing