Handout for part 4:

Termination:
modular termination proofs using dependency pairs

1. Modularity

2

Motivation

Goal:

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

This is useful with an eye on program analysis. Programs tend to be more than just a few lines!

Secondary goal:

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

In a programming context, it makes sense to separately analyse **modules** without knowing the full program. This allows us to create certified libraries, or to not have to completely redo an expensive analysis.

3

Running example

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

4

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 $\mathcal{R}$

However, this situation is not to be. Even in first-order rewriting this does not hold, as evidenced by the example below.

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 $\mathtt{f}(\mathtt{a}, \mathtt{b}, \pi(\mathtt{a}, \mathtt{b}))$

5

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 $\mathcal{R} \cup \mathcal{C}_\epsilon$

This one does hold in first-order rewriting! Unfortunately, in the higher-order case it still does not hold.

My counterexample:

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

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

Note that $\mathtt{comp2}(\mathtt{s}^n(\mathtt{0}), \mathtt{s}^m(\mathtt{0}))$ reduces to $\mathtt{true}$ if $n \geq 2 * m$, and to $\mathtt{false}$ otherwise. If we can only use λ-terms, constructors and $\mathtt{find}$, then we cannot construct a function F such that $F(n) \geq 2n$ for all n, so eventually any term in the signature of A terminates. But in the combined system, the term $\mathtt{find}(\mathtt{double}, \mathtt{0}, \mathtt{true})$ does not terminate.

6

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 (†)

7

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

2. First-order

8

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.

(I conveniently colour-coded them for you!)

Dependency pairs:

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

Technically, we introduce a **fresh** symbol $\mathtt{f}^\sharp$ for each defined symbol $\mathtt{f}$, with the same arity. These symbols are constructors with respect to the original set $\mathcal{R}$, but are the defined symbols of the rules $\mathtt{DP}(\mathcal{R})$ forming the *dependency pairs* of $\mathcal{R}$.

9

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 $\to_{\mathcal{R}}$

10

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 $|s| \to_{\mathcal{R}} \cdot \trianglerighteq |t|$, where $|s|$ is s with every $\mathtt{f}^\sharp$ replaced by $\mathtt{f}$, and $\trianglerighteq$ is the subterm relation. Hence, an infinite chain induces an infinite reduction $|s_1| \to_{\mathcal{R}} C_1[|t_1|] \to_{\mathcal{R}}^* C_1[|s_2|] \to_{\mathcal{R}} C_1[C_2[|t_2|]] \to_{\mathcal{R}}^* \ldots$.

$\Leftarrow$ If $\to_{\mathcal{R}}$ is non-terminating, there is a **minimal non-terminating** term s (that is, a term s that is non-terminating, but all its subterms do terminate).
Hence, there is an infinite reduction

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

where $\to_{\mathcal{R},in}$ refers to a reduction in the arguments. There must be a root step eventually, because if not, we would have an infinite reduction in an argument, contradicting their termination.

Let p be the smallest subterm of r such that $p\gamma$ is non-terminating. We know that p exists, because $r\gamma$ is non-terminating.

We easily see: $\ell^\sharp \to p^\sharp$ is a dependency pair! (Because if p is a variable, then $p\gamma$ is a subterm of some s'_i and therefore terminatinng; and if $p = \mathtt{f}(p_1, \ldots, p_n)$ with $\mathtt{f}$ a constructor, then any infinite reduction starting in $p\gamma$ induces an infinite reduction starting in some $p_i\gamma$, thus contradicting minimality of p.)

11

Proving termination using dependency pairs

Rules:

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

Dependency pairs:

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

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

12

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:

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

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)?

3. Higher-order

14

Higher-order challenges

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

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

When we consider function calls, should we include **collapsing** calls $F \cdot t$?

This is not a trivial question! There are different ways to answer this question, and they lead to different methodologies:

Two approaches:

- **dynamic dependency pairs:** include collapsing DPs like $\texttt{map}^\sharp(F, \texttt{cons}(x, l)) \to F \cdot x$
  Arguably this is the most general approach. A function is called, even if we do not know a priori which function it is. Using this approach allows us to define a **sound and complete** method for higher-order termination: if a system is terminating, this can in principle be proved using dynamic dependency pairs.
- **static dependency pairs:** only include non-collapsing DPs like $\texttt{map}^\sharp(F, \texttt{cons}(x, l)) \to \texttt{map}^\sharp(F, l)$.
  As we will see, this approach is less general: it comes with restrrictions, and we cannot always find a termination proof using static dependency pairs even for a terminating system. However, the methods from the first-order setting extend much more naturally. And most importantly, this approach seems to be much more usable for *modular* proofs.

Underlying proof idea:

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

Recall that in the soundness proof of the first-order dependency pair approach, we considered minimal non-terminating terms – and as a result, dependency pairs $\texttt{f}^\sharp(\ell_1, \ldots, \ell_k) \to \texttt{g}^\sharp(r_1, \ldots, r_n)$ were built from terms $\texttt{f}(\ell_1, \ldots, \ell_k)\gamma$, $\texttt{g}(r_1, \ldots, r_n)\gamma$ whose immediate subterms $\ell_i\gamma$ and $r_j\gamma$ were terminating. In the higher-order setting, we could use the same proof strategy – but we can either use termination directly, or use computability, which is arguably its higher-order counterpart. However, if we choose the latter, we also need to *prove* computability of all terms – which, considered per term, is a stronger property.

In this talk, we will use the **static** approach.

15

Higher-order challenges

Discussion: what should be the dependency pairs of:

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

(Rules for `double` are also assumed to be given.)

Likely answer:

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

But: we may assume x is **computable**. (That is, it will only be instantiated by computable terms.)

When we use the static DP approach, *minimal non-computability* of $\mathtt{f}(\vec{s})$ implies the existence of an infinite chain. Then, on the right-hand side of rules, we identify subterms that may lead to *non-computability* rather than merely non-termination. This includes $\lambda x.\mathtt{double}(x)$: while this term is clearly terminating (as it does not reduce), for computability we must see that $\mathtt{double}(s)$ is computable for **all** computable s. This is why we include a dependency pair $\mathtt{up}^\sharp(l) \to \mathtt{double}^\sharp(x)$ where x may be instantiated by any computable term.

16

Higher-order challenges

Discussion: what should be the dependency pairs of:

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

Likely answer:

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

Again: we may assume that x is **computable**.

This holds by the same reasoning as before: for $\mathtt{double}$ to be computable, its application to an arbitrary computable term must be computable.

17

Higher-order challenges

Discussion: what should be the dependency pairs of:

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

Likely answer:

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

Since this rule is basically a shorter version of $\mathtt{up}(l) \to \mathtt{map}(\mathtt{double}, l)$, it seems wise to treat them the same. Essentially, we can expand a rule into a maximally applied one before computing dependency pairs, to avoid needlessly introducing fresh variables.

18

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:

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

Likely answer: This should not be allowed!

In this case, computability of $\mathtt{lam}(F)$ does **not** imply computability of F. So we have two choices: limit the static DP framework to systems where this does not occur, or include some collapsing DPs $\mathtt{app}^\sharp(\mathtt{lam}(F), x) \to F \cdot x$ after all. We choose the former, as it would be much harder to define what exactly is a chain in the latter case.

19

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

20

Plain function passing

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

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

✓

21

Plain function passing

$$\begin{array}{rcl} \mathtt{app} & :: & \mathsf{term} \Rightarrow \mathsf{term} \Rightarrow \mathsf{term} \\ \mathtt{lam} & :: & (\mathsf{term} \Rightarrow \mathsf{term}) \Rightarrow \mathsf{term} \end{array}$$

$$\mathtt{app}(\mathtt{lam}(F)) \to F$$

✗

22

Plain function passing

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

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

✗

So this definition does **not** allow us to handle lists of functions. However, that is only because I did not want to make things *too* complex for this presentation; there are extensions of the method (using a different definition of computability) where systems like this are admitted.

23

Plain function passing

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

✓

24

Definition

Definition

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

$$\begin{array}{rcll}
\mathrm{Cand}(\mathtt{f}(s_1,\ldots,s_n)) & = & \{\mathtt{f}(s_1,\ldots,s_n)\} \cup \bigcup_{i=1}^{n} \mathrm{Cand}(s_i) & \text{(if } \mathtt{f} \text{ is a defined symbol)} \\
\mathrm{Cand}(\mathtt{c}(s_1,\ldots,s_n)) & = & \bigcup_{i=1}^{n} \mathrm{Cand}(s_i) & \text{(if } \mathtt{c} \text{ is a constructor symbol)} \\
\mathrm{Cand}(x \cdot s_1 \cdots s_n) & = & \bigcup_{i=1}^{n} \mathrm{Cand}(s_i) & \\
\mathrm{Cand}(\lambda x.s) & = & \mathrm{Cand}(s[x := y]) & \text{(some fresh } y\text{)} \\
\mathrm{Cand}((\lambda x.t) \cdot s_0 \cdots s_n & = & \mathrm{Cand}(t[x := s_1] \cdot s_1 \cdots s_n) \cup \mathrm{Cand}(s_1) &
\end{array}$$

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

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

25

Exercise

Compute the dependency pairs of:

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

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

26

Higher-order-order dependency pair chain

(Changes compared to the first-order definition are highlighted in red.)

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

(Here, $FV(s)$ denotes the free variables of s.)

27

Dependency chain claim: proof sketch

Claim:

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

Proof sketch:

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

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

- 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.

28

Discussion:

Plain-function passingness:

- admits most (terminating) common examples

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

Polymorphism overall:

- forcing rules into base-type before computing dependency pairs is not so practical when we consider polymorphic systems, where an output type α may be instantiated both by a base type and an arrow type...
- can be done with slightly different definitions (but I'll leave this out of the course)

Dependency Pair Processors

So now, our goal is to prove that no infinite computable $(\mathtt{DP}(\mathcal{R}), \mathcal{R})$-chain exists. We will consider a few methods to split up DP problems $(\mathcal{P}, \mathcal{R})$, and perhaps prove their harmlessness altogether.

4. Graph

30

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!

31

Splitting call groups method

Idea:

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

32

Running example

The following are the dependency pairs of the 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}
$$

This HTRS was already written in such a way that all rules only called function symbols that were defined above it, or that were mutually recursive with the same symbol. Hence, we can very naturally divide it as follows. Note that the split below was made only by looking at head symbols, and in the case of A_5, by the observation that $\mathtt{s}(\mathtt{0})$ is a ground constructor term that does not reduce to $\mathtt{0}$.

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

Now, as there are only finitely many groups, any infinite chain must have a tail that only uses DPs in one of the groups – which, moreover, must be a blue group. Hence, the DPs in the red groups (which

can be used at most once in any infinite chain) may be removed altogether, while we can split up the DP problem into the subproblems defined by the blue groups. In this case, our large initial problem is split up into 8 subproblems defined by much smaller sets of dependency pairs. This is quite typical when using the DP framework, and is of course great for our goal of *modularity*.

33

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** since there are some very efficient graph algorithms for finding SCCs.
- The groups approach is natural for **certification** since the person who wants to certify a proof can just specify the groups, and a tool like Isabelle or Coq does not have to consider how the groups were found; only that they satisfy the requirement on subsequent DPs.

34

Example

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

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.)

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

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

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

5. Subterms

36

The subterm criterion: intuition

Recall: one of our tasks is to prove that there is no infinite computable chain over $(A_8, \mathcal{R})$ where A_8 is given by:

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

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 `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!

37

The subterm criterion: definition

Given: $(\mathcal{P}, \mathcal{R})$ with marked symbols $\mathtt{f}^\sharp_1, \dots, \mathtt{f}^\sharp_n$

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

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

- either $\ell_{\nu(\mathtt{f}^\sharp_i)} \rhd r_{\nu(\mathtt{f}^\sharp_j)}$
- or $\ell_{\nu(\mathtt{f}^\sharp_i)} = r_{\nu(\mathtt{f}^\sharp_j)}$

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 (since $\to_{\mathcal{R}} \cup \rhd$ is wellfounded on computable terms). Hence, any such chain must have an infinite tail without $\rhd$ steps.

38

Examples

Let's consider all of our remaining DP problems!

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	$\mathtt{ack}^\sharp(\mathtt{s}(x),0)$	$\to$	$\mathtt{ack}^\sharp(x,\mathtt{s}(0))$
	$\mathtt{ack}^\sharp(\mathtt{s}(x),\mathtt{s}(y))$	$\to$	$\mathtt{ack}^\sharp(\mathtt{s}(x),y)$
	$\mathtt{ack}^\sharp(\mathtt{s}(x),\mathtt{s}(y))$	$\to$	$\mathtt{ack}^\sharp(x,\mathtt{ack}(\mathtt{s}(x),y))$
A_6	$\mathtt{fexp}^\sharp(\mathtt{s}(x),y)$	$\to$	$\mathtt{double}^\sharp(x,y,0)$
	$\mathtt{double}^\sharp(x,0,z)$	$\to$	$\mathtt{fexp}^\sharp(x,z)$
	$\mathtt{double}^\sharp(x,\mathtt{s}(y),z)$	$\to$	$\mathtt{double}^\sharp(x,y,\mathtt{s}(\mathtt{s}(z)))$
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}	$\mathtt{sma}^\sharp(\mathtt{false},F,\mathtt{s}(x))$	$\to$	$\mathtt{sma}^\sharp(F\cdot x,F,\mathtt{quot}(x,\mathtt{s}\ (\mathtt{s}\ 0)))$
A_{13}	$\mathtt{H}^\sharp(\mathtt{s}(x))$	$\to$	$\mathtt{H}^\sharp(\mathtt{twice}(\mathtt{I},x))$

39

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

This allows us to remove the only DP, so the problem $(\mathcal{P},\mathcal{R})$ is clearly finite.

40

Examples: A_3

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

Argument position: method does not apply (we could choose position 2, but it does not allow us to remove anything)

41

Examples: A_4

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

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

This allows us to remove the first and third dependency pairs, leaving us with:

Remaining:

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

Having fewer dependency pairs, we can apply the method again, now on a different argument!

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

This removes the only remaining dependency pair in this set. Hence, also $(A_4, \mathcal{R})$ is finite.

42

Examples: A_6

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

Argument positions:

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

(There is nothing that compels us to select the same argument index in all marked symbols; however, that is just how this example ends up.)

This choice allows us to remove the first dependency pair, leaving us with:

Remaining:

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

The subterm criterion doesn't apply again. But we *can* use the group splitting (graph) processor: the first dependency pair cannot be followed by either the first or the second, so can be removed, since it cannot occur in an infinite chain. This leaves us with just:

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

Now we can apply the subterm criterion again, this time with $\nu(\mathtt{double}^\sharp) = 2$. This allows us to remove the last remaining dependency pair in A_6.

43

Running example

We have already seen A_8 and A_7 and A_9 are easily handled, while A_{11} and A_{12} cannot be handled using the subterm criterion. Hence, of our running example we only have three sets $\mathcal{P}$ left where we have to prove the absence of an infinite computable chain:

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

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

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

6. argument filters

44

First-order example

Consider:

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

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

This gives:

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

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

45

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 $1 \le i_1, i_2, \ldots, i_k \le N_\mathtt{f}$ 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 $n \ge N_\mathtt{f}$ and $\overline{\nu}(\mathtt{f}(s_1,\ldots,s_n)) = \lambda x_{n+1}\ldots x_{N_\mathtt{f}}.\mathtt{f}'(\overline{\nu}(s_{i_1}),\ldots,\overline{\nu}(s_{i_k}))$ otherwise, where we let $s_j := x_j$ for $n < j \le N_\mathtt{f}$ and $\mathtt{f}'$ be a fresh symbol of appropriate type
- $\overline{\nu}(x \cdot s_1 \cdots s_n) = x \cdot \overline{\nu}(s_1) \cdots \overline{\nu}(s_n)$
- $\overline{\nu}((\lambda x.s_0) \cdot s_1 \cdots s_n) = (\lambda x.\overline{\nu}(s_0)) \cdot \overline{\nu}(s_1) \cdots \overline{\nu}(s_n)$

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 $\succ$

Proof idea: We can do this because it is not so hard to prove that:

- If all occurrences of each $\mathtt{f}$ in s have at least $N_{\mathtt{f}}$ arguments, then $\overline{\nu}(s)\gamma^{\overline{\nu}} \to_\beta^* \overline{\nu}(s\gamma)$, where $\gamma^{\overline{\nu}}$ maps each x to $\overline{\nu}(\gamma(x))$
- If all occurrences of each $\mathtt{f}$ in s have at least $N_{\mathtt{f}}$ arguments, and s has no subterm $t_0 \cdot t_1 \cdots t_n$ with $n > 0$ and t_0 a variable or abstraction, then $\overline{\nu}(s)\gamma^{\overline{\nu}} = \overline{\nu}(s\gamma)$.
- Hence, since a reduction ordering includes $\to_\beta$ by definition and is stable, if $s = \ell\gamma$ and $t = r\gamma$ for $\ell \to r \in \mathcal{P} \cup \mathcal{R}$, then $\overline{\nu}(s) \succeq \overline{\nu}(t)$ (and even $\overline{\nu}(s) \succ \overline{\nu}(t)$ if $\ell \to r \in \mathcal{P}$ was oriented with $\succ$).
- By monotonicity of $\succ$ we see: if $s \to_{\mathcal{R}} t$ then $\overline{\nu}(s) = \overline{\nu}(t)$ or $\overline{\nu}(s) \succ \overline{\nu}(t)$.
- Hence, an infinite $\to_{\mathcal{P}} \cdot \to_{\mathcal{R}}^*$ chain with the $\to_{\mathcal{P}}$ step at the root, yields an infinite number of $\succeq$ or $\succ$ steps. Due to wellfoundedness of $\succ$, at most finitely many steps can use a DP that was oriented with $\succ$, so an infinite $(\mathcal{P}, \mathcal{R})$-chain has an infinite tail that uses only dependency pairs where $\overline{\nu}(\ell) = \overline{\nu}(r)$.

46

Exercise

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

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

47

Most important missing steps

Because this course is already getting quite long, I did not get around to several parts of the DP framework that are often quite useful important in proofs – including some that are needed to finish the running example. These are:

- 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