Handout for lecture 10:

Termination

1. Overview

2

Recall: Term Rewriting Systems

In the previous lecture, we introduced term rewriting systems. These are, essentially, sets of oriented equations, which can be used to model equalities in some theories (and thus for reasoning in equational logic, as we will see in two weeks). For example, for the integer theory we may define rules like the following:

$$\begin{array}{rcl} \mathtt{add}(x,\mathtt{s}(y)) & \Rightarrow & \mathtt{s}(\mathtt{add}(x,y)) \\ \mathtt{add}(x,\mathtt{p}(y)) & \Rightarrow & \mathtt{p}(\mathtt{add}(x,y)) \\ \mathtt{add}(x,\mathtt{0}) & \Rightarrow & x \\ \mathtt{s}(\mathtt{p}(x)) & \Rightarrow & x \\ \mathtt{p}(\mathtt{s}(x)) & \Rightarrow & x \end{array}$$

Term rewriting systems are also a natural model for functional programming language. For example, list concatenation and reversal can be represented through the following TRS:

$$\begin{array}{rcl} \mathtt{rev}(\mathtt{nil}) & \Rightarrow & \mathtt{nil} \\ \mathtt{rev}(a \mathbin{\mathtt{:}} x) & \Rightarrow & \mathtt{conc}(\mathtt{rev}(x), a \mathbin{\mathtt{:}} \mathtt{nil}) \\ \mathtt{conc}(\mathtt{nil}, x) & \Rightarrow & x \\ \mathtt{conc}(a \mathbin{\mathtt{:}} x, y) & \Rightarrow & a \mathbin{\mathtt{:}} \mathtt{conc}(x,y) \end{array}$$

3

Lecture plan

Last week, we discussed one method to prove termination.

- LPO is both simple and powerful.
- LPO satisfies certain properties that make it very convenient to use in both completion and superposition (more about this in two weeks).
- But: it is not the most intuitive, or even the most commonly used method for proving termination!

We will discuss two others: **monotonic algebras** and **dependency pairs**.

2. Monotonic algebras

2.1 Intuition

4

Basic intuition of monotonic algebras

> Find a weight function W from terms to natural numbers in such a way that $W(u) > W(v)$ for all terms u, v satisfying $u \Rightarrow_{\mathcal{R}} v$.

If such a function W exists then $\mathcal{R}$ is terminating since an infinite rewriting sequence would give rise to an infinite decreasing sequence of natural numbers which does not exist.

5

Example

For the following rules:

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

We find such a weight function W for instance by defining inductively:

- $W(\mathtt{0}) = 1$
- $W(\mathtt{s}(t)) = W(t) + 1$
- $W(\mathtt{add}(t, u)) = 2W(t) + W(u)$

Then for instance:

$$\begin{array}{rcl} W(\mathtt{add}(\mathtt{s}(\mathtt{0}), \mathtt{0})) & = & 2 * W(\mathtt{s}(\mathtt{0})) + W(\mathtt{0}) = 2 * 2 + 1 = 5 \\ W(\mathtt{s}(\mathtt{add}(\mathtt{0}, \mathtt{0}))) & = & W(\mathtt{add}(\mathtt{0}, \mathtt{0})) + 1 = (2 * 1 + 1) + 1 = 4 \\ W(\mathtt{s}(\mathtt{0})) & = & W(\mathtt{0}) + 1 = 1 + 1 = 2 \end{array}$$

6

Better idea: reduction ordering

The basic idea of weight functions is too general:

It allows arbitrary definitions of weight functions, and we have to prove that $W(s) > W(t)$ for **all** rewrite steps $s \Rightarrow t$, while typically there are infinitely many of them.

We already saw how to handle this last week: use a **reduction ordering** $\succ$:

- $\succ$ is **well-founded**

- $\succ$ is **stable** (so preserved under substitution)
- $\succ$ is **monotonic** (so preserved under contexts)

Then it suffices to prove $\ell \succ r$ for all the rules.

2.2 Monotonic algebras

7

Monotonic algebras

Hence, **monotonic algebras** are a special case of this idea of weight functions.

They allow us to generate a reduction ordering in a way where we only have to:

- choose interpretations for the (finitely many) function symbols rather than for all terms, and
- check $W(\ell) \succ W(r)$ for the (finitely many) rules $\ell \Rightarrow r$ rather than all rewrite steps.

They also allow us to map to other sets than $\mathbb{N}$.

8

Definition

Let $\mathcal{A}$ be a set and $>$ a well-founded relation on $\mathcal{A}$ (we typically take $\mathcal{A} := \mathbb{N}$).

For every function symbol $\mathtt{f}$ of arity n, we choose a **monotonic** function $[\mathtt{f}] : \mathcal{A}^n \to \mathcal{A}$.

Here **monotonic** means:

> if for all $a_i, b_i \in \mathcal{A}$ for $i = 1, \dots, n$ with $a_i > b_i$ for some i and $a_j \geq b_j$ for all $j \neq i$ then
> $$[\mathtt{f}](a_1, \dots, a_n) \; > \; [\mathtt{f}](b_1, \dots, b_n)$$

Examples:

monotonic	*not monotonic*
$\lambda x.\ x$	$\lambda x.\ 2$
$\lambda x.\ x+1$	$\lambda x, y.\ x+1$
$\lambda x.\ 2*x$	$\lambda x, y.\ x*y$
$\lambda x, y.\ x+y$	$\lambda x, y.\ \max(x, y)$
$\lambda x, y.\ 2*x+y+1$	

9

Definition (continued)

Define:

- $W(x) = x$ for a variable

- $W(\mathtt{f}(s_1, \ldots, s_n)) = [\mathtt{f}](W(s_1), \ldots, W(s_n))$

Theorem

The relation $\succ$ defined by:

$$s \succ t \text{ if and only if } \forall \vec{x}[W(s) \succ W(t)],$$
$$\text{where } \{\vec{x}\} \text{ is the set of variables occurring in } s, t$$

is a reduction ordering.

Proof idea. Stability follows from the $\forall \vec{x}$, monotonicity from monotonicity of all $[\mathtt{f}]$ interpretation functions, and well-foundedness from well-foundedness of $>$ in $\mathcal{A}$. □

Hence, if we can prove $\ell \succ r$ for all rules with such an interpretation, then $\mathcal{R}$ is terminating.

10

Example

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

We let:

- $[\mathtt{0}] = 1$
- $[\mathtt{s}] = \lambda x.x + 1$
- $[\mathtt{add}] = \lambda x, y.2 * x + y$

Now indeed for all x, y we have:

$$W(\mathtt{add}(\mathtt{0}, y)) = 2 * 1 + y = 2 + y > y = W(y)$$

and:

$$\begin{array}{rcl} W(\mathtt{add}(\mathtt{s}(x), y) & = & 2 * (x + 1) + y = 2 * x + y + 2 \\ & > & 2 * x + y + 1 = W(\mathtt{s}(\mathtt{add}(x, y))) \end{array}$$

Hence, we have proved termination of this TRS.

11

Another example

For the TRS $\mathcal{R}$ consisting of the single rule

$$\mathtt{f}(\mathtt{g}(x)) \to \mathtt{g}(\mathtt{g}(\mathtt{f}(x)))$$

we choose monotonic functions

$$[\mathtt{f}](x) = 3x$$
$$[\mathtt{g}](x) = x + 1$$

Now indeed for all $x \in \mathbb{N}$ we have:

$$W(\mathtt{f}(\mathtt{g}(x))) = 3(x+1) >$$
$$3x + 1 + 1 = W(\mathtt{g}(\mathtt{g}(\mathtt{f}(x))))$$

Hence proving termination of $\mathcal{R}$.

12

Yet another example?

For the TRS consisting of the single rule

$$\mathtt{f}(x) \to \mathtt{g}(\mathtt{f}(x))$$

we choose the functions

$$[\mathtt{f}](x) = x + 1$$
$$[\mathtt{g}](x) = 0$$

Then we indeed have:

$$W(\mathtt{f}(x)) = x + 1 > 0 = W(\mathtt{g}(\mathtt{f}(x))$$

Yet, this system is non-terminating! Where is the error?

Answer: $[\mathtt{g}]$ is not monotonic.

So monotonicity really is essential.

2.3 Implementation

13

Automating monotonic algebras

Modern termination provers implement a variety of techniques, including LPO and interpretations to $\mathbb{N}$.

Modern termination provers rely heavily on SAT and SMT solvers to handle underlying constraints. We saw this last week for LPO.

Also interpretations can be handled in this way – to some extent.

14

Parametric interpretations

Idea: assign to function symbol f of arity n a *parametric interpretation function* of a specific shape; for instance

$$\begin{array}{rcl} [0] & = & \underline{n} \\ [s] & = & \lambda x.\underline{s_0} + \underline{s_1} * x \\ [\mathtt{add}] & = & \lambda x, y.\underline{a_0} + \underline{a_1} * x + \underline{a_2} * y \end{array}$$

Use these parametric functions to obtain parametric interpretations of both sides of each rule; for instance:

$$\begin{array}{rcl} W(\mathtt{add}(0, y)) & = & \underline{a_0} + \underline{a_1} * \underline{n} + \underline{a_2} * y \\ W(y) & = & y \\ W(\mathtt{add}(\mathtt{s}(x), y)) & = & \underline{a_0} + \underline{a_1} * (\underline{s_0} + \underline{s_1} * x) + \underline{a_2} * y \\ & = & \underline{a_0} + \underline{a_1} * \underline{s_0} + \underline{a_1} * \underline{s_1} * x + \underline{a_2} * y \\ W(\mathtt{s}(\mathtt{add}(x, y))) & = & \underline{s_0} + \underline{s_1} * (\underline{a_0} + \underline{a_1} * x + \underline{a_2} * y) \\ & = & \underline{s_0} + \underline{s_1} * \underline{a_0} + \underline{s_1} * \underline{a_1} * x + \underline{s_1} * \underline{a_2} * y \end{array}$$

15

Inequalities with (universally quantified) variables

We now have to solve a problem of the shape:

find parameters such that:

- *all* $[f]$ *are monotonic functions, and*
- *for all rules* $\ell \to r$*, all* $\vec{x}$*:* $W(\ell) \succ W(r)$.

Requiring monotonicity in our example is not hard:

$$\begin{array}{rcl} [0] & = & \underline{n} \\ [s] & = & \lambda x.\underline{s_0} + \underline{s_1} * x \\ [\mathtt{add}] & = & \lambda x, y.\underline{a_0} + \underline{a_1} * x + \underline{a_2} * y \end{array}$$

We require that: $\underline{s_1} \geq 1$, $\underline{a_1} \geq 1$, $\underline{a_2} \geq 1$.

For the second requirement, we use **absolute positiveness**.

16

Absolute positiveness

Absolute positiveness is a sufficient condition for checking positiveness of a polynomial – or, equivalently, comparing two polynomials.

$$a_0 + a_1 * x_1 + \cdots + a_m * x_m > b_0 + b_1 * x_1 + \cdots + b_m * x_m$$

certainly holds if:

- $a_0 > b_0$
- each $a_i \geq b_i$

Example (continued)

We must show that:

$$\begin{array}{rcl} W(\mathtt{add}(\mathtt{0},y)) & = & \underline{a_0}+\underline{a_1}*\underline{n}+\underline{a_2}*y \\ & > & y \\ & = & W(y) \\ W(\mathtt{add}(\mathtt{s}(x),y)) & = & \underline{a_0}+\underline{a_1}*\underline{s_0}+\underline{a_1}*\underline{s_1}*x+\underline{a_2}*y \\ & > & \underline{s_0}+\underline{s_1}*\underline{a_0}+\underline{s_1}*\underline{a_1}*x+\underline{s_1}*\underline{a_2}*y \\ & = & (\mathtt{s}(\mathtt{add}(x,y))) \end{array}$$

That is:

$$(\underline{a_0}+\underline{a_1}*\underline{n})+\underline{a_2}*y>y$$

and

$$(\underline{a_0}+\underline{a_1}*\underline{s_0})+(\underline{a_1}*\underline{s_1})*x+\underline{a_2}*y>(\underline{s_0}+\underline{s_1}*\underline{a_0})+(\underline{s_1}*\underline{a_1})*x+(\underline{s_1}*\underline{a_2})*y$$

Using absolute positiveness, it suffices if:

$$\begin{array}{rclcrcl} \underline{a_0}+\underline{a_1}*\underline{n} & > & 0 & \quad & \underline{a_0}+\underline{a_1}*\underline{s_0} & > & \underline{s_0}+\underline{s_1}*\underline{a_0} \\ & & & & \underline{a_1}*\underline{s_1} & \geq & \underline{s_1}*\underline{a_1} \\ \underline{a_2} & \geq & 1 & & \underline{a_2} & \geq & \underline{s_1}*\underline{a_2} \end{array}$$

Completing the example

Hence, the problem of finding a monotonic interpretation is reduced to the following SMT problem:

$$\begin{array}{rclcrclcrcl} \underline{a_0}+\underline{a_1}*\underline{n} & > & 0 & \quad & \underline{a_0}+\underline{a_1}*\underline{s_0} & > & \underline{s_0}+\underline{s_1}*\underline{a_0} & \quad & \underline{s_1} & \geq & 1 \\ & & & & \underline{a_1}*\underline{s_1} & \geq & \underline{s_1}*\underline{a_1} & & \underline{a_1} & \geq & 1 \\ \underline{a_2} & \geq & 1 & & \underline{a_2} & \geq & \underline{s_1}*\underline{a_2} & & \underline{a_2} & \geq & 1 \end{array}$$

An SMT solver will for instance yield

$$\underline{n}=1,\ \underline{s_0}=1,\ \underline{s_1}=1,\ \underline{a_0}=0,\ \underline{a_1}=2,\ \underline{a_2}=1$$

giving the same interpretations we had before:

$$\begin{array}{rcl} [\mathtt{0}] & = & \underline{n} \\ [\mathtt{s}] & = & \lambda x.\underline{s_0}+\underline{s_1}*x \\ [\mathtt{add}] & = & \lambda x,y.\underline{a_0}+\underline{a_1}*x+\underline{a_2}*y \end{array} \quad \Longrightarrow \quad \begin{array}{rcl} [\mathtt{0}] & = & 1 \\ [\mathtt{s}] & = & \lambda x.1+x \\ [\mathtt{add}] & = & \lambda x,y.2*x+y \end{array}$$

Many other interpretations are also possible.

In practice, if usually suffices to limit the search space for parameters to $\{0,1,2,3\}$.

19

Limitations

Absolute positiveness also works with combinations of variables, e.g.,

$$\underline{a} + \underline{b}y + \underline{c}xy + \underline{d}x^2y > \underline{e}x + \underline{d}y + \underline{f}xy$$

if $\underline{a} > 0 \wedge 0 \geq \underline{e} \wedge \underline{b} \geq \underline{d} \wedge \underline{c} \geq \underline{f} \wedge \underline{d} \geq 0$.

Warning: this is not a complete method!
We may not find an interpretation this way, even if there is one:

- absolute positiveness does not capture all inequalities; e.g., we do not derive $x^2 \geq x$
- we might not guess the right interpretation shape

20

Limitations example

$$\mathtt{mul}(\mathtt{s}(x), y) \to \mathtt{add}(y, \mathtt{mul}(x, y))$$

We will not find a suitable interpretation with the shapes

$$[\mathtt{f}] = \lambda x_1, \dots, x_n.\underline{a_0} + \Sigma_{i=1}^n \underline{a_i} * x_i$$

that we tried before.

To interpret this we will need at least a **quadratic** interpretation function, e.g., a shape

$$[\mathtt{f}] = \lambda x_1, \dots, x_n.\underline{a} + \Sigma_{i=1}^n \underline{b_i} * x_i + \Sigma_{i=1}^n \Sigma_{j=i}^n \underline{c_{ij}} * x_i * x_j$$

Trying more sophisticated interpretation shapes results in an often much more complex SMT problem.

2.4 Other interpretation domains

21

Mapping to pairs of numbers

Note: $\mathcal{A}$ is not **required** to be $\mathbb{N}$. Any well-founded set will do.

Example: $\mathcal{A} = \mathbb{N}^2$: vectors of size 2, and $(x_1, y_1) > (x_2, y_2)$ if $x_1 > x_2$ and $y_1 \geq y_2$.

Typical uses:

- **matrix interpretations**: interpretation shapes of the form $[\mathtt{f}] = A_0 + A_1x_1 + \cdots + A_nx_n$, where all A_i are 2 x 2 **matrixes**.

- **interpretations using max**: in the *second* component, we can use this without losing monotonicity.

Example:

$$\mathtt{f}(\mathtt{s}(x)) \to \mathtt{f}(\mathtt{p}(\mathtt{s}(x))) \qquad \mathtt{p}(\mathtt{0}) \to \mathtt{0} \qquad \mathtt{p}(\mathtt{s}(x)) \to x$$

$$\begin{array}{rcl} [\mathtt{0}] & = & \langle 0, 0 \rangle \\ [\mathtt{s}] & = & \lambda \langle x_1, x_2 \rangle . \langle x_1, x_2 + 1 \rangle \\ [\mathtt{p}] & = & \lambda \langle x_1, x_2 \rangle . \langle x_1, \max(x_2 - 1, 0) \rangle \\ [\mathtt{f}] & = & \lambda \langle x_1, x_2 \rangle . \langle x_1 + x_2, 0 \rangle \end{array}$$

22

Cost-size interpretations

One way to think of such interpretations to $\mathbb{N}^2$ is as a **combination** of two measures.

The first component bounds the **number of steps** that can be done starting in a given term (as it must decrease in each step).

The second component may give some kind of **size measure**, which is allowed to stay the same under reduction.

For example, in

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

we may choose:

$$\begin{array}{rcl} [\mathtt{0}] & = & \langle 0, 0 \rangle \\ [\mathtt{s}] & = & \lambda \langle x_1, x_2 \rangle . \langle x_1, x_2 + 1 \rangle \\ [\mathtt{add}] & = & \lambda \langle x_1, x_2 \rangle, \langle y_1, y_2 \rangle . \langle x_1 + x_2 + y_1, x_2 + y_2 \rangle \end{array}$$

This expresses that to reduce a term $\mathtt{add}(s, t)$ to normal form, we need at most the number of steps to reduce s to normal form, plus the number of steps to reduce t to normal form, plus the size of s, and the size of the result is at most the size of s plus the size of t.

This is also valuable in **complexity analysis** of programs. (A form of resource analysis.)

23

Alternative interpretation domains

Other domains used for monotonic algebras include (but are not limited to):

- $\mathbb{N}^k$ for any number k
- rational or real numbers (where "$a > b$" if $a \geq b + \epsilon$ for some fixed number ϵ)
- integer numbers above some bound, e.g., $\{k, k + 1, \dots\}$, where k may be positive or negative
- sets of terms terminating under some different well-founded ordering $\succ$

3. Dependency pairs

3.1 Motivation

24

Subterm property

Definition
A reduction ordering $\succ$ has the subterm property if $\mathtt{f}(\dots, s, \dots) \succeq s$ for all f.

That is, a term is always greater than or equal to its subterms in the ordering.—

All recursive path orderings have the subterm property, as do monotonic algebras to $\mathbb{N}$.

The subterm property is useful in some cases; for example, it is essential in the proof of superposition.

However, having the subterm property is not always a good thing, since it presents a restriction on which TRSs can be handled with such reduction pairs.

25

Motivating example

Consider the following system:

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

If $\mathtt{minus}(x, y) \succ y$ and $\mathtt{s}(a) \succ a$, then:

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

Hence, if the last rule is oriented with $\succ$, then by stability:

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

Hence, this system **cannot** be ordered using any recursive path ordering, or an interpretation to $\mathbb{N}$. (Although it **can** be ordered by an interpretation to $\mathbb{N}^2$.)

Wish: it would be nice to have a general approach that can handle all TRSs, including this one.

26

Secondary motivation

Program analysis: often hundreds or thousands of rules.

Problem: finding an ordering for all at once is computationally difficult. While SMT solvers can do a lot, making the problem larger still typically increases the time for solving exponentially.

Wish: it would be nice if we could split a termination problem into multiple smaller problems.

3.2 Dependency pairs and chains: definition

27

The dependency pair framework

One of the most popular methods to prove termination is the **dependency pair framework**.

Idea: the dependency pair framework is a **general** framework for termination analysis:

- The DP framework does not suffer from the "subterm property" problem, and can (in principle) handle any TRS.
  (Although there is no algorithm that will definitively **find** a proof using dependency pairs.)

- Reduction orderings are building blocks that can be used inside the DP framework, as well as many other techniques.

- The DP framework can both be used to prove termination and non-termination, although in this course we will only consider its power for termination.

The core idea of dependency pairs is to look at **function calls**.

To start, we split the set of function symbols into `constructors` and `defined symbols`.

Here, for a TRS R, a symbol `f` is called a **defined symbol** if `f` is the root symbol of a left-hand side of a rule in R. All other symbols are **constructors**.

28

Identifying function calls

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

We isolate the calls from one defined symbol to another:

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

Here, we mark the **outer symbols** with a $\sharp$, since they have a different status than the inner symbols.

These pairs $\mathtt{f}^\sharp(\ell_1,\dots,\ell_k) \Rightarrow \mathtt{g}^\sharp(r_1,\dots,r_n)$ are called **dependency pairs**.

29

Dependency pair chains

We can prove the following result:

Theorem

For a given set of rules R, let $\mathtt{DP}$ be the corresponding set of dependency pairs.

$\Rightarrow_\mathcal{R}$ is terminating if and only if there is no infinite $(\mathtt{DP}, R)$**-chain**: a reduction $s_1 \Rightarrow_{\mathtt{DP}}\Rightarrow_\mathcal{R}^* s_2 \Rightarrow_{\mathtt{DP}}\Rightarrow_\mathcal{R}^* s_3 \dots$

Since the root symbols are all marked with a $\sharp$, the steps using $\Rightarrow_\mathcal{R}$ can only occur in **strict subterms**, and the steps with $\Rightarrow$ can only occur at the root.

Hence, this theorem says that $\Rightarrow_\mathcal{R}$ is terminating if and only if there is no infinite sequence of $\Rightarrow_{\mathtt{DP}} \cup \Rightarrow_\mathcal{R}$ where:

- the steps using $\Rightarrow_{\mathtt{DP}}$ occur at the root of the term;
- the steps using $\Rightarrow_\mathcal{R}$ do not occur at the root of the term;
- there are infinitely many steps using $\Rightarrow_{\mathtt{DP}}$.

30

Example: an infinite DP chain

$$R = \left\{ \begin{array}{rcl} \mathtt{f}(\mathtt{0},x) & \to & \mathtt{g}(\mathtt{f}(\mathtt{g}(x),x)) \\ \mathtt{g}(x) & \to & x \end{array} \right\}$$

$$\mathtt{DP} = \left\{ \begin{array}{rcl} \mathtt{f}^\sharp(\mathtt{0},x) & \Rightarrow & \mathtt{g}^\sharp(\mathtt{f}(\mathtt{g}(x),x)) \\ \mathtt{f}^\sharp(\mathtt{0},x) & \Rightarrow & \mathtt{f}^\sharp(\mathtt{g}(x),x) \\ \mathtt{f}^\sharp(\mathtt{0},x) & \Rightarrow & \mathtt{g}^\sharp(x) \end{array} \right\}$$

Infinite chain: $\underline{\mathtt{f}^\sharp(\mathtt{0},\mathtt{0})} \Rightarrow_{\mathtt{DP}} \mathtt{f}^\sharp(\underline{\mathtt{g}(\mathtt{0})},\mathtt{0}) \Rightarrow_\mathcal{R} \underline{\mathtt{f}^\sharp(\mathtt{0},\mathtt{0})} \Rightarrow_{\mathtt{DP}} \mathtt{f}^\sharp(\underline{\mathtt{g}(\mathtt{0})},\mathtt{0}) \Rightarrow_\mathcal{R} \dots$

3.3 Using a reduction pair

31

Building block: reduction pairs

How do you prove that there is no infinite chain?

One possibility: using a variant of a reduction ordering: a **reduction pair**.

Theorem

If:

- $\ell \succ r$ for all dependency pairs $\ell \Rightarrow r \in \mathtt{DP}$,
- $\ell \succeq r$ for all rules $\ell \rightarrow r \in R$,
- $\succ$ is **well-founded** and **stable** (but not necessarily monotonic),
- $\succeq$ is **stable** and **monotonic** (but not necessarily well-founded),
- and $s \succ t \succeq u$ implies $s \succ u$,

then there is no infinite $(\mathtt{DP}, R)$-chain!

Proof idea. By these requirements:

- If $s \Rightarrow_{\mathtt{DP}} t$ (by a step at the root), then $s \succ t$.
- If $s \Rightarrow_{\mathcal{R}} t$ then $s \succeq t$.
- Hence, if $s \Rightarrow_{\mathtt{DP}} \cdot \Rightarrow_{\mathcal{R}}^* t$, then $s \succ \cdot \succeq^* t$, and therefore $s \succ t$.

Hence, an infinite $(\mathtt{DP}, R)$-chain yields an infinite decreasing $\succ$-sequence. $\square$

32

Quot/minus: ordering requirements

Our quot/minus example from before gives the following ordering requirements:

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

33

Weakly monotonic algebras

One way to generate a reduction pair is through **weakly monotonic** algebras.

This is just like the monotonic algebras we saw before, but $[\mathtt{f}]$ is only required to be **weakly monotonic**:

$$\text{if } s \geq t, \text{ then } [\mathtt{f}](\dots, s, \dots) \geq [\mathtt{f}](\dots, t, \dots).$$

Many functions that are not monotonic, are still weakly monotonic; for example:

- functions that ignore arguments: $\lambda x, y.\ x$
- min / max functions: $\lambda x, y.\ \min(x, y)$ or $\lambda x.\ \max(x - 1, 0)$

Completing quot/minus

Back to our example!

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

This is satisfied by choosing:

$$\begin{array}{rclcrclcrcl}
[\mathtt{0}] & = & 0 & & [\mathtt{minus}] & = & \lambda x, y.\ \ x & & [\mathtt{minus}^\sharp] & = & \lambda x, y.\ \ x \\
[\mathtt{s}] & = & \lambda x.\ \ x + 1 & & [\mathtt{quot}] & = & \lambda x, y.\ \ x & & [\mathtt{quot}^\sharp] & = & \lambda x, y.\ \ x
\end{array}$$

As then the requirements become:

$$\begin{array}{rcl}
x & \geq & x \\
x + 1 & \geq & x \\
0 & \geq & 0 \\
x + 1 & \geq & x + 1 \\
x + 1 & > & x \\
x + 1 & > & x \\
x + 1 & > & x
\end{array}$$

Hence, we clearly gained something by using dependency pairs rather than using a reduction ordering directly!

Step-by-step proofs

Thus we see that dependency pairs can help with our first problem: proving termination of TRSs that cannot be handled with orderings like LPO or interpretations to $\mathbb{N}$ as they satisfy the subterm property. Now let us consider the second problem: dealing with **large** systems (for example with thousands of rules).

Through the dependency pair framework, we can prove termination **step-by-step**, by splitting large problems into a number of smaller ones that can be analysed separately.

For example, consider again the quot/minus example:

$$\begin{array}{llrcl}
\textbf{Rules:} & & \texttt{minus}(x,\texttt{0}) & \Rightarrow & x \\
 & & \texttt{minus}(\texttt{s}(x),\texttt{s}(y)) & \Rightarrow & \texttt{minus}(x,y) \\
 & & \texttt{quot}(\texttt{0},\texttt{s}(y)) & \Rightarrow & \texttt{0} \\
 & & \texttt{quot}(\texttt{s}(x),\texttt{s}(y)) & \Rightarrow & \texttt{s}(\texttt{quot}(\texttt{minus}(x,y),\texttt{s}(y))) \\
\textbf{DPs:} & \text{A} & \texttt{minus}^\sharp(\texttt{s}(x),\texttt{s}(y)) & \Rightarrow & \texttt{minus}^\sharp(x,y) \\
 & \text{B} & \texttt{quot}^\sharp(\texttt{s}(x),\texttt{s}(y)) & \Rightarrow & \texttt{quot}^\sharp(\texttt{minus}(x,y),\texttt{s}(y)) \\
 & \text{C} & \texttt{quot}^\sharp(\texttt{s}(x),\texttt{s}(y)) & \Rightarrow & \texttt{minus}^\sharp(x,y)
\end{array}$$

Consider the question: what does an infinite chain look like? Say, we have a chain $s_1 \Rightarrow t_1 \Rightarrow^* s_2 \Rightarrow t_2 \Rightarrow^* s_3 \ldots$. Observing that dependency pairs can only be used at the root of the term, and the rules only below the root, we know that:

- t_i and s_{i+1} always have the same root symbol, which must be $\texttt{minus}^\sharp$ or $\texttt{quot}^\sharp$;
- if s_i has a root symbol $\texttt{minus}^\sharp$, then the only dependency pair that can be used is A, so t_i and s_{i+1} also have $\texttt{minus}^\sharp$ as root symbol;
- so either *all* s_i have $\texttt{quot}^\sharp$ as root symbol, or eventually some s_i has $\texttt{min}^\sharp$, and afterwards all s_{i+j} have the same root;
- in conclusion: either there is an infinite chain where all root symbols are $\texttt{quot}^\sharp$, or there is an infinite chain where all root symbols are $\texttt{minus}^\sharp$;
- in the former case, this DP chain uses only DP A; in the latter case, it uses only DP C.

We conclude: to prove termination it suffices to *separately* prove that there is no infinite $(\{\text{A}\},\mathcal{R})$-chain, and that there is no infinite $(\{\text{C}\},\mathcal{R})$-chain.

The dependency pair framework defines a number of **DP processors**: functions that take a pair $(\texttt{DP},\mathcal{R})$ and return a set $\mathit{proc}(\texttt{DP},\mathcal{R})$ of pairs $(\texttt{DP}',\mathcal{R}')$ such that:

$$\begin{array}{c}
\text{There is an infinite } (\texttt{DP},\mathcal{R})\text{-chain} \\
\text{if and only if} \\
\text{there is some } (\texttt{DP}',\mathcal{R}') \in \mathit{proc}(\texttt{DP},\mathcal{R}) \text{ such that there is an infinite } (\texttt{DP}',\mathcal{R}')\text{-chain}
\end{array}$$

Thus, we can use the framework to gradually split and decrease a termination problem.

36

Mandatory material ends here

The material before this slide is expected knowledge, and you should be able to use monotonic algebras, and dependency pairs with weakly monotonic algebras, on the exam. You should also know the overall idea of the DP framework.

The following slides are optional material. However, you are *allowed* to use it on the exam; for example to answer a question "prove termination of this system", which can often be done much faster using the dependency pair framework with the graph and subterm criterion (see subsequent slides).

37

Using a graph

In the literature this idea is described as the **dependency graph processor**.

Idea: We place all dependency pairs in a graph, with an edge between pair ρ and pair μ if it is possible that ρ is directly followed by μ in a chain.

Example:

$$\begin{array}{lrcl} \text{A.} & \mathtt{minus}^\sharp(\mathtt{s}(x),\mathtt{s}(y)) & \Rightarrow & \mathtt{minus}^\sharp(x,y) \\ \text{B.} & \mathtt{quot}^\sharp(\mathtt{s}(x),\mathtt{s}(y)) & \Rightarrow & \mathtt{minus}^\sharp(x,y) \\ \text{C.} & \mathtt{quot}^\sharp(\mathtt{s}(x),\mathtt{s}(y)) & \Rightarrow & \mathtt{quot}^\sharp(\mathtt{minus}(x,y),\mathtt{s}(y)) \end{array}$$

$$\begin{tikzcd} \text{A} \arrow[loop left] & \text{B} \arrow[l] & \text{C} \arrow[l] \arrow[loop right] \end{tikzcd}$$

Observation: each **strongly connected component** may be considered separately.

38

Dependency graph processor – another example

$$R = \{\mathtt{f}(\mathtt{f}(x)) \ \rightarrow\ \mathtt{f}(\mathtt{g}(\mathtt{f}(x)))\}$$

$$\mathtt{DP} = \left\{ \begin{array}{lrcl} \text{A.} & \mathtt{f}^\sharp(\mathtt{f}(x)) & \Rightarrow & \mathtt{f}^\sharp(\mathtt{g}(\mathtt{f}(x))) \\ \text{B.} & \mathtt{f}^\sharp(\mathtt{f}(x)) & \Rightarrow & \mathtt{f}^\sharp(x) \end{array} \right\}$$

(Because $\mathtt{g}(\mathtt{f}(t))$ does not reduce to $\mathtt{f}(q)$ for any terms t, q.)

$$\begin{tikzcd} \text{A} & \text{B} \arrow[l] \arrow[loop right] \end{tikzcd}$$

Hence, we can remove dependency pair A.

It is not always easy to determine if one dependency pair can follow another, but we can always make an overapproximation (for example by only looking at the marked symbols $\mathtt{f}^\sharp$ on either side of the pair, and the constructors directly beneath a marked symbol).

39

Dependency graph processor

Formally, the "processor" is formulated as follows:

Definition

Let $(\mathcal{D}, R)$ be a DP problem, and G a graph whose nodes are the elements of $\mathcal{D}$, and which has an edge from ρ to μ if it is possible for ρ to be followed by μ in a $(\mathcal{D}, R)$-chain (there may be more edges than this).

Suppose $A_1, \dots, A_n$ are the **strongly connected components** of G.

Then the dependency graph processor maps $(\mathcal{D}, R)$ to $\{(A_1, R), \dots, (A_n, R)\}$.

This processor *proc* satisfies the very desirable property: if $(\mathcal{D}, R)$ admits an infinite chain, then some element of *proc*$(\mathcal{D}, R)$ does too. This property is called *soundness* in the literature.

40

The subterm criterion

$$\begin{array}{lrcl}
\text{A.} & \mathtt{exp}^\sharp(\mathtt{s}(x), y) & \Rightarrow & \mathtt{double}^\sharp(x, y, \mathtt{0}) \\
\text{B.} & \mathtt{double}^\sharp(x, \mathtt{0}, z) & \Rightarrow & \mathtt{exp}^\sharp(x, z) \\
\text{C.} & \mathtt{double}^\sharp(x, \mathtt{0}, z) & \Rightarrow & \mathtt{double}^\sharp(x, y, \mathtt{s}(\mathtt{s}(z)))
\end{array}$$

Idea: consider the **first argument** of each side of the dependency pairs. In a DP step, this is either unchanged, or replaced by a strict subterm.

$$\begin{array}{lrcl}
\text{A.} & \mathtt{s}(x) & \text{goes to} & x \\
\text{B.} & x & \text{goes to} & x \\
\text{C.} & x & \text{goes to} & x
\end{array}$$

Then we can remove the dependency pairs where the chosen argument becomes smaller (in this case A).

41

The subterm criterion processor

Definition

For all marked symbols $\mathtt{f}^\sharp$, let $\nu(\mathtt{f}^\sharp) \in \{1, \dots, \mathit{arity}(\mathtt{f})\}$.

Let $(\mathcal{D}, R)$ be a DP problem, and write $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2$ for disjoint sets $\mathcal{D}_1, \mathcal{D}_2$.

Suppose:

- $\ell_{\nu(\mathtt{f}^\sharp)} = r_{\nu(\mathtt{g}^\sharp)}$ for all $\mathtt{f}^\sharp(\ell_1, \dots, \ell_n) \Rightarrow \mathtt{g}^\sharp(r_1, \dots, r_m) \in \mathcal{D}_1$
- $r_{\nu(\mathtt{f}^\sharp)}$ is a subterm of $\ell_{\nu(\mathtt{g}^\sharp)}$ for all $\mathtt{f}^\sharp(\ell_1, \dots, \ell_n) \Rightarrow \mathtt{g}^\sharp(r_1, \dots, r_m) \in \mathcal{D}_2$

Then the subterm criterion processor maps $(\mathcal{D}, R)$ to $\{(\mathcal{D}_1, R)\}$, or to $\emptyset$ if $\mathcal{D}_1 = \emptyset$.

This processor also satisfies the soundness property as described above.

Implementation: This is implemented with a simple SMT implementation using integer variables $\nu(\mathtt{f}^\sharp)$, and boolean variables $\mathtt{strict}_\rho$ for each DP ρ. The code itself can check for every pair (ℓ_i, r_j) if $\ell_i = r_j$ or r_j is a subterm of ℓ_i, and set up requirements on $\nu(\mathtt{f}^\sharp)$, $\nu(\mathtt{g}^\sharp)$ and $\mathtt{strict}_{\mathtt{f}^\sharp(\ell_1,\dots,\ell_n) \Rightarrow \mathtt{g}^\sharp(r_1,\dots,r_m)}$ accordingly.

42

Graph + subterm criterion

The dependency graph processor and the subterm criterion processor are a powerful combination! In many practical cases, just these two suffice to prove termination.

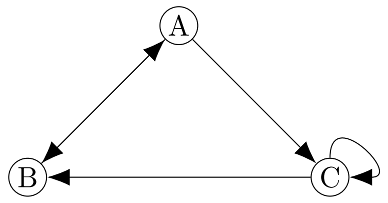
Class exercise:

$$\begin{array}{rcl}
\mathtt{exp}(\mathtt{0}, y) & \Rightarrow & y \\
\mathtt{exp}(\mathtt{s}(x), y) & \Rightarrow & \mathtt{double}(\mathtt{0}, x, y) \\
\mathtt{double}(r, x, \mathtt{0}) & \Rightarrow & \mathtt{exp}(x, r) \\
\mathtt{double}(r, x, \mathtt{s}(y)) & \Rightarrow & \mathtt{double}(\mathtt{s}(\mathtt{s}(r)), x, y)
\end{array}$$

Solution: Dependency pairs are:

$$\begin{array}{lrcl}
A. & \mathtt{exp}^\sharp(\mathtt{s}(x), y) & \Rightarrow & \mathtt{double}^\sharp(\mathtt{0}, x, y) \\
B. & \mathtt{double}^\sharp(r, x, \mathtt{0}) & \Rightarrow & \mathtt{exp}^\sharp(x, r) \\
C. & \mathtt{double}^\sharp(r, x, \mathtt{s}(y)) & \Rightarrow & \mathtt{double}^\sharp(\mathtt{s}(\mathtt{s}(r)), x, y)
\end{array}$$

This gives the following graph:



Nothing can be removed so far. However, by the subterm criterion with $\nu(\mathtt{exp}^\sharp) = 1$ and $\nu(\mathtt{double}^\sharp) = 2$, we can remove A. This leaves only $\{B, C\}$, which are placed in the following graph:

$$\begin{tikzcd} B & C \arrow[l] \arrow[loop right] \end{tikzcd}$$

Note that B is not part of a strongly connected component, so can be removed. This leaves only $\{C\}$. But $\{C\}$ can be removed using the subterm criterion with $\nu(\mathtt{double}^\sharp) = 2$.

As there are no dependency pairs left, we conclude that there is no infinite chain. Hence, the original TRS is terminating.

3.5 Other processors

We will briefly consider a few more processors. You are not expected to know these, but if you are interested in the area, it is good to know of their existence.

43

Reduction pair processor

We can also reformulate reduction pairs as a processor:

Definition

Let $\succ$ be a well-founded, stable ordering and $\succeq$ a stable monotonic quasi-ordering on the set of terms, such that $\succ\succeq\ \subseteq\ \succ$.

Let $(\mathcal{D}, R)$ be a DP problem, and write $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2$ for disjoint sets $\mathcal{D}_1, \mathcal{D}_2$.

Suppose:

- $\ell \succeq r$ for all $\ell \to r \in R$
- $\ell \succeq r$ for all $\ell \Rightarrow r \in \mathcal{D}_1$, and
- $\ell \succ r$ for all $\ell \Rightarrow r \in \mathcal{D}_2$.

Then the reduction pair processor maps $(\mathcal{D}, R)$ to $\{(\mathcal{D}_1, R)\}$, or to $\emptyset$ if $\mathcal{D}_1 = \emptyset$.

That is, we can use a reduction pair, and remove all dependency pairs that were ordered with $\succ$.

44

Example: using a reduction pair processor

$$\begin{array}{rcl}
\mathtt{append}(\mathtt{nil}, z) & \to & z \\
\mathtt{append}(\mathtt{cons}(x, y), z) & \to & \mathtt{cons}(x, \mathtt{append}(y, z)) \\
\mathtt{rev}(\mathtt{nil}) & \to & \mathtt{nil} \\
\mathtt{rev}(\mathtt{cons}(x, y)) & \to & \mathtt{append}(\mathtt{rev}(y), \mathtt{cons}(x, \mathtt{nil})) \\
\mathtt{append}^\sharp(\mathtt{cons}(x, y), z) & \Rightarrow & \mathtt{append}^\sharp(y, z) \\
\mathtt{rev}^\sharp(\mathtt{cons}(x, y)) & \Rightarrow & \mathtt{rev}^\sharp(y) \\
\mathtt{rev}^\sharp(\mathtt{cons}(x, y)) & \Rightarrow & \mathtt{append}^\sharp(\mathtt{rev}(y), \mathtt{cons}(x, \mathtt{nil}))
\end{array}$$

We choose:

$$\begin{array}{rcl} [\texttt{nil}] & = & 0 \\ [\texttt{cons}] & = & \lambda x,y.\ y+1 \end{array} \qquad \begin{array}{rcll} [\texttt{append}] & = & \lambda x,y. & x+y \\ [\texttt{rev}] & = & \lambda x. & x \\ [\texttt{append}^\sharp] & = & \lambda x,y. & x+y \\ [\texttt{rev}^\sharp] & = & \lambda x. & x \end{array}$$

Then the weight functions for the rules and dependency pairs translate to:

$$\begin{array}{rcl} z & \geq & z \\ (y+1)+z & \geq & (y+z)+1 \\ 0 & \geq & 0 \\ y+1 & \geq & y+(0+1) \\ (y+1)+z & > & y+z \\ y+1 & > & y \\ y+1 & \geq & y+(0+1) \end{array}$$

Hence, we can remove all but the last dependency pair, and continue with the DP problem:

$$(\{\texttt{rev}^\sharp(\texttt{cons}(x,y)) \Rightarrow \texttt{append}^\sharp(\texttt{rev}(y),\texttt{cons}(x,\texttt{nil}))\}, R)$$

(Note that in this case, we could also have removed all DPs in one go by choosing $[\texttt{rev}^\sharp] = \lambda x.2 * x$. Although it is often useful, we are not obliged to choose $[\texttt{f}^\sharp] = [\texttt{f}]$!)

Argument filterings

To turn a reduction ordering into a reduction pair, we can combine it with a transformation to remove arguments.

For example, given the requirements:

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

We may replace each occurrence of $\texttt{minus}(x,y)$ by $\texttt{minus}'(x)$, and leave other symbols unchanged. Then the requirements become:

$$\begin{array}{rcl} \texttt{minus}'(x) & \succeq & x \\ \texttt{minus}'(\texttt{s}(x)) & \succeq & \texttt{minus}'(x) \\ \texttt{quot}(\texttt{0},\texttt{s}(y)) & \succeq & \texttt{0} \\ \texttt{quot}(\texttt{s}(x),\texttt{s}(y)) & \succeq & \texttt{s}(\texttt{quot}(\texttt{minus}'(x),\texttt{s}(y))) \\ \texttt{minus}^\sharp(\texttt{s}(x),\texttt{s}(y)) & \succ & \texttt{minus}^\sharp(x,y) \\ \texttt{quot}^\sharp(\texttt{s}(x),\texttt{s}(y)) & \succ & \texttt{quot}^\sharp(\texttt{minus}'(x),\texttt{s}(y)) \\ \texttt{quot}(^\sharp\texttt{s}(x),\texttt{s}(y)) & \succ & \texttt{minus}^\sharp(x,y) \end{array}$$

This can be handled using LPO with $\mathtt{s} \rhd \mathtt{minus}'$.

In implementations of recursive path orderings, the choice of argument filtering is typically included in the SAT encoding.

Usable rules

When considering only a small number of dependency pairs, we might not need all the rules anymore.

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

Here, the one dependency pair calls the `minus` function, but not the `quot` function.

Also the rules for `minus` do not call `quot`.

Hence, we do not need to consider the two `quot`-rules when finding a reduction pair.

We do get two extra requirements, $\mathtt{c}(x,y) \succeq x$ and $\mathtt{c}(x,y) \succeq y$. But this is not a problem for most reduction pairs.

Finding usable rules is a very simple reachability algorithm.

We can also combine usable rules with an argument filtering to eliminate even more rules. This is typically done with SAT or SMT, in combination with the search for a reduction pair.

4. Quiz

Quiz

1. Prove termination of the following TRS using a monotonic algebra to $\mathbb{N}$:

$$\begin{array}{rcl} \texttt{append}(\texttt{nil}, z) & \to & z \\ \texttt{append}(\texttt{cons}(x, y), z) & \to & \texttt{cons}(x, \texttt{append}(y, z)) \end{array}$$

   - give (linear) parametric interpretations for all function symbols
   - compute the requirements (monotonicity and rule orientation)
   - use absolute positiveness to find SMT requirements
   - solve them by hand and give the resulting inteprretation functions, and check your result!

2. Determine the dependency pairs of:

$$\begin{array}{rcl} \texttt{f}(\texttt{h}(x), y) & \to & \texttt{g}(x, \texttt{f}(x, \texttt{h}(y))) \\ \texttt{g}(x, \texttt{h}(y)) & \to & \texttt{g}(\texttt{h}(x), y) \end{array}$$

3. Split these dependency pairs up into one or more groups of DPs that can be analysed separately.
   (Or, if you read the extra material: calculate a graph approximation for these dependency pairs, and determine the result of the dependency graph processor.)

4. Prove termination of the above TRS.