Rewriting - Computation and Deduction

Horatiu CIRSTEA, Hélène KIRCHNER

INRIA

Nancy and Bordeaux, France

October 2008

Companion Document : `www.loria.fr/~hkirchne`

Information Science and Rewriting

Information science and technology address

- data representation
- data transformation

Information Science and Rewriting

Information science and technology address

- data representation
- data transformation

What about Rewriting in this context ?

Information Science and Rewriting

Information science and technology address

- data representation
- data transformation

What about Rewriting in this context ?

- data are terms or more generally structured objects

Information Science and Rewriting

Information science and technology address

- data representation
- data transformation

What about Rewriting in this context ?

- data are terms or more generally structured objects
- this is a way to describe transformations of these objects

Information Science and Rewriting

Information science and technology address

- data representation
- data transformation

What about Rewriting in this context ?

- data are terms or more generally structured objects
- this is a way to describe transformations of these objects
- it allows formalizing and analysing the relations between these objects

What can you do with Rewriting ?

Can rewriting be used

- for formal specifications ?

What can you do with Rewriting ?

Can rewriting be used

- for formal specifications ?
  functional or algebraic framework, express and check properties of specifications.

What can you do with Rewriting ?

Can rewriting be used

- for formal specifications ?
  functional or algebraic framework, express and check properties of specifications.
- as a programming langage ?

What can you do with Rewriting ?

Can rewriting be used

- for formal specifications ?
  functional or algebraic framework, express and check properties of specifications.
- as a programming langage ?
  high-level, type discipline, prototyping, efficient compilation

What can you do with Rewriting ?

Can rewriting be used

- for formal specifications ?
  functional or algebraic framework, express and check properties of specifications.
- as a programming langage ?
  high-level, type discipline, prototyping, efficient compilation
- in a proof environnement ?

What can you do with Rewriting ?

Can rewriting be used

- for formal specifications ?
  functional or algebraic framework, express and check properties of specifications.
- as a programming langage ?
  high-level, type discipline, prototyping, efficient compilation
- in a proof environnement ?
  equality in first-order theories, computational part of proofs, as a logic and a higher-order calculus.

(Some) Additional Recommended Readings

- Term Rewriting Systems
  Terese (M. Bezem, J. W. Klop and R. de Vrijer, eds.)
  Cambridge Univerty press, 2002
- Term *Re*writing and *all That*
  Franz Baader and Tobias Nipkow
  Cambridge Univerty press, 1998
- Repository of Lectures on Rewriting and Related Topics
  `qsl.loria.fr`
- The rewriting and IFIP WG1.6 page
  `rewriting.loria.fr`
- The Rewriting Calculus Home page
  `rho.loria.fr`

A simple game

The rules of the game :

●● ⟶ ○

○○ ⟶ ○

●○ ⟶ ●

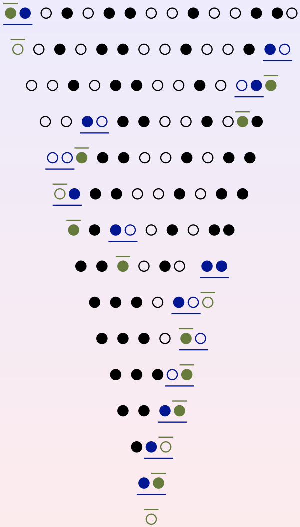
○● ⟶ ●

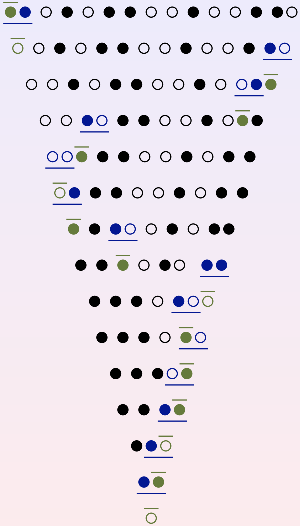
A starting point :

● ○ ● ○ ● ○ ● ● ● ● ○ ○ ● ○ ○ ● ●○

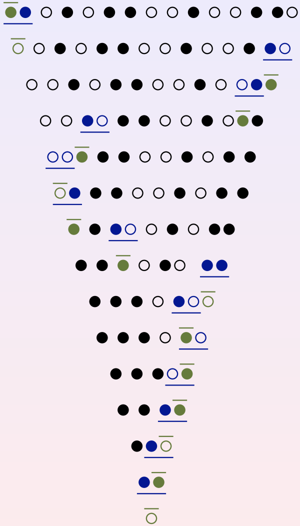
Who wins ?

➥ Who puts the last white ?

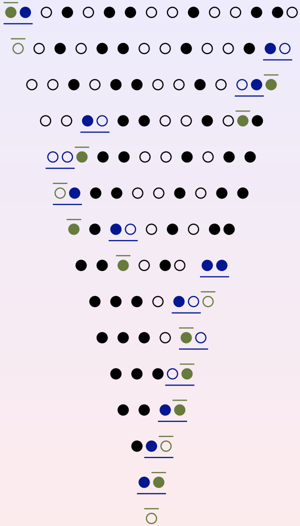




Can I always win ?



Can I always win ? Does the game terminate ?



Can I always win ? Does the game terminate ? Do we always get the same result ?

What are the basic operations that have been used ?

What are the basic operations that have been used ?

1– Matching

The data : ●● ●○ ○ ● ○ ●●

The rewrite rule : ●○ → ●

What are the basic operations that have been used ?

1– Matching

The data : ●● ●○ ○ ● ○ ●●

The rewrite rule : ●○ → ●

2– Compute what should be substituted

The lefthand side : ●

What are the basic operations that have been used ?

1– Matching

The data : ●● ●○ ○ ● ○ ●●

The rewrite rule : ●○ ⟶▷ ●

2– Compute what should be substituted

The lefthand side : ●

3– Replacement

The new generated data : ● ● $\boxed{\bullet}$ ○ ● ○ ●●

What are the basic operations that have been used ?

1– Matching

The data : ●● ●○ ○ ● ○ ●●

The rewrite rule : ●○ → ●

2– Compute what should be substituted

The lefthand side : ●

3– Replacement

The new generated data : ● ● ● ○ ● ○ ●●

Note that the last list is a NEW object.

Addition in Peano arithmetic

Peano gives a meaning to addition by using the following axioms :

$$0 + x = x$$
$$s(x) + y = s(x + y)$$

Addition in Peano arithmetic

Peano gives a meaning to addition by using the following axioms :

$$0 + x = x$$
$$s(x) + y = s(x + y)$$

What's the result of $s(s(0)) + s(s(0))$?

Addition in Peano arithmetic

Peano gives a meaning to addition by using the following axioms :

$$0 + x = x$$
$$s(x) + y = s(x + y)$$

What's the result of $s(s(0)) + s(s(0))$?

$$s(s(0)) + s(s(0)) = s(s(0) + s(s(0)))$$

Addition in Peano arithmetic

Peano gives a meaning to addition by using the following axioms :

$$0 + x = x$$
$$s(x) + y = s(x + y)$$

What's the result of $s(s(0)) + s(s(0))$?

$$\begin{aligned} s(s(0)) + s(s(0)) &= s(s(0) + s(s(0))) \\ &= s(s(0 + s(s(0)))) \end{aligned}$$

Addition in Peano arithmetic

Peano gives a meaning to addition by using the following axioms :

$$
\begin{aligned}
0 + x &= x \\
s(x) + y &= s(x + y)
\end{aligned}
$$

What's the result of $s(s(0)) + s(s(0))$?

$$
\begin{aligned}
s(s(0)) + s(s(0)) &= s(s(0) + s(s(0))) \\
&= s(s(0 + s(s(0)))) \\
&= s(s(s(s(0)))) \\
&= s(0) + s(s(s(0))) \\
&= 0 + 0 + 0 + s(s(s(s(0)))) \\
&= \ldots
\end{aligned}
$$

Addition in Peano arithmetic

Peano gives a meaning to addition by using the following axioms :

$$0 + x = x$$
$$s(x) + y = s(x + y)$$

What's the result of $s(s(0)) + s(s(0))$?

$$\begin{aligned} s(s(0)) + s(s(0)) &= s(s(0) + s(s(0))) \\ &= s(s(0 + s(s(0)))) \\ &= s(s(s(s(0)))) \\ &= s(0) + s(s(s(0))) \\ &= 0 + 0 + 0 + s(s(s(s(0)))) \\ &= \ldots \end{aligned}$$

Is there a *better* result ?

Addition in Peano arithmetic

Compute a result by turning the equalities into rewrite rules :

$$\begin{aligned} 0 + x &\rightarrow x \\ s(x) + y &\rightarrow s(x + y) \end{aligned}$$

Addition in Peano arithmetic

Compute a result by turning the equalities into rewrite rules :

$$0 + x \rightarrow x$$
$$s(x) + y \rightarrow s(x + y)$$

$s(s(0)) + s(s(0)) \rightarrow s(s(0) + s(s(0))) \rightarrow$
$s(s(0 + s(s(0)))) \rightarrow$
$s(s(s(s(0))))$

Addition in Peano arithmetic

Compute a result by turning the equalities into rewrite rules :

$$0 + x \rightarrow x$$
$$s(x) + y \rightarrow s(x + y)$$

$s(s(0)) + s(s(0)) \rightarrow s(s(0) + s(s(0))) \rightarrow$
$s(s(0 + s(s(0)))) \rightarrow$
$s(s(s(s(0))))$

Is this computation *terminating* ,

Addition in Peano arithmetic

Compute a result by turning the equalities into rewrite rules :

$$0 + x \rightarrow x$$
$$s(x) + y \rightarrow s(x + y)$$

$s(s(0)) + s(s(0)) \rightarrow s(s(0) + s(s(0)) \rightarrow$
$s(s(0 + s(s(0)))) \rightarrow$
$s(s(s(s(0))))$

Is this computation *terminating*,

is there always a *result* (e.g. an expression without $+$)

Addition in Peano arithmetic

Compute a result by turning the equalities into rewrite rules :

$$0 + x \to x$$
$$s(x) + y \to s(x + y)$$

$s(s(0)) + s(s(0)) \to s(s(0) + s(s(0)) \to$
$s(s(0 + s(s(0)))) \to$
$s(s(s(s(0))))$

Is this computation *terminating*,

is there always a *result* (e.g. an expression without $+$)

is such a result *unique* ? ? ?

Addition in Peano arithmetic

Compute a result by turning the equalities into rewrite rules :

$$0 + x \rightarrow x$$
$$s(x) + y \rightarrow s(x + y)$$

$s(s(0)) + s(s(0)) \rightarrow s(s(0) + s(s(0))) \rightarrow$
$s(s(0 + s(s(0)))) \rightarrow$
$s(s(s(s(0))))$

Is this computation *terminating* ,

is there always a *result* (e.g. an expression without $+$)

is such a result *unique* ? ? ?

What are the basic operations that have been used ?

What are the basic operations that have been used ?

1– Matching

The data : $s(s(0)) + s(s(0))$

The rewrite rule : $s(x) + y \to s(x + y)$

What are the basic operations that have been used ?

1– Matching

The data : $s(\boxed{s(0)}) + \boxed{s(s(0))}$

The rewrite rule : $s(\boxed{x}) + \boxed{y} \rightarrow s(x+y)$

2– Compute what should be substituted

The instanciated lhs : $s(\boxed{s(0)} + \boxed{s(s(0))})$

What are the basic operations that have been used ?

1– Matching

The data : $s(s(0)) + s(s(0))$

The rewrite rule : $s(x) + y \rightarrow s(x + y)$

2– Compute what should be substituted

The instanciated lhs : $s(s(0) + s(s(0)))$

3– Replacement

The new generated data : s(s(0)+s(s(0)))

What are the basic operations that have been used ?

1– Matching

The data : $s(\boxed{s(0)}) + \boxed{s(s(0))}$

The rewrite rule : $s(\boxed{x}) + \boxed{y} \to s(x + y)$

2– Compute what should be substituted

The instanciated lhs : $s(\boxed{s(0)} + \boxed{s(s(0))})$

3– Replacement

The new generated data : $\boxed{\text{s(s(0)+s(s(0)))}}$

Note that this last entity is a NEW object.

Fibonacci

$$\begin{array}{llll} [\alpha] & fib(0) & \rightarrow & 1 \\ [\beta] & fib(1) & \rightarrow & 1 \\ [\gamma] & fib(n) & \rightarrow & fib(n-1) + fib(n-2) \end{array}$$

$fib(\boxed{3})$

Fibonacci

$$\begin{array}{llll} [\alpha] & fib(0) & \rightarrow & 1 \\ [\beta] & fib(1) & \rightarrow & 1 \\ [\gamma] & fib(n) & \rightarrow & fib(n-1) + fib(n-2) \end{array}$$

$fib(3) \rightarrow fib(2) + fib(1)$

Fibonacci

$$\begin{array}{llll} [\alpha] & fib(0) & \rightarrow & 1 \\ [\beta] & fib(1) & \rightarrow & 1 \\ [\gamma] & fib(n) & \rightarrow & fib(n-1) + fib(n-2) \end{array}$$

$$\boxed{fib(\boxed{3})} \rightarrow \boxed{fib(\boxed{2}) + fib(\boxed{1})}$$

$$fib(2) + \boxed{fib(1)}$$

Fibonacci

$$
\begin{array}{llll}
[\alpha] & fib(0) & \rightarrow & 1 \\
[\beta] & fib(1) & \rightarrow & 1 \\
[\gamma] & fib(n) & \rightarrow & fib(n-1) + fib(n-2)
\end{array}
$$

$$
\begin{array}{lll}
fib(\boxed{3}) \rightarrow & fib(\boxed{2}) + fib(\boxed{1}) & \\
& fib(2) + fib(1) \rightarrow & fib(2) + 1
\end{array}
$$

Fibonacci

$$
\begin{array}{llll}
[\alpha] & fib(0) & \rightarrow & 1 \\
[\beta] & fib(1) & \rightarrow & 1 \\
[\gamma] & fib(n) & \rightarrow & fib(n-1) + fib(n-2)
\end{array}
$$

$$
\begin{array}{llll}
fib(3) & \rightarrow & fib(2) + fib(1) & \\
 & & fib(2) + fib(1) & \rightarrow \quad fib(2) + 1 \\
 & & & \quad fib(2) + 1
\end{array}
$$

Fibonacci

$$
\begin{array}{llll}
[\alpha] & \mathit{fib}(0) & \rightarrow & 1 \\
[\beta] & \mathit{fib}(1) & \rightarrow & 1 \\
[\gamma] & \mathit{fib}(n) & \rightarrow & \mathit{fib}(n-1) + \mathit{fib}(n-2)
\end{array}
$$

$$
\boxed{\mathit{fib}(\boxed{3})} \rightarrow \boxed{\mathit{fib}(\boxed{2}) + \mathit{fib}(\boxed{1})}
$$

$$
\mathit{fib}(2) + \boxed{\mathit{fib}(1)} \rightarrow \mathit{fib}(2) + \boxed{1}
$$

$$
\boxed{\mathit{fib}(\boxed{2})} + 1 \rightarrow \boxed{\mathit{fib}(\boxed{1}) + \mathit{fib}(\boxed{0})} + 1
$$

Fibonacci

$$
\begin{array}{llcl}
[\alpha] & fib(0) & \rightarrow & 1 \\
[\beta] & fib(1) & \rightarrow & 1 \\
[\gamma] & fib(n) & \rightarrow & fib(n-1) + fib(n-2)
\end{array}
$$

$$
\begin{array}{l}
\boxed{fib(\boxed{3})} \rightarrow \boxed{fib(\boxed{2}) + fib(\boxed{1})} \\
\quad fib(2) + \boxed{fib(1)} \rightarrow fib(2) + \boxed{1} \\
\quad\quad \boxed{fib(\boxed{2})} + 1 \rightarrow \boxed{fib(\boxed{1}) + fib(\boxed{0})} + 1 \\
\quad\quad\quad \boxed{fib(1)} + fib(0) + 1
\end{array}
$$

Fibonacci

$$
\begin{array}{llll}
[\alpha] & fib(0) & \rightarrow & 1 \\
[\beta] & fib(1) & \rightarrow & 1 \\
[\gamma] & fib(n) & \rightarrow & fib(n-1) + fib(n-2)
\end{array}
$$

$$fib(3) \rightarrow fib(2) + fib(1)$$

$$fib(2) + fib(1) \rightarrow fib(2) + 1$$

$$fib(2) + 1 \rightarrow fib(1) + fib(0) + 1$$

$$fib(1) + fib(0) + 1 \rightarrow 1 + fib(0) + 1$$

Fibonacci

$$
\begin{array}{llll}
[\alpha] & fib(0) & \rightarrow & 1 \\
[\beta] & fib(1) & \rightarrow & 1 \\
[\gamma] & fib(n) & \rightarrow & fib(n-1) + fib(n-2)
\end{array}
$$

$\boxed{fib(\boxed{3})} \rightarrow \boxed{fib(\boxed{2}) + fib(\boxed{1})}$

$fib(2) + \boxed{fib(1)} \rightarrow fib(2) + \boxed{1}$

$\boxed{fib(\boxed{2})} + 1 \rightarrow \boxed{fib(\boxed{1}) + fib(\boxed{0})} + 1$

$\boxed{fib(1)} + fib(0) + 1 \rightarrow \boxed{1} + fib(0) + 1$

$\ldots$

Finally $fib(3) = 3$, $fib(4) = 5$, ...

Graphical Rewriting

$$F \rightarrow F + F - F - FF + F + F - F$$

Graphical Rewriting

$$F \rightarrowtriangle F + F - F - FF + F + F - F$$

Graphical Rewriting

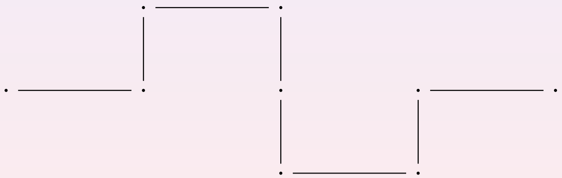
$$F \;\rightarrowtriangle\; F + F - F - FF + F + F - F$$

$\rightarrowtriangle$

Graphical Rewriting

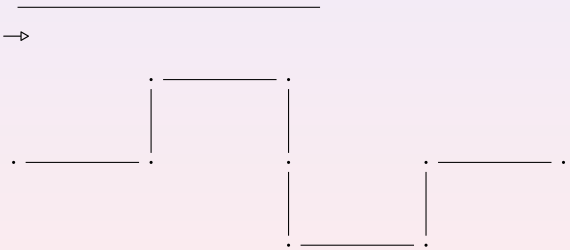
$$F \rightarrow F + F - F - FF + F + F - F$$

$\rightarrow$



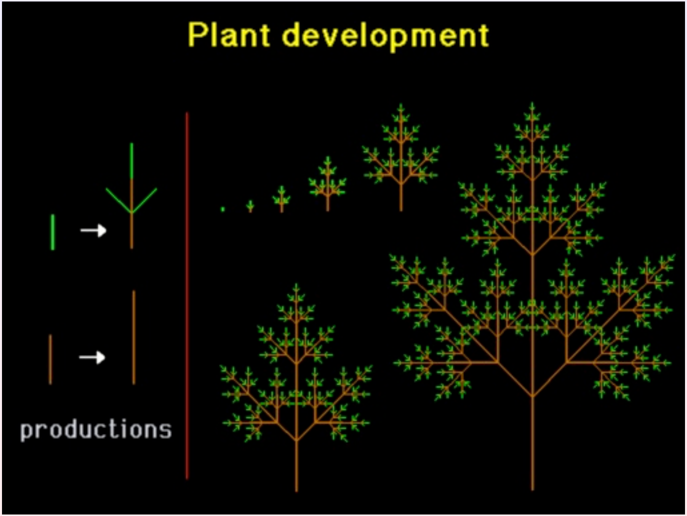
Graphical Rewriting

$$F \rightarrowtriangle F + F - F - FF + F + F - F$$



L-systems (Lindenmeier)

Ecological Rewriting



http ://algorithmicbotany.org/

Sorting by rewriting

```
rules for List
  X, Y : Nat ; L L' L'' : List;
   hd (X L) => X ;                       tl (X L) => L ;
   sort nil => nil .
   sort (L X L' Y L'') =>  sort (L Y L' X L'')  if Y < X .
end

sort (6 5 4 3 2 1) => ...
```

On what objects can rewriting act ?

It can be defined on

- terms like 2 + i(3) or XML documents
- strings like "What is rewriting ?" (`sed` performs string rewriting)
- graphs
- sets
- multisets
- ...

On what objects can rewriting act ?

It can be defined on

- terms like 2 + i(3) or XML documents
- strings like “What is rewriting ?” (`sed` performs string rewriting)
- graphs
- sets
- multisets
- ...

We will “restrict” in this lecture to first-order terms

The relation, the logic, the calculus

This part deals with the
rewriting relation
on
first-order term

This is just the oriented version of replacement of equal by equal

First-order terms

Signature and first-order terms

$\mathcal{F}_0$ a set of symbols of arity 0 (the constants)

$\mathcal{F}_i$ a set of symbols of arity i

$\mathcal{F} = \cup_n \mathcal{F}_n$

$\mathcal{X}$ a set of arity 0 symbols called variables.

$\mathcal{T}(\mathcal{F},\mathcal{X})$ is the smallest set such that :

- $\mathcal{X} \subseteq \mathcal{T}(\mathcal{F},\mathcal{X})$,
- $\forall f \in \mathcal{F}, \forall t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F},\mathcal{X}) : \quad f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F},\mathcal{X})$.

$\mathcal{T}(\mathcal{F},\emptyset) = \mathcal{T}(\mathcal{F})$ is the set of ground terms.

Terms as mappings : $(\mathbf{N}, .) \quad \rightharpoonup \quad \mathcal{F}$

$t = f(a + x, h(f(a, b)))$ is represented by :

$$\begin{array}{lcr}
position & \mapsto & symbol \\
\Lambda & \mapsto & f \\
1 & \mapsto & + \\
1.1 & \mapsto & a \\
1.2 & \mapsto & x \\
2 & \mapsto & h \\
2.1 & \mapsto & f \\
2.1.1 & \mapsto & a \\
2.1.2 & \mapsto & b
\end{array}$$

$\mathcal{D}om(t) = \{\Lambda, 1, 1.1, 1.2, 2, 2.1, 2.1.1, 2.1.2\}$

Examples and (some) terminology

With the following signature :

$\mathcal{F} = \{f, a\}$ with $arity(f) = 2$, $arity(a) = 0$, $x, y, z \in \mathcal{X}$:

what are the following terms ?

$f(a, a)$

$f(x, f(a, x))$

$f(x, f(y, z))$

Examples and (some) terminology

With the following signature :

$\mathcal{F} = \{f, a\}$ with $arity(f) = 2$, $arity(a) = 0$, $x, y, z \in \mathcal{X}$:

what are the following terms ?

$f(a, a)$ is ground,

$f(x, f(a, x))$

$f(x, f(y, z))$

Examples and (some) terminology

With the following signature :

$\mathcal{F} = \{f, a\}$ with $arity(f) = 2$, $arity(a) = 0$, $x, y, z \in \mathcal{X}$:

what are the following terms ?

$f(a, a)$ is ground,

$f(x, f(a, x))$ is not linear but

$f(x, f(y, z))$

Examples and (some) terminology

With the following signature :

$\mathcal{F} = \{f, a\}$ with $arity(f) = 2$, $arity(a) = 0$, $x, y, z \in \mathcal{X}$:

what are the following terms ?

$f(a, a)$ is ground,

$f(x, f(a, x))$ is not linear but

$f(x, f(y, z))$ is linear

What about the following terms ?

Examples and (some) terminology

With the following signature :

$\mathcal{F} = \{f, a\}$ with $arity(f) = 2$, $arity(a) = 0$, $x, y, z \in \mathcal{X}$:

what are the following terms ?

$f(a, a)$ is ground,

$f(x, f(a, x))$ is not linear but

$f(x, f(y, z))$ is linear

What about the following terms ?

$f(a, a, a)$ is

Examples and (some) terminology

With the following signature :

$\mathcal{F} = \{f, a\}$ with $arity(f) = 2$, $arity(a) = 0$, $x, y, z \in \mathcal{X}$:

what are the following terms ?

$f(a, a)$ is ground,

$f(x, f(a, x))$ is not linear but

$f(x, f(y, z))$ is linear

What about the following terms ?

$f(a, a, a)$ is ill-formed (since f is of arity 2)

Examples and (some) terminology

With the following signature :

$\mathcal{F} = \{f, a\}$ with $arity(f) = 2$, $arity(a) = 0$, $x, y, z \in \mathcal{X}$:

what are the following terms ?

$f(a, a)$ is ground,

$f(x, f(a, x))$ is not linear but

$f(x, f(y, z))$ is linear

What about the following terms ?

$f(a, a, a)$ is ill-formed (since f is of arity 2)

a is

Examples and (some) terminology

With the following signature :

$\mathcal{F} = \{f, a\}$ with $arity(f) = 2$, $arity(a) = 0$, $x, y, z \in \mathcal{X}$:

what are the following terms ?

$f(a, a)$ is ground,

$f(x, f(a, x))$ is not linear but

$f(x, f(y, z))$ is linear

What about the following terms ?

$f(a, a, a)$ is ill-formed (since f is of arity 2)

a is correct

Examples and (some) terminology

With the following signature :

$\mathcal{F} = \{f, a\}$ with $arity(f) = 2$, $arity(a) = 0$, $x, y, z \in \mathcal{X}$:

what are the following terms ?

$f(a, a)$ is ground,

$f(x, f(a, x))$ is not linear but

$f(x, f(y, z))$ is linear

What about the following terms ?

$f(a, a, a)$ is ill-formed (since f is of arity 2)

a is correct

$x(a)$ is

Examples and (some) terminology

With the following signature :

$\mathcal{F} = \{f, a\}$ with $arity(f) = 2$, $arity(a) = 0$, $x, y, z \in \mathcal{X}$:

what are the following terms ?

$f(a, a)$ is ground,

$f(x, f(a, x))$ is not linear but

$f(x, f(y, z))$ is linear

What about the following terms ?

$f(a, a, a)$ is ill-formed (since f is of arity 2)

a is correct

$x(a)$ is ill-formed (since all variables are assumed of arity 0)

Examples and (some) terminology

With the following signature :

$\mathcal{F} = \{f, a\}$ with $arity(f) = 2$, $arity(a) = 0$, $x, y, z \in \mathcal{X}$:

what are the following terms ?

$f(a, a)$ is ground,

$f(x, f(a, x))$ is not linear but

$f(x, f(y, z))$ is linear

What about the following terms ?

$f(a, a, a)$ is ill-formed (since f is of arity 2)

a is correct

$x(a)$ is ill-formed (since all variables are assumed of arity 0)

f is

Examples and (some) terminology

With the following signature :

$\mathcal{F} = \{f, a\}$ with $arity(f) = 2$, $arity(a) = 0$, $x, y, z \in \mathcal{X}$:

what are the following terms ?

$f(a, a)$ is ground,

$f(x, f(a, x))$ is not linear but

$f(x, f(y, z))$ is linear

What about the following terms ?

$f(a, a, a)$ is ill-formed (since f is of arity 2)

a is correct

$x(a)$ is ill-formed (since all variables are assumed of arity 0)

f is ill-formed (since f is of arity 2)

Subterms

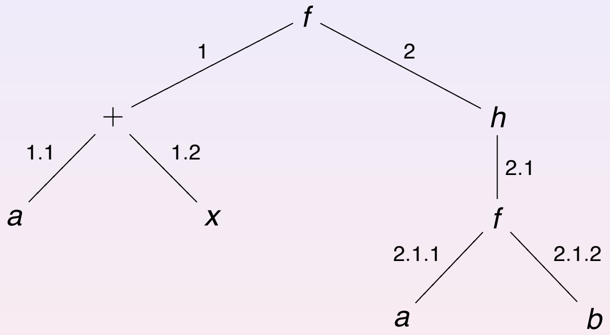
$t[s]_\omega$ denotes the term t with s as **subterm** at **position** (or **occurrence**) ω.

$t|_\omega$ denotes the subterm at occurrence ω.

$$f(a+x, h(f(a,b)))|_2 = h(f(a,b))$$

Terms as trees

$t = f(a + x, h(f(a, b)))$ is represented by :



$|t|$ is the size of t i.e. the cardinality of $\mathcal{D}om(t)$.

$$|f(a + x, h(f(a, b)))| = 8$$

$\mathcal{V}ar(t)$ denotes the set of variables in t.

$$\mathcal{V}ar(f(a + x, h(f(a, b)))) = \{x\}$$

Simple questions—

What is $f(f(a,b),g(a))|_{1.1}$?

Simple questions —

What is $f(f(a,b),g(a))|_{1.1}$? — a

What is $f(f(a,b),g(a))|_{\Lambda}$?

Simple questions—

What is $f(f(a,b),g(a))|_{1.1}$? — a

What is $f(f(a,b),g(a))|_{\Lambda}$? — $f(f(a,b),g(a))$

What is $f(f(a,b),g(a))|_{1.2}$?

Simple questions —

What is $f(f(a,b),g(a))|_{1.1}$? — a

What is $f(f(a,b),g(a))|_{\Lambda}$? — $f(f(a,b),g(a))$

What is $f(f(a,b),g(a))|_{1.2}$? — b

What is the arity of f just above ?

Simple questions —

What is $f(f(a,b),g(a))|_{1.1}$? — a

What is $f(f(a,b),g(a))|_{\Lambda}$? — $f(f(a,b),g(a))$

What is $f(f(a,b),g(a))|_{1.2}$? — b

What is the arity of f just above ? — 2

What is the arity of a just above ?

Simple questions—

What is $f(f(a,b),g(a))|_{1.1}$? — a

What is $f(f(a,b),g(a))|_{\Lambda}$? — $f(f(a,b),g(a))$

What is $f(f(a,b),g(a))|_{1.2}$? — b

What is the arity of f just above ? — 2

What is the arity of a just above ? — 0

What are the variables of $f(f(a,b),g(a))|_{1.2}$?

Simple questions —

What is $f(f(a,b),g(a))|_{1.1}$? — a

What is $f(f(a,b),g(a))|_{\Lambda}$? — $f(f(a,b),g(a))$

What is $f(f(a,b),g(a))|_{1.2}$? — b

What is the arity of f just above ? — 2

What is the arity of a just above ? — 0

What are the variables of $f(f(a,b),g(a))|_{1.2}$? — $\emptyset$

What are the variables of $f(f(x,x),g(a))|_{1.2}$?

Simple questions —

What is $f(f(a,b),g(a))|_{1.1}$? — a

What is $f(f(a,b),g(a))|_{\Lambda}$? — $f(f(a,b),g(a))$

What is $f(f(a,b),g(a))|_{1.2}$? — b

What is the arity of f just above ? — 2

What is the arity of a just above ? — 0

What are the variables of $f(f(a,b),g(a))|_{1.2}$? — $\emptyset$

What are the variables of $f(f(x,x),g(a))|_{1.2}$? — $\{x\}$

What are the variables of $f(f(x,x),g(a))$?

Simple questions—

What is $f(f(a,b),g(a))|_{1.1}$? — a

What is $f(f(a,b),g(a))|_{\Lambda}$? — $f(f(a,b),g(a))$

What is $f(f(a,b),g(a))|_{1.2}$? — b

What is the arity of f just above ? — 2

What is the arity of a just above ? — 0

What are the variables of $f(f(a,b),g(a))|_{1.2}$? — $\emptyset$

What are the variables of $f(f(x,x),g(a))|_{1.2}$? — $\{x\}$

What are the variables of $f(f(x,x),g(a))$? — $\{x\}$

Substitutions

Substitution

A substitution σ is a mapping from the set ot variables to the set of terms :

$$\sigma : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F}, \mathcal{X})$$

It is extended as a morphism from terms to terms :

$$\sigma : \mathcal{T}(\mathcal{F}, \mathcal{X}) \mapsto \mathcal{T}(\mathcal{F}, \mathcal{X})$$

$$\sigma(f(t_1, t_2)) = f(\sigma(t_1), \sigma(t_2))$$

If $\sigma = \{x \mapsto a, y \mapsto f(a, g(z)), z \mapsto g(z)\}$, then $\sigma(f(x, f(a, z))) = f(a, f(a, g(z)))$.

Matching

Matching

Finding a substitution σ such that

$$\sigma(l) = t$$

is called the matching problem from l to t.

This is denoted $l \ll^? t$

It is decidable in linear time in the size of t.

It induces a relation on terms called subsumption

Matching : A rule based description

Delete		$t \ll^? t \ \wedge\ P$	
	$\rightarrow$	P	
Decomposition		$f(t_1, \ldots, t_n) \ll^? f(t'_1, \ldots, t'_n) \ \wedge\ P$	
	$\rightarrow$	$\bigwedge_{i=1,\ldots,n} t_i \ll^? t'_i \ \wedge\ P$	
SymbolClash		$f(t_1, \ldots, t_n) \ll^? g(t'_1, \ldots, t'_m) \ \wedge\ P$	
	$\rightarrow$	*Fail*	if $f \neq g$
SymbolVariableClash		$f(t_1, \ldots, t_n) \ll^? x \ \wedge\ P$	
	$\rightarrow$	*Fail*	if $x \in \mathcal{X}$

Matching : A rule based description

Delete		$t \ll^? t \;\wedge\; P$	
	$\to$	P	
Decomposition		$f(t_1, \ldots, t_n) \ll^? f(t'_1, \ldots, t'_n) \;\wedge\; P$	
	$\to$	$\bigwedge_{i=1,\ldots,n} t_i \ll^? t'_i \;\wedge\; P$	
SymbolClash		$f(t_1, \ldots, t_n) \ll^? g(t'_1, \ldots, t'_m) \;\wedge\; P$	
	$\to$	*Fail*	if $f \neq g$
SymbolVariableClash		$f(t_1, \ldots, t_n) \ll^? x \;\wedge\; P$	
	$\to$	*Fail*	if $x \in \mathcal{X}$
MergingClash		$x \ll^? t \;\wedge\; x \ll^? t' \;\wedge\; P$	
	$\to$	*Fail*	if $t \neq t'$

Find a match

$$x+(y*3) \ll^? 1+(4*3)$$

$$\Rightarrow_{\text{Decomposition}}\ x \ll^? 1 \ \wedge\ y*3 \ll^? 4*3$$

$$\Rightarrow_{\text{Decomposition}}\ x \ll^? 1 \ \wedge\ y \ll^? 4 \ \wedge\ 3 \ll^? 3$$

$$\Rightarrow_{\text{Delete}}\ x \ll^? 1 \ \wedge\ y \ll^? 4$$

Find a match

$$x+(y*3) \ll^? 1+(4*3)$$

$$\Rightarrow_{\text{Decomposition}}\ x \ll^? 1 \ \wedge\ y*3 \ll^? 4*3$$

$$\Rightarrow_{\text{Decomposition}}\ x \ll^? 1 \ \wedge\ y \ll^? 4 \ \wedge\ 3 \ll^? 3$$

$$\Rightarrow_{\text{Delete}}\ x \ll^? 1 \ \wedge\ y \ll^? 4$$

$$x+(y*y) \ll^? 1+(4*3)$$

$$\Rightarrow_{\text{Decomposition}}\ x \ll^? 1 \ \wedge\ y*y \ll^? 4*3$$

$$\Rightarrow_{\text{Decomposition}}\ x \ll^? 1 \ \wedge\ y \ll^? 4 \ \wedge\ y \ll^? 3$$

$$\Rightarrow_{\text{MergingClash}}\ \mathit{Fail}$$

Matching rules

Does it terminate ?
Do we always get the same result ?

Matching rules

Does it terminate ?
Do we always get the same result ?

Theorem The normal form by the rules in *Match*, of any matching problem $t \ll^? t'$ such that $\mathcal{V}ar(t) \cap \mathcal{V}ar(t') = \emptyset$, exists and is unique.

1. If it is *Fail*, then there is no match from t to t'.
2. If it is of the form $\bigwedge_{i \in I} x_i \ll^? t_i$ with $I \neq \emptyset$, the substitution $\sigma = \{x_i \mapsto t_i\}_{i \in I}$ is the unique match from t to t'.
3. If it is empty then t and t' are identical : $t = t'$.

Matching is used everywhere

ML
TOM
XQUERY
“pattern matching” in general
. . .

Matching is used everywhere

ML
TOM
XQUERY
“pattern matching” in general
...

CyberSitter censors "menu */ #define" because of the string "nu...de".

From Internet Risks Forum NewsGroup (RISKS), vol. 19, issue 56.

Term subsumption

$$s \ll t \iff \sigma(s) = t$$

Vocabulary :
t is called an instance of s
s is said more general than t or
s subsumes t
σ is a match from s to t.
$\ll$ is a quasi-ordering on terms called subsumption.

$$f(x, y) \ll f(f(a, b), h(y))$$

Theorem : [Huet78]
Up to renaming, the subsumption ordering on terms is well-founded.

Notice that

$s \leq t \quad \not\Rightarrow \quad f(u,s) \leq f(u,t)$
since
$x \leq a$ but $f(x,x) \not\leq f(x,a)$

$s \leq t \quad \not\Rightarrow \quad \sigma(s) \leq \sigma(t)$
since
$x \leq a$ but $(x \mapsto b)x \not\leq (x \mapsto b)a$

Rewriting

Definition of rewriting

It relies on 5 notions :

- The objects : terms and rewrite rules
- The actions
  - matching
  - substitutions
  - replacement

and, given a rule and a term, it consists in :

- finding a subterm of the term
- that matches the left hand side of the rule
- and replacing that subterm by the right hand side of the rule instanciated by the match

Formally

t rewrites to t' using the rule $l \rightarrow r$ if

$$t_{|p} = \sigma(l) \quad \text{and} \quad t' = t[\sigma(r)]_p$$

This is denoted

$$t \longrightarrow_{p}^{l \rightarrow r} t'$$

Rewrite relation

A term rewrite system R (a set of rewrite rules) determines a relation on terms denoted $\longrightarrow_R$:

Rewrite relation

A term rewrite system R (a set of rewrite rules) determines a relation on terms denoted $\longrightarrow_R$:

$$\boxed{u \rightarrow_R v}$$

iff
there exist t, $l \rightarrow r \in R$, an occurrence ω in t, such that
$\boxed{u = t[\sigma(l)]_\omega}$
and
$\boxed{v = t[\sigma(r)]_\omega}$

Rewrite relation

A term rewrite system R (a set of rewrite rules) determines a relation on terms denoted $\longrightarrow_R$:

$$u \rightarrow_R v$$

iff

there exist t, $l \rightarrow r \in R$, an occurrence ω in t, such that

$u = t[\sigma(l)]_\omega$

and

$v = t[\sigma(r)]_\omega$

$$t[\sigma(l)]_\omega \rightarrow_R t[\sigma(r)]_\omega$$

Rewrite relation

A term rewrite system R (a set of rewrite rules) determines a relation on terms denoted $\longrightarrow_R$:

$$\boxed{u \rightarrow_R v}$$

iff
there exist t, $l \rightarrow r \in R$, an occurrence ω in t, such that
$\boxed{u = t[\sigma(l)]_\omega}$
and
$\boxed{v = t[\sigma(r)]_\omega}$

$$\boxed{t[\sigma(l)]_\omega \rightarrow_R t[\sigma(r)]_\omega}$$

USUALLY, when defining the rewriting relation, one requires the all rewrite rules satisfy $Var(r) \subseteq Var(l)$.

Simple examples —

Consider the rewrite system R :

$$\begin{array}{lcl} x + x & \rightarrow & x \\ (a + x) + y & \rightarrow & y + x \end{array}$$

How many redexes are in $(a + a) + (a + a)$?

Simple examples —

Consider the rewrite system R :

$$\begin{array}{lcl} x + x & \twoheadrightarrow & x \\ (a + x) + y & \twoheadrightarrow & y + x \end{array}$$

How many redexes are in $(a + a) + (a + a)$? — 4

Draw the rewrite derivation tree issued from $(a + a) + (a + a)$.

Simple examples —

Consider the rewrite system R :

$$\begin{array}{lcl} x + x & \rightarrow & x \\ (a + x) + y & \rightarrow & y + x \end{array}$$

How many redexes are in $(a + a) + (a + a)$? — 4

Draw the rewrite derivation tree issued from $(a + a) + (a + a)$.

Is $((a + a) + (a + a), a)$ in the transitive closure of $\rightarrow$?

Simple examples —

Consider the rewrite system R :

$$\begin{array}{lcl} x + x & \rightharpoonup & x \\ (a + x) + y & \rightharpoonup & y + x \end{array}$$

How many redexes are in $(a + a) + (a + a)$? — 4

Draw the rewrite derivation tree issued from $(a + a) + (a + a)$.

Is $((a + a) + (a + a), a)$ in the transitive closure of $\rightharpoonup$? — yes

Is (a, a) in the transitive closure of $\rightharpoonup$?

Simple examples —

Consider the rewrite system R :

$$\begin{array}{lcl} x + x & \rightarrow & x \\ (a + x) + y & \rightarrow & y + x \end{array}$$

How many redexes are in $(a + a) + (a + a)$? — 4

Draw the rewrite derivation tree issued from $(a + a) + (a + a)$.

Is $((a + a) + (a + a), a)$ in the transitive closure of $\rightarrow$? — yes

Is (a, a) in the transitive closure of $\rightarrow$? — no

Is (a, a) in the reflexive closure of $\rightarrow$?

Simple examples —

Consider the rewrite system R :

$$\begin{array}{lcl} x + x & \rightarrow & x \\ (a + x) + y & \rightarrow & y + x \end{array}$$

How many redexes are in $(a + a) + (a + a)$? — 4

Draw the rewrite derivation tree issued from $(a + a) + (a + a)$.

Is $((a + a) + (a + a), a)$ in the transitive closure of $\rightarrow$? — yes

Is (a, a) in the transitive closure of $\rightarrow$? — no

Is (a, a) in the reflexive closure of $\rightarrow$? — yes

Is there any infinite derivation starting from a finite tree using R ?

Simple examples —

Consider the rewrite system R :

$$\begin{array}{lcl} x + x & \rightarrow & x \\ (a + x) + y & \rightarrow & y + x \end{array}$$

How many redexes are in $(a + a) + (a + a)$? — 4

Draw the rewrite derivation tree issued from $(a + a) + (a + a)$.

Is $((a + a) + (a + a), a)$ in the transitive closure of $\rightarrow$? — yes

Is (a, a) in the transitive closure of $\rightarrow$? — no

Is (a, a) in the reflexive closure of $\rightarrow$? — yes

Is there any infinite derivation starting from a finite tree using R ? — no

Why ?

Expressiveness of rewriting

[Max Dauchet 1989]
A Turing machine can be simulated by a single rewrite rule
This unique rewrite rule can further be left linear and regular !
... Termination of a rewrite relation

On the use of term rewriting

- for programming (ASF, ELAN, MAUDE, ML, OBJ, Stratego, . . .)
- for proving (Completion procedures, proof systems, . . .)
- for solving (Constraint manipulations, . . .)
- for verifying (Exhaustive (and may be intelligent) search)

What are the typical problems of the field ?

Confluence
Termination
Control of rewriting : strategies
Conditional rewriting
Theorem proving and rewriting
Rewriting and higher-order features : ρ-calculus
Types and rewriting

Extended notions of rewriting

Conditional rules

$$l \rightarrow r \ \ \textbf{if} \ \ c$$

- $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$,
- c a boolean term
- $\mathcal{V}ar(r) \cup \mathcal{V}ar(c) \subseteq \mathcal{V}ar(l)$

The rule applies on a term t provided the matching substitution σ allows $c\sigma$ to reduce to *true*.

Applying a conditional rewrite rule

$$\begin{array}{rcll} even(0) & \rightarrowtriangle & true & \\ even(s(x)) & \rightarrowtriangle & odd(x) & \\ odd(x) & \rightarrowtriangle & true & \textbf{if} \quad not(even(x)) \\ odd(x) & \rightarrowtriangle & false & \textbf{if} \quad even(x) \end{array}$$

$$even(s(0)) \longrightarrow odd(0) \longrightarrow false$$

Generalized rules

$$l \twoheadrightarrow r \textbf{ where } p_1 := c_1 \ldots \textbf{ where } p_n := c_n$$

- $l, r, p_1, \ldots, p_n, c_1, \ldots, c_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$,
- $\mathcal{V}ar(p_i) \cap (\mathcal{V}ar(l) \cup \mathcal{V}ar(p_1) \cup \cdots \cup \mathcal{V}ar(p_{i-1})) = \emptyset$,
- $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l) \cup \mathcal{V}ar(p_1) \cup \cdots \cup \mathcal{V}ar(p_n)$
- $\mathcal{V}ar(c_i) \subseteq \mathcal{V}ar(l) \cup \mathcal{V}ar(p_1) \cup \cdots \cup \mathcal{V}ar(p_{i-1})$.

where *true* $:= c$ is equivalently written **if** c.
p_i is oftern reduced to a variable x.

Generalized rule application

$$l \twoheadrightarrow r \textbf{ where } p_1 := c_1 \ldots \textbf{ where } p_n := c_n$$

To apply this rewrite rule on t, the matching substitution σ from l to t (i.e. such that $l\sigma = t$) is successively composed with each match μ_i from p_i to $c_i\sigma\mu_1 \ldots \mu_{i-1}$, for all $i = 1, \ldots, n$.

$$move(S) \twoheadrightarrow C(x, y) \;\; \textbf{where} \;\; < x, y >:= position(S) \;\; \textbf{if } x = y$$

Think abstractly

The properties of this relation could be studied in an abstract way :
$\Rightarrow$ Abstract rewrite systems

Abstract rewrite systems

- **Consider a set $\mathcal{T}$**
- **Consider a binary relation $\longrightarrow$ on $\mathcal{T}$ (one-step reduction)**
  - $a \longrightarrow b$: b is the reduct of a
- **Induced relations**
  - transitive closure : $\xrightarrow{+}$
  - transitive reflexive closure : $\xrightarrow{*}$
  - symetric closure : $\longleftrightarrow$

Normalization

Consider an ARS $(\mathcal{T},\rightarrow)$

- An element $t \in \mathcal{T}$ is a **$\rightarrow$-normal form** if there exists no $t' \in \mathcal{T}$ such that $t \rightarrow t'$.

Normalization

Consider an ARS $(\mathcal{T},\rightarrow)$

- An element $t \in \mathcal{T}$ is a **$\rightarrow$-normal form** if there exists no $t' \in \mathcal{T}$ such that $t \rightarrow t'$.

- The relation $\rightarrow$ is **terminating** (or **strongly normalizing**, or **noetherian**) if every reduction sequence is finite.

Normalization

Consider an ARS $(\mathcal{T},\rightarrow)$

- An element $t \in \mathcal{T}$ is a **$\rightarrow$-normal form** if there exists no $t' \in \mathcal{T}$ such that $t \rightarrow t'$.

- The relation $\rightarrow$ is **terminating** (or **strongly normalizing**, or **noetherian**) if every reduction sequence is finite.

Normalization

Consider an ARS ($\mathcal{T}$,$\rightarrow$)

- An element $t \in \mathcal{T}$ is a **$\rightarrow$-normal form** if there exists no $t' \in \mathcal{T}$ such that $t \rightarrow t'$.
- The relation $\rightarrow$ is **terminating** (or **strongly normalizing**, or **noetherian**) if every reduction sequence is finite.
  .
  $a \rightarrow a$ is not terminating
- The relation $\rightarrow$ is **weakly normalizing** (or weakly terminating) if every element $t \in \mathcal{T}$ has a normal form.

Normalization

Consider an ARS $(\mathcal{T},\to)$

- An element $t \in \mathcal{T}$ is a **$\to$-normal form** if there exists no $t' \in \mathcal{T}$ such that $t \to t'$.
- The relation $\to$ is **terminating** (or **strongly normalizing**, or **noetherian**) if every reduction sequence is finite.
  .
  $a \to a$ is not terminating
- The relation $\to$ is **weakly normalizing** (or weakly terminating) if every element $t \in \mathcal{T}$ has a normal form.

Normalization

Consider an ARS ($\mathcal{T}$,$\rightarrow$)

- An element $t \in \mathcal{T}$ is a **$\rightarrow$-normal form** if there exists no $t' \in \mathcal{T}$ such that $t \rightarrow t'$.
- The relation $\rightarrow$ is **terminating** (or **strongly normalizing**, or **noetherian**) if every reduction sequence is finite.
  . $a \rightarrow a$ is not terminating
- The relation $\rightarrow$ is **weakly normalizing** (or weakly terminating) if every element $t \in \mathcal{T}$ has a normal form.
  . $a \rightarrow a \quad a \rightarrow b$ is weakly terminating
- The relation $\rightarrow$ has the **unique normal form property** if for any $t, t' \in \mathcal{T}$, $t \xleftrightarrow{*} t'$ and t, t' are normal forms imply $t = t'$.

Showing normalization

A (partial) order on $\mathcal{T}$ is a reflexive, antisymetric and transitive relation.

An ordering is **total** on $\mathcal{T}$ when two terms are always comparable

$>$ is well-founded or Noetherian on $\mathcal{T}$ if there is no infinite decreasing sequence on $\mathcal{T}$:

$$t_1 > t_2 > t_3 > \dots$$

Theorem

Consider an ARS $(\mathcal{A}, \rightarrow)$.

$\rightarrow$ is terminating

iff

there exists a well-founded (partial) order $>$ on $\mathcal{T}$ and a mapping ϕ s.t.

for all rewrite rule $a \rightarrow a'$ implies $\phi(a) > \phi(a')$.

Example

Use the order $(>, \mathbb{N})$ which is well-founded.

Several choices for strings $\mathcal{A} = (\bullet \mid \circ)^*$

- $\phi(w)$ = number of $\bullet$
  works for all $\bullet$-decreasing reductions
- $\phi(w)$ = number of $\circ$
  works for all $\circ$-decreasing reductions

Example

Use the order $(>, \mathbb{N})$ which is well-founded.

Several choices for strings $\mathcal{A} = (\bullet \mid \circ)^*$

- $\phi(w)$ = number of $\bullet$
  works for all $\bullet$-decreasing reductions
- $\phi(w)$ = number of $\circ$
  works for all $\circ$-decreasing reductions

$$\bullet\bullet \rightarrow \circ$$
$$\circ\circ \rightarrow \circ$$
$$\bullet\circ \rightarrow \bullet$$
$$\circ\bullet \rightarrow \bullet$$

Example

Use the order $(>, \mathbb{N})$ which is well-founded.

Several choices for strings $\mathcal{A} = (\bullet \mid \circ)^*$

- $\phi(w)$ = number of $\bullet$
  works for all $\bullet$-decreasing reductions
- $\phi(w)$ = number of $\circ$
  works for all $\circ$-decreasing reductions

$$\begin{aligned}
\bullet\bullet &\rightarrow \circ \\
\circ\circ &\rightarrow \circ \\
\bullet\circ &\rightarrow \bullet \\
\circ\bullet &\rightarrow \bullet
\end{aligned}$$

- $\phi(w)$ = number of $\bullet$ and $\circ$
  works for all length-decreasing reductions

Definitions (Relathionships)

Localy confluent (LC)

$$\begin{tikzcd}
 & t_0 \arrow[dl] \arrow[dr] & \\
t_1 \arrow[dr, dotted, "*"'] & & t_2 \arrow[dl, dotted, "*"] \\
 & t_3 &
\end{tikzcd}$$

Diamond property (DP)

$$\begin{tikzcd}
 & t_0 \arrow[dl] \arrow[dr] & \\
t_1 \arrow[dr, dotted] & & t_2 \arrow[dl, dotted] \\
 & t_3 &
\end{tikzcd}$$

Church Rosser (CR)

$$\begin{tikzcd}
t \arrow[rr, leftrightarrow, "+"] \arrow[dr, dotted, "*"'] & & s \arrow[dl, dotted, "*"] \\
 & u &
\end{tikzcd}$$

Confluent (C)

$$\begin{tikzcd}
 & t_0 \arrow[dl, "*"'] \arrow[dr, "*"] & \\
t_1 \arrow[dr, dotted, "*"'] & & t_2 \arrow[dl, dotted, "*"] \\
 & t_3 &
\end{tikzcd}$$

Local versus global confluence

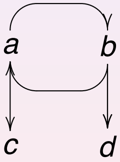
1. $C \Rightarrow LC$
2. $LC \Rightarrow C?$

Local versus global confluence

1. $C \Rightarrow LC$
2. $LC \Rightarrow C?$

➥ Consider four distinct elements a, b, c, d of $\mathcal{T}$ and the relation :

$$a \longrightarrow b$$
$$b \longrightarrow a$$
$$a \longrightarrow c$$
$$b \longrightarrow d$$



Newman's lemma

[Newman 1942]

Provided the relation $\longrightarrow$ is terminating

then

$\longrightarrow$ is confluent iff it is locally confluent

Proof :

Newman's lemma

[Newman 1942]

Provided the relation $\longrightarrow$ is terminating

then

$\longrightarrow$ is confluent iff it is locally confluent

Proof :

- locally confluent if confluent
  ➥ obvious

Newman's lemma

[Newman 1942]

Provided the relation $\longrightarrow$ is terminating

then

$\longrightarrow$ is confluent iff it is locally confluent

Proof :

- locally confluent if confluent
  ➥ obvious
- confluent if locally confluent
  ➥ ?

Noetherian induction : a fondamental tool

Let $(\mathcal{T}, >)$ be an ordered set s.t. $>$ is well-founded.

Let $\mathcal{P}$ be a proposition :

1. $\forall t \in \mathcal{T},\ [\forall t' \in \{t' \mid t > t'\},\ \mathcal{P}(t')] \Rightarrow \mathcal{P}(t)$
2. $\mathcal{P}(t)$ is provable for all minimal element t,

then $\forall t \in \mathcal{T}, \mathcal{P}(t)$.

Noetherian induction : a fondamental tool

Consider $(\mathcal{T}, \rightarrow)$

$$
\begin{tikzcd}
 & & t \arrow[dl] \arrow[dr] & & \\
 & t_1' \arrow[dl, "m-1"'] \arrow[dr, "*"] & & t_2' \arrow[dl, "*"'] \arrow[dr, "n-1"] & \\
t_1 \arrow[dr, "*"] & & t' \arrow[dl, "*"'] & & t_2 \arrow[ddll, "*"'] \\
 & t_1'' \arrow[dr, "*"] & & & \\
 & & u & &
\end{tikzcd}
$$

How to build well founded orderings ?

Termination

R (or $\rightarrow_R$) terminates
iff all derivation issued from any term terminates.

Termination implies the existence of normal form(s) for any term.

Termination is in general undecidable
but interesting sufficient condition can be found.

Proving termination could be tricky . . .

Proving termination could be tricky . . .

$$f(a, b, x) \rightarrow f(x, x, x)$$

is terminating

Proving termination could be tricky . . .

$$f(a, b, x) \rightarrow f(x, x, x)$$

is terminating

$$\begin{array}{lcl} g(x, y) & \rightarrow & x \\ g(x, y) & \rightarrow & y, \end{array}$$

is terminating

Proving termination could be tricky . . .

$$f(a, b, x) \rightarrow f(x, x, x)$$

is terminating

$$\begin{array}{rcl} g(x, y) & \rightarrow & x \\ g(x, y) & \rightarrow & y, \end{array}$$

is terminating

Is the union terminating ?

$$\begin{array}{rcl} f(a,b,x) & \rightarrow & f(x,x,x) \\ g(x,y) & \rightarrow & x \\ g(x,y) & \rightarrow & y, \end{array}$$

$$
\begin{array}{rcl}
f(a,b,x) & \rightarrow & f(x,x,x) \\
g(x,y) & \rightarrow & x \\
g(x,y) & \rightarrow & y,
\end{array}
$$

We have the derivation :

$$
f(g(a,b),g(a,b),g(a,b)) \longrightarrow f(a,g(a,b),g(a,b)) \longrightarrow f(a,b,g(a,b))
$$

[Toyama 1986]

Termination

- ensures finiteness of computations

Termination

- ensures finiteness of computations
- is a necessary condition for deciding of other properties (non ambiguity, reachability tests, ...)

Termination

- ensures finiteness of computations
- is a necessary condition for deciding of other properties (non ambiguity, reachability tests, . . .)
- is undecidable.

Proving Termination

Termination of rewriting can be checked by sufficient conditions :

- Syntactic and semantic methods (applying directly to the TRS) KBO [Knuth & Bendix 1970], LPO [Kamin & Levy 1980], RPO [Dershowitz 1982], RPOS [Steinbach 1989], GPO [Dershowitz & Hoot 1995], Polynomial interpretations [Lankford 1975, Ben Cherifa & Lescanne 1986],. . .
- Transformational approaches (transforming one TRS into another) Semantic labelling [Zantema 1995], Dependency pairs [Arts & Giesl 1996], . . .
- Induction on the derivation trees (schematization by abstraction and narrowing of the derivations) [Fissore & Gnaedig & Kirchner 2003]

Termination

Orderings on terms

A **Reduction ordering** is an ordering on $\mathcal{T}$,
stable by context and substitution :
➥ for every context $C[\_]$ and for all substitutions σ,
if $t > s$ then $C[t] > C[s]$ and $\sigma(t) > \sigma(s)$.

Theorem R terminates iff there exists a well-founded reduction ordering $>$ s.t. for all rewrite rule $(l \rightarrow r) \in R$, $l > r$.

Example

The rules of the game :

$$\bullet\bullet \rightarrow \circ$$
$$\circ\circ \rightarrow \circ$$
$$\bullet\circ \rightarrow \bullet$$
$$\circ\bullet \rightarrow \bullet$$

$l > r$ if $|l| > |r|$

Example

The rules of the game :

$$\bullet\bullet \rightarrow \circ$$
$$\circ\circ \rightarrow \circ$$
$$\bullet\circ \rightarrow \bullet$$
$$\circ\bullet \rightarrow \bullet$$

$l > r$ if $|l| > |r|$

$|f(f(x,x),y)| > |f(y,y)|$
but
$|f(f(x,x),f(x,x))| \not> |g(g(x,x),g(x,x))|$

Example modified

The rules of the game slightly change :

$$\bullet\bullet \rightarrow \circ\circ$$
$$\circ\circ \rightarrow \circ$$
$$\bullet\circ \rightarrow \bullet$$
$$\circ\bullet \rightarrow \bullet$$

Example modified

The rules of the game slightly change :

$$\bullet\bullet \rightarrow \circ\circ$$
$$\circ\circ \rightarrow \circ$$
$$\bullet\circ \rightarrow \bullet$$
$$\circ\bullet \rightarrow \bullet$$

$l > r$ if $|l|_{\bullet\circ} > |r|_{\bullet\circ}$
($|t|_{\bullet\circ}$=number of $\bullet$ and $\circ$ of the term t)

$$|\bullet\bullet|_{\bullet\circ} = 2 \not> 2 = |\circ\circ|_{\bullet\circ}$$

Example

The rules of the game :

$$
\begin{aligned}
\bullet\bullet &\rightarrow \circ\circ \\
\circ\circ &\rightarrow \bullet \\
\bullet\circ &\rightarrow \bullet \\
\circ\bullet &\rightarrow \bullet
\end{aligned}
$$

$l > r$ if $|l|_{\bullet\circ+\bullet} > |r|_{\bullet\circ+\bullet}$

$$|\circ\,\circ|_{\bullet\circ+\bullet} = 2 \not> 2 = |\bullet|_{\bullet\circ+\bullet}$$

Extensions of reduction ordering

Lexicographical extensions

Let $>$ be an ordering on $\mathcal{T}$.
Its **lexicographical extension** $>^{lex}$ on $\mathcal{T}^n$ is defined as :

$$(s_1, \ldots, s_n) >^{lex} (t_1, \ldots, t_n)$$

if there exists i, $1 \leq i \leq n$ s.t. $s_i >_i t_i$, and $\forall j, 1 \leq j < i$, $s_j = t_j$.

If $>$ is well-founded on $\mathcal{T}$, then $>^{lex}$ is well-founded on $\mathcal{T}^n$.

Lexicographical extensions

Let $>$ be an ordering on $\mathcal{T}$.
Its **lexicographical extension** $>^{lex}$ on $\mathcal{T}^n$ is defined as :

$$(s_1, \ldots, s_n) >^{lex} (t_1, \ldots, t_n)$$

if there exists i, $1 \leq i \leq n$ s.t. $s_i >_i t_i$, and $\forall j, 1 \leq j < i, s_j = t_j$.

If $>$ is well-founded on $\mathcal{T}$, then $>^{lex}$ is well-founded on $\mathcal{T}^n$.

FALSE for an infinite product of ordered sets :
$\mathcal{T} = \{a, b\}$ with $a < b$

$$b >^{lex} ab >^{lex} aab >^{lex} aaab >^{lex} \ldots$$

Multiset extensions

Let $>$ an ordering on $\mathcal{T}$.

Its (strict) multiset extension denoted $>^{mult}$ is defined by :

$$\mathcal{M} = \{s_1, \ldots, s_m\} >^{mult} \mathcal{N} = \{t_1, \ldots, t_n\}$$

if there exist $i \in \{1, \ldots, m\}$ and $1 \leq j_1 < \ldots < j_k \leq n$ with $k \geq 0$, such that :

- $s_i > t_{j_1}, \ldots, s_i > t_{j_k}$ and,
- either $\mathcal{M} - \{s_i\} >^{mult} \mathcal{N} - \{t_{j_1}, \ldots, t_{j_k}\}$ or the multisets $\mathcal{M} - \{s_i\}$ and $\mathcal{N} - \{t_{j_1}, \ldots, t_{j_k}\}$ are equal.

Multiset extensions - Examples

if $>$ is well-founded on $\mathcal{T}$, then$>^{mult}$ is well-founded on $\mathcal{M}\sqcap\updownarrow\sqcup(\mathcal{T})$.

Multiset extensions - Examples

if $>$ is well-founded on $\mathcal{T}$, then$>^{mult}$ is well-founded on $\mathcal{M}\sqcap\updownarrow\sqcup(\mathcal{T})$.

$\{3, 3, 1, 2\} >^{mult} \{3, 1\}$
$\{3, 3, 1, 2\} >^{mult} \{3, 2, 2, 2, 2\}$
$\{3, 3, 1, 2\} >^{mult} \{3, 0\} >^{mult} \{3\} >^{mult} \{\}$.

Syntactic reduction ordering

Lexicographic Path Ordering (LPO)

For a given precedence on $\mathcal{F}$,

$$s = f(s_1, .., s_n) >_{lpo} t = g(t_1, \ldots, t_m)$$

if at least one of the following condition is satisfied :

Lexicographic Path Ordering (LPO)

For a given precedence on $\mathcal{F}$,

$$s = f(s_1, .., s_n) >_{lpo} t = g(t_1, \ldots, t_m)$$

if at least one of the following condition is satisfied :

1. $f = g$ and $(s_1, \ldots, s_n) >_{lpo}^{lex} (t_1, \ldots, t_m)$ and $\forall j \in \{1, \ldots, m\}, s >_{lpo} t_j$

Lexicographic Path Ordering (LPO)

For a given precedence on $\mathcal{F}$,

$$s = f(s_1, .., s_n) >_{lpo} t = g(t_1, \ldots, t_m)$$

if at least one of the following condition is satisfied :

1. $f = g$ and $(s_1, \ldots, s_n) >^{lex}_{lpo} (t_1, \ldots, t_m)$ and $\forall j \in \{1, \ldots, m\}, s >_{lpo} t_j$
2. $f >_{\mathcal{F}} g$ and $\forall j \in \{1, \ldots, m\}, s >_{lpo} t_j$

Lexicographic Path Ordering (LPO)

For a given precedence on $\mathcal{F}$,

$$s = f(s_1, .., s_n) >_{lpo} t = g(t_1, \dots, t_m)$$

if at least one of the following condition is satisfied :

1. $f = g$ and $(s_1, \dots, s_n) >^{lex}_{lpo} (t_1, \dots, t_m)$ and $\forall j \in \{1, \dots, m\}, s >_{lpo} t_j$
2. $f >_{\mathcal{F}} g$ and $\forall j \in \{1, \dots, m\}, s >_{lpo} t_j$
3. $\exists i \in \{1, \dots, n\}$ s.t either $s_i >_{lpo} t$, or $s_i = t$.

Lexicographic Path Ordering (LPO)

For a given precedence on $\mathcal{F}$,

$$s = f(s_1, .., s_n) >_{lpo} t = g(t_1, \dots, t_m)$$

if at least one of the following condition is satisfied :

1. $f = g$ and $(s_1, \dots, s_n) >_{lpo}^{lex} (t_1, \dots, t_m)$ and $\forall j \in \{1, \dots, m\}, s >_{lpo} t_j$
2. $f >_{\mathcal{F}} g$ and $\forall j \in \{1, \dots, m\}, s >_{lpo} t_j$
3. $\exists i \in \{1, \dots, n\}$ s.t either $s_i >_{lpo} t$, or $s_i = t$.

Theorem LPO is a simplification ordering
i.e. a reduction ordering that contains the subterm ordering.

Extension of LPO

The definition of the ordering can be extended to terms with variables by adding the following conditions :

1. two different variables are incomparable,
2. a function symbol and a variable are incomparable.

A typical LPO example

Termination of the Ackermann function :

$$
\begin{array}{rcl}
ack(0, y) & \rightarrow & succ(y) \\
ack(succ(x), 0) & \rightarrow & ack(x, succ(0)) \\
ack(succ(x), succ(y)) & \rightarrow & ack(x, ack(succ(x), y)).
\end{array}
$$

A typical LPO example

Termination of the Ackermann function :

$$\begin{array}{rcl} ack(0,y) & \rightarrow & succ(y) \\ ack(succ(x),0) & \rightarrow & ack(x,succ(0)) \\ ack(succ(x),succ(y)) & \rightarrow & ack(x,ack(succ(x),y)). \end{array}$$

With $ack >_{\mathcal{F}} succ$, we can show that

$$\begin{array}{rcl} ack(0,y) & >_{lpo} & succ(y) \\ ack(succ(x),0) & >_{lpo} & ack(x,succ(0)) \\ ack(succ(x),succ(y)) & >_{lpo} & ack(x,ack(succ(x),y)). \end{array}$$

Multiset Path Ordering (MPO)

For a given precedence on $\mathcal{F}$,
$s = f(s_1, .., s_n) >_{mpo} t = g(t_1, \ldots, t_m)$ if one at least of the following conditions holds :

1. $f = g$ and $\{s_1, \ldots, s_n\} >_{mpo}^{mult} \{t_1, \ldots, t_m\}$
2. $f >_{\mathcal{F}} g$ and $\forall j \in \{1, \ldots, m\}, s >_{mpo} t_j$
3. $\exists i \in \{1, \ldots, n\}$ such that either $s_i >_{mpo} t$ or $s_i \sim t$
   where $\sim$ means equivalent up to permutation of subterms.

An MPO example

Termination of the *max* function :

$$\begin{array}{rcl} max(n,0) & \rightarrow & n \\ max(0,n) & \rightarrow & n \\ max(succ(n), succ(m)) & \rightarrow & succ(max(n,m)) \end{array}$$

An MPO example

Termination of the *max* function :

$$\begin{aligned} max(n,0) &\to n \\ max(0,n) &\to n \\ max(succ(n),succ(m)) &\to succ(max(n,m)) \end{aligned}$$

Precedence $\boxed{?} >_{\mathcal{F}} \boxed{?}$

An MPO example

Termination of the *max* function :

$$
\begin{array}{rcl}
max(n, 0) & \rightarrow & n \\
max(0, n) & \rightarrow & n \\
max(succ(n), succ(m)) & \rightarrow & succ(max(n, m))
\end{array}
$$

Precedence $max >_{\mathcal{F}} succ$

Semantic reduction ordering

Building reduction orderings using interpretations

Consider a homomorphism τ from ground terms to $(\mathcal{A}, >)$ with $>$ a well-founded ordering and let f_τ denote the image of $f \in \mathcal{F}$ using τ ; τ and $>$ are constrained to satisfy the monotonicity condition :

$$\forall a, b \in \mathcal{A}, \forall f \in \mathcal{F}, \; a > b \text{ implies } f_\tau(\ldots, a, \ldots) > f_\tau(\ldots, b, \ldots).$$

Then the ordering $>_\tau$ defined by :

$$\forall s, t \in \mathcal{T}(\mathcal{F}), \; s >_\tau t \text{ if } \tau(s) > \tau(t),$$

is well-founded.

Building reduction orderings using interpretations

Then the ordering $>_\tau$ is extended by defining

$$\forall s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X}),\ s >_\tau t \text{ if } \nu(\tau(s)) > \nu(\tau(t))$$

for all assignment ν of values in $\mathcal{A}$ to variables of $\tau(s)$ and $\tau(t)$.
Because $>$ is assumed to be well-founded, a rewrite system is terminating if one can find $\mathcal{A}, \tau$ and $>$ as defined above.

Example

Is the reduction induced by $i(f(x, y)) \rightarrow f(f(i(x), y), y)$ terminating ?

Example

Is the reduction induced by $i(f(x,y)) \rightarrow f(f(i(x),y),y)$ terminating ?

$$\begin{array}{rcl} \tau(i(x)) & = & \tau(x)^{2} \\ \tau(f(x,y)) & = & \tau(x) + \tau(y) \end{array} \qquad \begin{array}{rcl} \tau(x) & = & x \\ \tau(y) & = & y \end{array}$$

Example

Is the reduction induced by $i(f(x,y)) \rightarrow f(f(i(x),y),y)$ terminating ?

$$\begin{aligned} \tau(i(x)) &= \tau(x)^{2} \\ \tau(f(x,y)) &= \tau(x) + \tau(y) \end{aligned} \qquad \begin{aligned} \tau(x) &= x \\ \tau(y) &= y \end{aligned}$$

Monotonicity : $a > b$ implies $f_\tau(a) > f_\tau(b)$
(each function is increasing on natural numbers)

Example

Is the reduction induced by $i(f(x,y)) \to f(f(i(x),y),y)$ terminating ?

$$\begin{array}{rcl} \tau(i(x)) & = & \tau(x)^{\boxed{2}} \\ \tau(f(x,y)) & = & \tau(x) \boxed{+} \tau(y) \end{array} \qquad \begin{array}{rcl} \tau(x) & = & \boxed{x} \\ \tau(y) & = & \boxed{y} \end{array}$$

Monotonicity : $a > b$ implies $f_\tau(a) > f_\tau(b)$
(each function is increasing on natural numbers)

$$\begin{array}{rcl} \tau(i(f(x,y))) & = & \boxed{(x+y)^2} = \boxed{x^2 + y^2 + 2xy} \\ \tau(f(f(i(x),y),y)) & = & \boxed{x^2 + 2y} \end{array}$$

Example

Is the reduction induced by $i(f(x,y)) \rightarrow f(f(i(x),y),y)$ terminating ?

$$\begin{array}{rcl} \tau(i(x)) & = & \tau(x)^{2} \\ \tau(f(x,y)) & = & \tau(x) + \tau(y) \end{array} \qquad \begin{array}{rcl} \tau(x) & = & x \\ \tau(y) & = & y \end{array}$$

Monotonicity : $a > b$ implies $f_\tau(a) > f_\tau(b)$
(each function is increasing on natural numbers)

$$\begin{array}{rcl} \tau(i(f(x,y))) & = & (x+y)^2 = x^2 + y^2 + 2xy \\ \tau(f(f(i(x),y),y)) & = & x^2 + 2y \end{array}$$

For any assignment of positive natural numbers n and m to the variables x and y : $n^2 + m^2 + 2nm > n^2 + 2m$

Another example

Is the following system terminating ?

$$
\begin{array}{rcl}
\ominus \ominus x & \rightarrow & x \\
\ominus(x \oplus y) & \rightarrow & (\ominus x) \oplus (\ominus y) \\
\ominus(x \otimes y) & \rightarrow & (\ominus x) \otimes (\ominus y) \\
x \otimes (y \oplus z) & \rightarrow & (x \otimes y) \oplus (x \otimes z) \\
(x \oplus y) \otimes z & \rightarrow & (x \otimes z) \oplus (y \otimes z)
\end{array}
$$

Another example

Is the following system terminating ?

$$
\begin{array}{rcl}
\ominus \ominus x & \rightarrow & x \\
\ominus(x \oplus y) & \rightarrow & (\ominus x) \oplus (\ominus y) \\
\ominus(x \otimes y) & \rightarrow & (\ominus x) \otimes (\ominus y) \\
x \otimes (y \oplus z) & \rightarrow & (x \otimes y) \oplus (x \otimes z) \\
(x \oplus y) \otimes z & \rightarrow & (x \otimes z) \oplus (y \otimes z)
\end{array}
$$

Interpretation :

$$
\begin{array}{rcl}
\tau(\ominus x) & = & \boxed{2^{\tau(x)}} \\
\tau(x \oplus y) & = & \boxed{\tau(x) + \tau(y) + 1} \\
\tau(x \otimes y) & = & \boxed{\tau(x)\tau(y)} \\
\tau(c) & = & \boxed{3}
\end{array}
$$

Recursion analysis

Dependency pairs method

Standard approaches compare left- and right-hand sides of rules
Automated techniques often use simplification orders, but fail on

$$\begin{array}{rcl} minus(x,0) & \rightarrow & x \\ minus(s(x),s(y)) & \rightarrow & minus(x,y) \\ div(0,s(y)) & \rightarrow & 0 \\ div(s(x),s(y)) & \rightarrow & s(div(minus(x,y),s(y))) \end{array}$$

Dependency pairs method

Standard approaches compare left- and right-hand sides of rules
Automated techniques often use simplification orders, but fail on

$$\begin{array}{rcl} minus(x,0) & \rightarrow & x \\ minus(s(x),s(y)) & \rightarrow & minus(x,y) \\ div(0,s(y)) & \rightarrow & 0 \\ div(s(x),s(y)) & \rightarrow & s(div(minus(x,y),s(y))) \end{array}$$

$$div(s(x),s(s(x))) \quad \not\succeq \quad s(div(minus(x,s(x)),s(s(x))))$$

The dependency pair approach focusses only on those subterms which are responsible for starting new reductions

Dependency pairs for termination

$$\begin{array}{rcl} minus(x,0) & \rightarrow & x \\ minus(s(x),s(y)) & \rightarrow & minus(x,y) \\ div(0,s(y)) & \rightarrow & 0 \\ div(s(x),s(y)) & \rightarrow & s(div(minus(x,y),s(y))) \end{array}$$

minus and *div* (top of lhs) are called defined functions.
If $f(s_1,\ldots,s_n) \rightarrow C[g(t_1,\ldots,t_m)]$ is a rule and g is defined, then $F(s_1,\ldots,s_n) \rightarrow G(t_1,\ldots,t_m)$ is a dependency pair.

Dependency pairs for termination

$$\begin{array}{rcl} minus(x,0) & \rightarrow & x \\ minus(s(x),s(y)) & \rightarrow & minus(x,y) \\ div(0,s(y)) & \rightarrow & 0 \\ div(s(x),s(y)) & \rightarrow & s(div(minus(x,y),s(y))) \end{array}$$

minus and *div* (top of lhs) are called defined functions.
If $f(s_1,...,s_n) \rightarrow C[g(t_1,...,t_m)]$ is a rule and g is defined, then $F(s_1,...,s_n) \rightarrow G(t_1,...,t_m)$ is a dependency pair.

$$\begin{array}{rcl} M(s(x),s(y)) & \rightarrow & M(x,y) \\ D(s(x),s(y)) & \rightarrow & M(x,y) \\ D(s(x),s(y)) & \rightarrow & D(minus(x,y),s(y)) \end{array}$$

Dependency pairs method

A sequence of dependency pairs $DP(R) = s_1 \rightharpoonup t_1, s_2 \rightharpoonup t_2, s_3 \rightharpoonup t_3,...$ is a dependency chain iff there exists a substitution σ s.t. :

$$t_1\sigma \longrightarrow^* s_2\sigma,\ t_2\sigma \longrightarrow^* s_3\sigma, \dots$$

Theorem : A rewrite system R terminates iff there is no infinite dependency chain.

Dependency Graph :

- Nodes are dependency pairs
- There is an arrow from $s_1 \rightharpoonup t_1$ to $s_2 \rightharpoonup t_2$ if there exists a substitution σ s.t. : $t_1\sigma \longrightarrow^* s_2\sigma$.

Dependency pairs method

$(\geq, >)$ is a reduction pair iff

- $>$ is stable by substitution and well-founded
- $\geq$ is stable by context and by substitution
- $>$ and $\geq$ are compatible : $> \circ \geq \subseteq >$ or $\geq \circ > \subseteq >$.

Theorem : A rewrite system R terminates if for any cycle P in the dependency graph, there exists a reduction pair $(\geq, >)$ such that

- $l \geq r$ for all rules $l \rightarrow r$ in R
- $s > t$ for at least one dependency pair $s \rightarrow t$ in P
- $s' \geq t'$ for all other dependency pairs $s' \rightarrow t'$ in P

Well-founded reduction orderings

- Syntactic
  Based on the precedence concept (i.e. a partiel order $>_{\mathcal{F}}$ on $\mathcal{F}$)
  Example : Recursive or Lexicographic path ordering [Dershowitz, 82]
- Semantic
  Terms are interpreted in another structure where a well-founded ordering is known (e.g. the natural numbers)
  Example : Polynomial interpretations
- Combinations
  Ordering combining semantical and syntactical behavior
- Recursion analysis
  Induction, dependency pairs

How to determine the unicity of the result ?

Back to ARS properties

Consider an ARS ($\mathcal{T}$,$\rightarrow$)

- An element $t \in \mathcal{T}$ is a **$\rightarrow$-normal form** if there exists no $t' \in \mathcal{T}$ such that $t \rightarrow t'$.

Back to ARS properties

Consider an ARS ($\mathcal{T}$,$\rightarrow$)

- An element $t \in \mathcal{T}$ is a **$\rightarrow$-normal form** if there exists no $t' \in \mathcal{T}$ such that $t \rightarrow t'$.
- The relation $\rightarrow$ is **terminating** (or **strongly normalizing**, or **noetherian**) if every reduction sequence is finite.

Back to ARS properties

Consider an ARS ($\mathcal{T}$,$\rightarrow$)

- An element $t \in \mathcal{T}$ is a **$\rightarrow$-normal form** if there exists no $t' \in \mathcal{T}$ such that $t \rightarrow t'$.
- The relation $\rightarrow$ is **terminating** (or **strongly normalizing**, or **noetherian**) if every reduction sequence is finite.
- The relation $\rightarrow$ is **weakly normalizing** (or weakly terminating) if every element $t \in \mathcal{T}$ has a normal form.

Back to ARS properties

Consider an ARS ($\mathcal{T}$,$\rightarrow$)

- An element $t \in \mathcal{T}$ is a **$\rightarrow$-normal form** if there exists no $t' \in \mathcal{T}$ such that $t \rightarrow t'$.
- The relation $\rightarrow$ is **terminating** (or **strongly normalizing**, or **noetherian**) if every reduction sequence is finite.
- The relation $\rightarrow$ is **weakly normalizing** (or weakly terminating) if every element $t \in \mathcal{T}$ has a normal form.
- The relation $\rightarrow$ has the **unique normal form property** if for any $t, t' \in \mathcal{T}$, $t \overset{*}{\longleftrightarrow} t'$ and t, t' are normal forms imply $t = t'$.

Definitions

Localy confluent (LC)

$$\begin{tikzcd} & t_0 \arrow[dl] \arrow[dr] & \\ t_1 \arrow[dr, dotted, "*"'] & & t_2 \arrow[dl, dotted, "*"] \\ & t_3 & \end{tikzcd}$$

Diamond property (DP)

$$\begin{tikzcd} & t_0 \arrow[dl] \arrow[dr] & \\ t_1 \arrow[dr, dotted] & & t_2 \arrow[dl, dotted] \\ & t_3 & \end{tikzcd}$$

Church Rosser (CR)

$$\begin{tikzcd} t \arrow[rr, leftrightarrow, "+"] \arrow[dr, dotted, "*"'] & & s \arrow[dl, dotted, "*"] \\ & u & \end{tikzcd}$$

Confluent (C)

$$\begin{tikzcd} & t_0 \arrow[dl, "*"'] \arrow[dr, "*"] & \\ t_1 \arrow[dr, dotted, "*"'] & & t_2 \arrow[dl, dotted, "*"] \\ & t_3 & \end{tikzcd}$$

Newman's lemma

[Newman 1942]

Provided the relation $\longrightarrow$ is terminating

then

$\longrightarrow$ is confluent iff it is locally confluent

Confluence

Allows us to forget about non-determinism :

Whatever rewriting is done we will converge later.

Back with the simple game

The rules of the game :

$$\bullet\bullet \rightarrow \circ$$
$$\circ\circ \rightarrow \circ$$
$$\bullet\circ \rightarrow \bullet$$
$$\circ\bullet \rightarrow \bullet$$

A starting point :

$$\bullet \circ \bullet \circ \bullet \circ \bullet \bullet \bullet \bullet \circ \circ \bullet \circ \circ \bullet \bullet\circ$$

From a given start, is the result determinist ?

Analysing the different cases

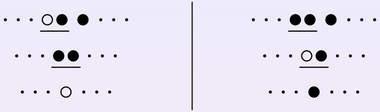
Disjoint redexes :

$$\begin{array}{c}
\cdots \underline{\otimes\otimes} \cdots \otimes \ \otimes \cdots \\
\cdots \otimes \cdots \underline{\otimes\otimes} \cdots \\
\cdots \otimes \cdots \otimes \cdots
\end{array}$$

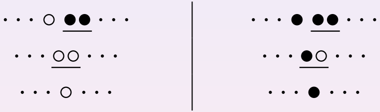
is the same as :

$$\begin{array}{c}
\cdots \otimes \ \otimes \cdots \underline{\otimes\otimes} \cdots \\
\cdots \underline{\otimes\otimes} \cdots \otimes \cdots \\
\cdots \otimes \cdots \otimes \cdots
\end{array}$$

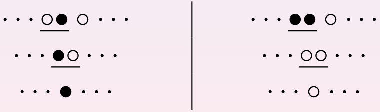
No disjoint redexes (central black) :



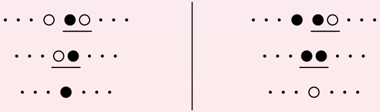
but



or



but



No disjoint redexes (central white) :

$\cdots\underline{\circ\circ}\ \bullet\cdots$	$\cdots\underline{\bullet\circ}\ \bullet\cdots$
$\cdots\underline{\circ\bullet}\cdots$	$\cdots\underline{\bullet\bullet}\cdots$
$\cdots\bullet\cdots$	$\cdots\circ\cdots$

but

$\cdots\circ\ \underline{\circ\bullet}\cdots$	$\cdots\bullet\ \underline{\circ\bullet}\cdots$
$\cdots\underline{\circ\bullet}\cdots$	$\cdots\underline{\bullet\bullet}\cdots$
$\cdots\bullet\cdots$	$\cdots\circ\cdots$

or

$\cdots\underline{\circ\circ}\ \circ\cdots$	$\cdots\underline{\bullet\circ}\ \circ\cdots$
$\cdots\underline{\circ\circ}\cdots$	$\cdots\underline{\bullet\circ}\cdots$
$\cdots\circ\cdots$	$\cdots\bullet\cdots$

but

$\cdots\circ\ \underline{\circ\circ}\cdots$	$\cdots\bullet\ \underline{\circ\circ}\cdots$
$\cdots\underline{\circ\circ}\cdots$	$\cdots\underline{\bullet\circ}\cdots$
$\cdots\circ\cdots$	$\cdots\bullet\cdots$

Thus :
$$\begin{tikzcd} & t_0 \arrow[dl] \arrow[dr] & \\ t_1 \arrow[dr, dashed] & & t_2 \arrow[dl, dashed] \\ & t_3 & \end{tikzcd}$$
but what about :
$$\begin{tikzcd} & t_0 \arrow[dl, "*"'] \arrow[dr, "*"] & \\ t_1 \arrow[dr, dashed, "*"] & & t_2 \arrow[dl, dashed, "*"'] \\ & t_3 & \end{tikzcd}$$

Confluence

➥ Undecidable in general, confluence is decidable for finite and terminating rewrite systems.

➥ Assuming termination of the rewrite relation, its confluence is equivalent to the confluence of critical pairs.

➥ If a rewrite system is orthogonal (linear and non-overlapping), then it is confluent.

Critical pair

A non-variable term t' and a term t overlap if there exists a position ω in t such that $t_{|\omega}$ and t' are unifiable (with $t_{|\omega}$ not a variable).

Critical pair

A non-variable term t' and a term t overlap if there exists a position ω in t such that $t_{|\omega}$ and t' are unifiable (with $t_{|\omega}$ not a variable).

Two terms t and t' are unifiable if there exists a substitution σ such that $\sigma(t) = \sigma(t')$. σ is called a unifier of t and t'.

Parenthesis

Unification problems

Solve an equation

Does it exist x, y, z such that

$$x+(z*y) = y+(x*z)$$

An infinity of solutions,
but a most general one

$$x = y = z$$

Unification problem : a most general unifier of t and t' is a minimal unifier for the subsumption ordering extended to substitutions. It is unique up to renaming.

General Unification Problems

$\mathcal{F}$ a set of function symbols,
$\mathcal{X}$ a set of variables,
$\mathcal{A}$ an $\mathcal{F}$-algebra.
A $<\mathcal{F}, \mathcal{X}, \mathcal{A}>$-unification problem
is a disjunction of existentially quantified formulas

$$P_j = \exists \vec{z} \bigwedge_{i \in I_j} s_i =^?_{\mathcal{A}} t_i$$

sometimes abbreviated

$$P_j = \exists \vec{z} \, \{s_i =^?_{\mathcal{A}} t_i\}_{i \in I_j}.$$

A unifier to such a problem is a *substitution* σ such that
$\exists j, \forall i \in I_j, \quad \mathcal{A} \models \exists \vec{z} \, \sigma_{|\mathcal{X}-\vec{z}}(s_i) = \sigma_{|\mathcal{X}-\vec{z}}(t_i)$.

SYNTACTIC UNIFICATION

Formulas : quantifier free unification problems
Domain : $\mathcal{T}(\mathcal{F}, \mathcal{X})$ (no equational axioms)
Interpretation : trivial one
Solved forms : Tree or dag solved forms

From : J.A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12 :23–41, 1965.

...

5.8 *Unification Algorithm*. The following process, applicable to any finite nonempty set A of well formed expressions, is called the Unification Algorithm :

Step 1. Set $\sigma_0 = \varepsilon$ and $k = 0$, and go to step 2.

Step 2. If $A\sigma_k$ is not a singleton, go to step 3. Otherwise, set $\sigma_A = \sigma_k$ and terminate.

Step 3. Let V_k be the earliest, and U_k the next earliest, in the lexical ordering of the disagreement set B_k of $A\sigma_k$. If V_k is a variable, and does not occur in U_k , set $\sigma_{k+1} = \sigma\{U_k/V_k\}$, add 1 to k, and return to step 2. Otherwise, terminate.

...

Rules for syntactic unification

$$\begin{array}{lcl} \textit{Delete} & & P \wedge s =^? s \\ & \rightarrow & P \end{array}$$

Rules for syntactic unification

Delete $P \wedge s =^? s$
$\rightarrow P$

Decompose $P \wedge f(s_1, \ldots, s_n) =^? f(t_1, \ldots, t_n)$
$\rightarrow P \wedge s_1 =^? t_1 \wedge \ldots \wedge s_n =^? t_n$

Rules for syntactic unification

$$
\begin{array}{lll}
\textit{Delete} & & P \wedge s =^? s \\
& \rightarrow & P \\
\\
\textit{Decompose} & & P \wedge f(s_1, \ldots, s_n) =^? f(t_1, \ldots, t_n) \\
& \rightarrow & P \wedge s_1 =^? t_1 \wedge \ldots \wedge s_n =^? t_n \\
\\
\textit{Conflict} & & P \wedge f(s_1, \ldots, s_n) =^? g(t_1, \ldots, t_p) \\
& \rightarrow & \textit{Fail} \qquad \text{if } f \neq g
\end{array}
$$

Rules for syntactic unification

Delete		$P \wedge s =^? s$	
	$\rightarrow$	P	
Decompose		$P \wedge f(s_1, \ldots, s_n) =^? f(t_1, \ldots, t_n)$	
	$\rightarrow$	$P \wedge s_1 =^? t_1 \wedge \ldots \wedge s_n =^? t_n$	
Conflict		$P \wedge f(s_1, \ldots, s_n) =^? g(t_1, \ldots, t_p)$	
	$\rightarrow$	*Fail*	if $f \neq g$
Coalesce		$P \wedge x =^? y$	
	$\rightarrow$	$\{x \mapsto y\}P \wedge x =^? y$	if $x, y \in \mathcal{V}ar(P) \wedge x \neq$

Rules for syntactic unification

Delete		$P \wedge s =^? s$	
	$\rightarrow$	P	
Decompose		$P \wedge f(s_1, \ldots, s_n) =^? f(t_1, \ldots, t_n)$	
	$\rightarrow$	$P \wedge s_1 =^? t_1 \wedge \ldots \wedge s_n =^? t_n$	
Conflict		$P \wedge f(s_1, \ldots, s_n) =^? g(t_1, \ldots, t_p)$	
	$\rightarrow$	*Fail*	if $f \neq g$
Coalesce		$P \wedge x =^? y$	
	$\rightarrow$	$\{x \mapsto y\}P \wedge x =^? y$	if $x, y \in \mathcal{V}ar(P) \wedge x \neq$

Rules for syntactic unification

$$\begin{array}{lll} \textit{Eliminate} & P \wedge x =^? s & \\ \Vvdash\!\!\twoheadrightarrow & \{x \mapsto s\}P \wedge x =^? s & \text{if } x \notin \mathcal{V}ar(s), s \notin \mathcal{X}, x \in \mathcal{V}ar(P) \end{array}$$

Rules for syntactic unification

$$
\begin{array}{lll}
\textit{Eliminate} & P \wedge x =^? s & \\
\Mapsto & \{x \mapsto s\}P \wedge x =^? s & \text{if } x \notin \mathcal{V}ar(s), s \notin \mathrm{x}, x \in \mathcal{V}ar(P) \\
 & & \\
\textit{Merge} & P \wedge x =^? s \wedge x =^? t & \\
\Mapsto & P \wedge x =^? s \wedge s =^? t & \text{if } 0 < |s| \leq |t|
\end{array}
$$

Rules for syntactic unification

Eliminate $P \wedge x =^? s$

$\mapsto\!\!\!\twoheadrightarrow \{x \mapsto s\}P \wedge x =^? s$ if $x \notin \mathcal{V}ar(s), s \notin \mathrm{x}, x \in \mathcal{V}ar(P)$

Merge $P \wedge x =^? s \wedge x =^? t$

$\mapsto\!\!\!\twoheadrightarrow P \wedge x =^? s \wedge s =^? t$ if $0 < |s| \leq |t|$

Check $P \wedge x =^? s$

$\mapsto\!\!\!\twoheadrightarrow$ *Fail* if $x \in \mathcal{V}ar(s)$ *and* $s \notin \mathrm{x}$

Rules for syntactic unification

Eliminate	$P \wedge x =^? s$	
$\Mapsto$	$\{x \mapsto s\}P \wedge x =^? s$	if $x \notin \mathcal{V}ar(s), s \notin \mathrm{x}, x \in \mathcal{V}ar(P)$
Merge	$P \wedge x =^? s \wedge x =^? t$	
$\Mapsto$	$P \wedge x =^? s \wedge s =^? t$	if $0 < \lvert s \rvert \leq \lvert t \rvert$
Check	$P \wedge x =^? s$	
$\Mapsto$	*Fail*	if $x \in \mathcal{V}ar(s)$ *and* $s \notin \mathrm{x}$
Check∗	$P \wedge \; x_1 =^? s_1[x_2] \wedge \ldots$ $\ldots \wedge x_n =^? s_n[x_1]$	
$\Mapsto$	*Fail*	if $s_i \notin \mathrm{x}$ *for some* $i \in [1..n]$

Solving an equation

$$x + (z * y) = y + (x * z)$$

Solving an equation

$$x + (z * y) = y + (x * z)$$

$$\Rightarrow_{\text{decompose}} x = y \quad \wedge \quad z * y = x * z$$

Solving an equation

$$x+(z*y) = y+(x*z)$$

$$\Rightarrow_{\text{decompose}} x = y \quad \wedge \quad z*y = x*z$$

$$\Rightarrow_{\text{decompose}} x = y \quad \wedge \quad z = x \quad \wedge \quad y = z$$

Solving an equation

$$x+(z*y) = y+(x*z)$$

$$\Rightarrow_{\text{decompose}}\ x = y \ \wedge\ z*y = x*z$$

$$\Rightarrow_{\text{decompose}}\ x = y \ \wedge\ z = x \ \wedge\ y = z$$

$$\Rightarrow_{\text{coalesce}}\ y = z \ \wedge\ x = z \ \wedge\ z = x$$

Solving an equation

$$x+(z*y) = y+(x*z)$$

$$\Rightarrow_{\text{decompose}} x = y \quad \wedge \quad z*y = x*z$$

$$\Rightarrow_{\text{decompose}} x = y \quad \wedge \quad z = x \quad \wedge \quad y = z$$

$$\Rightarrow_{\text{coalesce}} y = z \quad \wedge \quad x = z \quad \wedge \quad z = x$$

$$\Rightarrow_{\text{coalesce}} z = x \quad \wedge \quad y = x \quad \wedge \quad x = x$$

Solving an equation

$$x+(z*y) = y+(x*z)$$

$$\Rightarrow_{\text{decompose}}\ x = y \ \wedge\ z*y = x*z$$

$$\Rightarrow_{\text{decompose}}\ x = y \ \wedge\ z = x \ \wedge\ y = z$$

$$\Rightarrow_{\text{coalesce}}\ y = z \ \wedge\ x = z \ \wedge\ z = x$$

$$\Rightarrow_{\text{coalesce}}\ z = x \ \wedge\ y = x \ \wedge\ x = x$$

$$\Rightarrow_{\text{delete}}\ z = x \ \wedge\ y = x$$

Examples

Examples

$x =^? a$

Examples

$$x =^? a$$
$$x =^? a \wedge y =^? f(x, a)$$

Examples

$x =^? a$

$x =^? a \wedge y =^? f(x, a)$

$f(x, f(x, a)) =^? f(f(a, b), f(u, v))$

Examples

$$x =^? a$$
$$x =^? a \wedge y =^? f(x, a)$$
$$f(x, f(x, a)) =^? f(f(a, b), f(u, v))$$
$$x =^? a \wedge x =^? b$$

Examples

$$x =^? a$$
$$x =^? a \wedge y =^? f(x, a)$$
$$f(x, f(x, a)) =^? f(f(a, b), f(u, v))$$
$$x =^? a \wedge x =^? b$$

Strategy : No

A *tree solved form* for P is any conjonction of equations

$$x_1 =^? t_1 \ \wedge \ \cdots \ \wedge \ x_n =^? t_n$$

equivalent to P such that $\forall i, x_i \in \mathrm{x}$ and :

$$\begin{array}{ll}(i) & \forall 1 \leq i \leq n, x_i \in \mathcal{V}ar(P), \\ (ii) & \forall 1 \leq i, j \leq n, i \neq j \Rightarrow x_i \neq x_j, \\ (iii) & \forall 1 \leq i, j \leq n, x_i \notin \mathcal{V}ar(t_j).\end{array}$$

Example : $x =^? f(f(y)) \ \wedge \ z =^? g(a)$.

Theorem : Starting with a unification problem P and using the above rules repeatedly until none is applicable
— results in *Fail* iff P has no solution, or else it
— results in a tree solved form $x_1 =^? t_1 \wedge \cdots \wedge x_n =^? t_n$ with the same set of solutions than P.
Moreover

$$\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$$

is a most general unifier of P.

Strategy : Never apply eliminate

A dag solved form for a unification problem P is any system of equations :

$$x_1 =^? t_1 \ \wedge \ \cdots \ \wedge \ x_n =^? t_n$$

equivalent to P such that $\forall i, x_i \in$ x and :

$$\begin{array}{ll} (i) & \forall 1 \leq i \leq n, x_i \in \mathcal{V}ar(P), \\ (ii) & \forall 1 \leq i,j \leq n, i \neq j \Rightarrow x_i \neq x_j, \\ (iii) & \forall 1 \leq i \leq j \leq n, x_i \notin \mathcal{V}ar(t_j). \end{array}$$

Example : $x =^? f(u) \ \wedge \ u =^? f(y) \ \wedge \ z =^? g(a)$

Theorem : Starting with a unification problem P and using the above rules except *eliminate* repeatedly until none is applicable,
— results in *Fail* iff P has no solution, or else
— in a dag solved form :

$$x_1 =^? t_1 \ \wedge \ \ldots \ \wedge \ x_n =^? t_n$$

such that $\sigma = \{x_n \mapsto t_n\} \ldots \{x_1 \mapsto t_1\}$ is a most general unifier of P.

Critical pair

A non-variable term t' and a term t overlap if there exists a position ω in t such that $t_{|\omega}$ and t' are unifiable (with $t_{|\omega}$ not a variable).

Critical pair

A non-variable term t' and a term t overlap if there exists a position ω in t such that $t_{|\omega}$ and t' are unifiable (with $t_{|\omega}$ not a variable).

Do $0 + x \rightarrow x$ and $s(x) + y \rightarrow s(x + y)$ overlap ?

Critical pair

A non-variable term t' and a term t overlap if there exists a position ω in t such that $t_{|\omega}$ and t' are unifiable (with $t_{|\omega}$ not a variable).

Do $0 + x \rightarrow x$ and $s(x) + y \rightarrow s(x + y)$ overlap ?

Where do $(x + y) + z$ and $(x' + y') + z'$ overlap ?

Critical Pairs

Superposition

$$\mathrm{l}_1 \rightarrow \mathrm{r}_1 \qquad\qquad \mathrm{l}_2[\mathrm{u}] \rightarrow \mathrm{r}_2$$

$$\mathrm{l}_2[r_1]\sigma = \mathrm{r}_2\sigma$$

u is a non-variable sub-term of l_2
σ is the $mgu(u, l_1)$

Critical Pairs

Superposition

$$l_1 \rightarrow r_1 \qquad\qquad l_2[u] \rightarrow r_2$$
$$l_2[r_1]\sigma = r_2\sigma$$

u is a non-variable sub-term of l_2
σ is the $mgu(u, l_1)$

Do $0 + x \rightarrow x$ and $(x + y) + z \rightarrow x + (y + z)$ overlap ?

Critical Pairs

Superposition

$$\mathrm{l}_1 \rightarrow \mathrm{r}_1 \qquad\qquad \mathrm{l}_2[\mathrm{u}] \rightarrow \mathrm{r}_2$$

$$\mathrm{l}_2[r_1]\sigma = \mathrm{r}_2\sigma$$

u is a non-variable sub-term of l_2
σ is the $mgu(u, l_1)$

Do $0 + x \rightarrow x$ and $(x + y) + z \rightarrow x + (y + z)$ overlap ?

Compute the critical pairs between these two rules.

Critical Pair Lemma

R is locally confluent iff all critical pair satisfies :

$$l_2[r_1]\sigma \xrightarrow{*}_R \otimes \; {}_R\xleftarrow{*} r_2\sigma$$

Critical Pair Lemma

R is locally confluent iff all critical pair satisfies :

$$l_2[r_1]\sigma \xrightarrow{*}_R \otimes \;{}_R\xleftarrow{*} r_2\sigma$$

Prove that the following rewrite systen is locally confluent :

$$\begin{array}{rcl} (x * y) * z & \rightarrow & x * (y * z) \\ f(x * y) & \rightarrow & f(x) * f(y) \end{array}$$

Prove that it is confluent.

Orthogonal systems

A rewrite system that is both linear (the left-hand side of each rule is a linear term) and non-overlapping is called orthogonal.

Orthogonal systems

A rewrite system that is both linear (the left-hand side of each rule is a linear term) and non-overlapping is called orthogonal.

Theorem If a rewrite system is orthogonal, then it is confluent.

Orthogonal systems

A rewrite system that is both linear (the left-hand side of each rule is a linear term) and non-overlapping is called orthogonal.

Theorem If a rewrite system is orthogonal, then it is confluent.

Linearity is needed for non-terminating rewriting system :

$$\begin{array}{ll} d(x,x) & \rightarrow t \\ d(x,c(x)) & \rightarrow f \\ a & \rightarrow c(a) \end{array}$$

Other systems

What if the system is non-terminating and non-orthogonal ?

Other systems

What if the system is non-terminating and non-orthogonal ?

Theorem Consider a reduction relation $\longrightarrow_R$ and let $\longrightarrow_D$ s.t.

- $\longrightarrow_R \subseteq \longrightarrow_D \subseteq \overset{*}{\longrightarrow}_R$
- $\longrightarrow_D$ has the diamond property

Then, $\longrightarrow_R$ is confluent.

Completion of TRS

The group example

Let us concentrate on the use of rewriting for proving equational theorems.

$$G = \left\{ \begin{array}{lrl} [Assoc] & (x+y)+z = & x+(y+z) \\ [NElmt] & x+0 = & x \\ [Inver] & x+i(x) = & 0 \end{array} \right.$$

where these three equational axioms are implicitly assumed to be universaly quantified.

The group example

Let us concentrate on the use of rewriting for proving equational theorems.

$$G = \left\{ \begin{array}{lrr} [\mathit{Assoc}] & (x+y)+z = & x+(y+z) \\ [\mathit{NElmt}] & x+0 = & x \\ [\mathit{Inver}] & x+i(x) = & 0 \end{array} \right.$$

where these three equational axioms are implicitly assumed to be universaly quantified.
Simple (?) exercice, prove that $0+x=x$.

What is completion ?

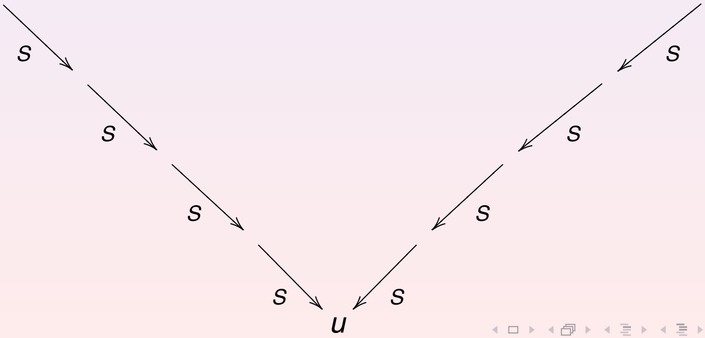
Transform any equational proof in E into a valley proof in R :

What is completion ?

Transform any equational proof in E into a valley proof in R :

$$u_0 \quad =_E \quad u_1 \quad =_E \quad \cdots \quad =_E \quad \cdots \quad =_E \quad u_{n-1} \quad =_E \quad u_n$$

$$\leftarrow_R \quad \leftarrow_R \quad \cdots \quad \rightarrow_R \quad \cdots \quad \leftarrow_R \quad \rightarrow_R$$



Completion as a compilation process

Given an equational theory E

Completion as a compilation process

Given an equational theory E
Find a term rewrite system R

Completion as a compilation process

Given an equational theory E
Find a term rewrite system R
Such that,

$$E \vdash t = t' \iff t \xrightarrow{*}_R \cdot {}_R\xleftarrow{*} t'$$

First completion principle : ORIENT

Orient equalities to build (at least) a well founded ordering

First completion principle : ORIENT

Orient equalities to build (at least) a well founded ordering
Simple example

$$x + 0 = x \text{ is oriented into } x + 0 \rightarrow x$$

First completion principle : ORIENT

Orient equalities to build (at least) a well founded ordering
Simple example

$$x + 0 = x \text{ is oriented into } x + 0 \rightarrow x$$

Less obvious, how to orient

$$(x + y) + z = x + (y + z)$$

First completion principle : ORIENT

Orient equalities to build (at least) a well founded ordering
Simple example

$$x + 0 = x \text{ is oriented into } x + 0 \rightarrow x$$

Less obvious, how to orient

$$(x + y) + z = x + (y + z)$$

Furthermore, well-founded orderings are used to decrease proof complexity

Completion of groups : starts with

$$P = \begin{cases} x + e & = & x \\ x + (y + z) & = & (x + y) + z \\ x + i(x) & = & e \end{cases} \qquad R = \emptyset$$

Completion of groups : starts with

$$P = \left\{ \begin{array}{lcl} x + e & = & x \\ x + (y + z) & = & (x + y) + z \\ x + i(x) & = & e \end{array} \right. \qquad R = \emptyset$$

Apply saturation rules

Orient	$P \cup \{p = q\}, R$	$\mapsto\!\!\!\twoheadrightarrow$	$P, R \cup \{p \rightarrow q\}$ si $p > q$
Deduce	P, R	$\mapsto\!\!\!\twoheadrightarrow$	$P \cup \{p = q\}, R$ si $(p, q) \in CP(R)$
Simplify	$P \cup \{p = q\}, R$	$\mapsto\!\!\!\twoheadrightarrow$	$P \cup \{p' = q\}, R$ si $p \rightarrow_R p'$
Delete	$P \cup \{p = p\}, R$	$\mapsto\!\!\!\twoheadrightarrow$	P, R
Compose	$P, R \cup \{l \rightarrow r\}$	$\mapsto\!\!\!\twoheadrightarrow$	$P, R \cup \{l \rightarrow r'\}$ si $r \rightarrow_R r'$
Collapse	$P, R \cup \{l \rightarrow r\}$	$\mapsto\!\!\!\twoheadrightarrow$	$P \cup \{l' = r\}, R$ si $l \rightarrow_R^{g \rightarrow d} l'$ and $l \rightarrow r \gg g \rightarrow d$

Completion of groups : ends with

$Q = \emptyset$

$$R = \left\{ \begin{array}{lll} x + e & \rightarrow & x \\ e + x & \rightarrow & x \\ x + (y + z) & \rightarrow & (x + y) + z \\ x + i(x) & \rightarrow & e \\ i(x) + x & \rightarrow & e \\ i(e) & \rightarrow & e \\ (y + i(x)) + x & \rightarrow & y \\ (y + x) + i(x) & \rightarrow & y \\ i(i(x)) & \rightarrow & x \\ i(x + y) & \rightarrow & i(y) + i(x) \end{array} \right.$$

[Knuth & Bendix 1970]

The associated proof transformations

1. Orient : $t \overset{*}{\longleftrightarrow}{}^{p=q}_{P} t' \Longrightarrow t \rightarrowtriangle^{p \rightarrowtriangle q}_{R} t'$
2. Deduce : $t' \leftarrowtriangle^{l \rightarrowtriangle r}_{R} t \rightarrowtriangle^{g \rightarrowtriangle d}_{R} t'' \Longrightarrow t' \longleftrightarrow^{p=q}_{P} t''$
3. Simplify : $t \longleftrightarrow^{p=q}_{P} t' \Longrightarrow t \rightarrowtriangle^{l \rightarrowtriangle r}_{R} t'' \longleftrightarrow^{p'=q}_{P} t'$ if $p \rightarrowtriangle^{l \rightarrowtriangle r}_{R} p'$.
4. Delete : $t \longleftrightarrow^{p=p}_{P} t \Longrightarrow \Lambda$
5. Compose : $t \rightarrowtriangle^{l \rightarrowtriangle r}_{R} t' \Longrightarrow t \rightarrowtriangle^{l \rightarrowtriangle r'}_{R} t'' \leftarrowtriangle^{g \rightarrowtriangle d}_{R} t'$ if $r \rightarrowtriangle^{g \rightarrowtriangle d}_{R} r'$.
6. Collapse : $t \rightarrowtriangle^{l \rightarrowtriangle r}_{R} t' \Longrightarrow t \rightarrowtriangle^{g \rightarrowtriangle d}_{R} t'' \longleftrightarrow^{l'=r}_{P} t'$ if $l \rightarrowtriangle^{g \rightarrowtriangle g}_{R} l'$.
7. Peak without overlap : $t' \leftarrowtriangle^{l \rightarrowtriangle r}_{R} t \rightarrowtriangle^{g \rightarrowtriangle d}_{R} t'' \Longrightarrow t' \rightarrowtriangle^{g \rightarrowtriangle d}_{R} t_1 \leftarrowtriangle^{l \rightarrowtriangle r}_{R} t''$
8. Peak with variable overlap :
   $t' \leftarrowtriangle^{l \rightarrowtriangle r}_{R} t \rightarrowtriangle^{g \rightarrowtriangle d}_{R} t'' \Longrightarrow t' \overset{*}{\longrightarrow}_{R} t_1 \longleftarrow *_{R} t''$

The main result

The sets of persisting rules and pairs are defined as :

$$P_\infty = \bigcup_{i\geq 0} \bigcap_{j>i} P_j \quad \text{and} \quad R_\infty = \bigcup_{i\geq 0} \bigcap_{j>i} R_j.$$

If the derivation $(P_0, R_0) \mapsto\!\!\!\!\twoheadrightarrow (P_1, R_1) \mapsto\!\!\!\!\twoheadrightarrow \cdots$ satisfies

- $CP(R_\infty)$ is a subset of $\bigcup_{i\geq 0} P_i$ (i.e. the set of all generated equalities),
- R_∞ is reduced and
- P_∞ is empty,

then R_∞ is Church-Rosser and terminating.
$\overset{*}{\longleftrightarrow}_{P_0 \cup R_0}$ and $\overset{*}{\longleftrightarrow}_{R_\infty}$ coincides.

Three possible issues

A completion process may

- terminate
- diverge by generating infinitely many rules
- fail on an unorientable equation

Exercise

Let $\mathcal{F} = \{c, f\}$ where c is a constant and f a unary operator. Complete the set of equalities

$$\begin{aligned} f(f(f(f(f(x))))) &= x \\ f(f(f(x))) &= x \end{aligned}$$

Exemple

The theory of idempotent semi-groups (sometimes called bands) is defined by a set E of two axioms :

$$\begin{array}{rcl} (x * y) * z & = & x * (y * z) \\ x * x & = & x \end{array}$$

From $P_0 = E$ the completion generates

$$\begin{array}{rcl} (x * y) * z & \rightarrow & x * (y * z) \\ x * x & \rightarrow & x \\ x * (x * z) & \rightarrow & x * z \\ x * (y * (x * y)) & \rightarrow & x * y \\ x * (y * (x * (y * z))) & \rightarrow & x * (y * z) \\ & \ldots & \\ x * (y * (z * (y * (x * (y * (z * x)))))) & \rightarrow & x * (y * (z * x)) \\ & \ldots & \end{array}$$

Matching and Rewriting Modulo

Equality modulo C

$$C(+): \quad \forall x, y \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \quad x + y = y + x$$

For example, on Peano integer, $+$ is commutative :

$$(s(0) + (x + s(y))) + x =_{C(+)} ((s(y) + x) + s(0)) + x$$

Theorem :

$$t_1 + t_2 =_{C(+)} t'_1 + t'_2 \iff \begin{array}{l} (t_1 =_{C(+)} t'_1 \ \wedge \ t_2 =_{C(+)} t'_2) \\ \vee \\ (t_1 =_{C(+)} t'_2 \ \wedge \ t_2 =_{C(+)} t'_1) \end{array}$$

Matching modulo

Finding a substitution σ such that

$$\sigma(l) = t$$

is called the matching problem from l to t (denoted $l \ll^? t$).

Matching modulo

Finding a substitution σ such that

$$\sigma(l) = t$$

is called the matching problem from l to t (denoted $l \ll^{?} t$).

Finding a substitution σ such that

$$\sigma(l) =_E t$$

is called the matching problem from l to t (denoted $l \ll^{?}_{E} t$).

Examples (commutative symbol(s))

$\mathcal{F} = \{a(0), b(0), c(0), f(2), g(2), h(1)\}$
f is assumed to be commutative (the other symbols have no property).

$$C(f): \quad \forall x, y \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \quad f(x, y) = f(y, x)$$

- $f(a, b) = f(b, a)$

Examples (commutative symbol(s))

$\mathcal{F} = \{a(0), b(0), c(0), f(2), g(2), h(1)\}$
f is assumed to be commutative (the other symbols have no property).

$$C(f): \ \forall x, y \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \quad f(x, y) = f(y, x)$$

- $f(a, b) = f(b, a)$ — yes
- $g(f(a, b), a) = g(f(b, a), a)$

Examples (commutative symbol(s))

$\mathcal{F} = \{a(0), b(0), c(0), f(2), g(2), h(1)\}$
f is assumed to be commutative (the other symbols have no property).

$$C(f): \quad \forall x, y \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \quad f(x, y) = f(y, x)$$

- $f(a, b) = f(b, a)$ — yes
- $g(f(a, b), a) = g(f(b, a), a)$ — yes
- $g(f(a, b), a) = g(a, f(b, a))$

Examples (commutative symbol(s))

$\mathcal{F} = \{a(0), b(0), c(0), f(2), g(2), h(1)\}$
f is assumed to be commutative (the other symbols have no property).

$$C(f): \ \forall x, y \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \quad f(x, y) = f(y, x)$$

- $f(a, b) = f(b, a)$ — yes
- $g(f(a, b), a) = g(f(b, a), a)$ — yes
- $g(f(a, b), a) = g(a, f(b, a))$ — no
- $f(a, f(a, b)) = f(f(b, a), a)$

Examples (commutative symbol(s))

$\mathcal{F} = \{a(0), b(0), c(0), f(2), g(2), h(1)\}$
f is assumed to be commutative (the other symbols have no property).

$$C(f): \quad \forall x, y \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \quad f(x, y) = f(y, x)$$

- $f(a, b) = f(b, a)$ — yes
- $g(f(a, b), a) = g(f(b, a), a)$ — yes
- $g(f(a, b), a) = g(a, f(b, a))$ — no
- $f(a, f(a, b)) = f(f(b, a), a)$ — yes
- $f(a, f(b, c)) = f(f(c, b), a)$

Examples (commutative symbol(s))

$\mathcal{F} = \{a(0), b(0), c(0), f(2), g(2), h(1)\}$
f is assumed to be commutative (the other symbols have no property).

$$C(f): \quad \forall x, y \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \quad f(x, y) = f(y, x)$$

- $f(a, b) = f(b, a)$ — yes
- $g(f(a, b), a) = g(f(b, a), a)$ — yes
- $g(f(a, b), a) = g(a, f(b, a))$ — no
- $f(a, f(a, b)) = f(f(b, a), a)$ — yes
- $f(a, f(b, c)) = f(f(c, b), a)$ — yes
- $f(f(a, b), c) = f(a, f(b, c))$

Examples (commutative symbol(s))

$\mathcal{F} = \{a(0), b(0), c(0), f(2), g(2), h(1)\}$
f is assumed to be commutative (the other symbols have no property).

$$C(f): \quad \forall x, y \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \quad f(x, y) = f(y, x)$$

- $f(a, b) = f(b, a)$ — yes
- $g(f(a, b), a) = g(f(b, a), a)$ — yes
- $g(f(a, b), a) = g(a, f(b, a))$ — no
- $f(a, f(a, b)) = f(f(b, a), a)$ — yes
- $f(a, f(b, c)) = f(f(c, b), a)$ — yes
- $f(f(a, b), c) = f(a, f(b, c))$ — no

Matching modulo C : examples

Solve the following problems :

- $f(x, y) \ll^?_C f(a, b)$

Matching modulo C : examples

Solve the following problems :

- $f(x, y) \ll^?_C f(a, b)$
  $\sigma = \{x \mapsto a, y \mapsto b\}$

Matching modulo C : examples

Solve the following problems :

- $f(x, y) \ll^?_C f(a, b)$
  $\sigma = \{x \mapsto a, y \mapsto b\}$
  $\sigma = \{x \mapsto b, y \mapsto a\}$

Matching modulo C : examples

Solve the following problems :

- $f(x, y) \ll^?_C f(a, b)$
  $\sigma = \{x \mapsto a, y \mapsto b\}$
  $\sigma = \{x \mapsto b, y \mapsto a\}$

- $f(y, f(x, x)) \ll^?_C f(f(f(a, b), f(b, a)), f(b, a))$

Matching modulo C : examples

Solve the following problems :

- $f(x, y) \ll^?_C f(a, b)$
  $\sigma = \{x \mapsto a, y \mapsto b\}$
  $\sigma = \{x \mapsto b, y \mapsto a\}$

- $f(y, f(x, x)) \ll^?_C f(f(f(a, b), f(b, a)), f(b, a))$
  $\sigma = \{x \mapsto f(a, b), y \mapsto f(a, b)\}$

Matching modulo C : A rule based description

Delete		$t \ll^? t \;\wedge\; P$	
	$\mapsto\!\!\!\!\twoheadrightarrow$	P	
Decomposition		$f(t_1, \ldots, t_n) \ll^? f(t'_1, \ldots, t'_n) \;\wedge\; P$	
	$\mapsto\!\!\!\!\twoheadrightarrow$	$\bigwedge_{i=1,\ldots,n} t_i \ll^? t'_i \;\wedge\; P$	
SymbolClash		$f(t_1, \ldots, t_n) \ll^? g(t'_1, \ldots, t'_m) \;\wedge\; P$	
	$\mapsto\!\!\!\!\twoheadrightarrow$	*Fail*	if $f \neq g$
SymbolVariableClash		$f(t_1, \ldots, t_n) \ll^? x \;\wedge\; P$	
	$\mapsto\!\!\!\!\twoheadrightarrow$	*Fail*	if $x \in \mathcal{X}$
MergingClash		$x \ll^? t \;\wedge\; x \ll^? t' \;\wedge\; P$	
	$\mapsto\!\!\!\!\twoheadrightarrow$	*Fail*	if $t \neq t'$

Assume $+$ commutative

$$
\begin{array}{ll}
\mathrm{C}{-}Dec & t_1 + t_2' \ll^?_C t_1' + t_2' \ \wedge \ P \\
\mapsto\!\!\!\!\twoheadrightarrow & (t_1 \ll^?_C t_1' \wedge t_2 \ll^?_C t_2' \ \wedge \ P) \vee (t_1 \ll^?_C t_2' \wedge t_2 \ll^?_C t_1' \ \wedge \ P)
\end{array}
$$

Find a match

$$x*(3+y) \ll^?_C 1*(4+3)$$

$$\Rightarrow_{\text{Decomposition}}\ x \ll^?_C 1 \;\wedge\; 3+y \ll^?_C 4+3$$

$$\Rightarrow_{\text{C}(+)-\text{Decomposition}}\ x \ll^?_C 1 \wedge ((3 \ll^?_C 4 \wedge y \ll^?_C 3) \vee (3 \ll^?_C 3 \wedge y \ll^?_C 4))$$

$$\Rightarrow_{\text{MergingClash}}\ x \ll^?_C 1 \;\wedge\; (\mathit{Fail} \vee (3 \ll^?_C 3 \;\wedge\; y \ll^?_C 4))$$

$$\Rightarrow_{\text{Delete}}\ x \ll^?_C 1 \;\wedge\; (\mathit{Fail} \vee (y \ll^?_C 4))$$

$$\Rightarrow_{\text{Bool}}\ x \ll^?_C 1 \;\wedge\; y \ll^?_C 4$$

Matching rules

Does it terminate ?
Do we always get the same result ?

Matching rules

Does it terminate ?
Do we always get the same result ?

Theorem The normal form by the rules in $Commutative - Match$, of any matching problem $t \ll^{?} t'$ such that $\mathcal{V}ar(t) \cap \mathcal{V}ar(t') = \emptyset$, exists and is unique.

1. If it is *Fail*, then there is no match from t to t'.
2. If it is of the form $\bigvee_{k \in K} \bigwedge_{i \in I} x_i^k \ll_C^{?} t_i^k$ with $I, K \neq \emptyset$, the substitutions $\sigma^k = \{x_i^k \mapsto t_i^k\}_{i \in I}$ are all the matches from t to t'.
3. If it is empty then t and t' are identical : $t = t'$.

Matching modulo associativity-commutativity (1)

$\cup$ is assumed to be an associative commutative (AC) symbol :

$\forall x, y, z, \;\; \cup(x, \cup(y, z)) = \cup(\cup(x, y), z)$ and $\forall x, y, \cup(x, y) = \cup(y, x)$.

$$\{i\} \cup s \ll^{?}_{AC} \{1\} \cup \{\,2\} \cup \{\,3\} \cup \{\,4\} \cup \{\,5\}$$

Matching modulo associativity-commutativity (1)

$\cup$ is assumed to be an associative commutative (AC) symbol :

$\forall x, y, z, \ \cup(x, \cup(y, z)) = \cup(\cup(x, y), z)$ and $\forall x, y, \cup(x, y) = \cup(y, x)$.

$$\{i\} \cup s \ll^{?}_{AC} \{1\} \cup \{2\} \cup \{3\} \cup \{4\} \cup \{5\}$$

$$\{1\} \cup \{2\} \cup \{3\} \cup \{4\} \cup \{5\} \quad =_{AC}$$
$$\{2\} \cup \{3\} \cup \{4\} \cup \{5\} \cup \{1\} \quad =_{AC}$$
$$\ldots$$
$$\{5\} \cup \{1\} \cup \{2\} \cup \{3\} \cup \{4\}$$

5 different and non *AC*-equivalent matches.

Matching modulo AC : examples

Solve the following problems :

- $f(x, y) \ll^{?}_{AC} f(a, b)$

Matching modulo AC : examples

Solve the following problems :

- $f(x, y) \ll^{?}_{AC} f(a, b)$
  $\sigma = \{x \mapsto a, y \mapsto b\}$
  $\sigma = \{x \mapsto b, y \mapsto a\}$

Matching modulo AC : examples

Solve the following problems :

- $f(x, y) \ll^?_{AC} f(a, b)$
  $\sigma = \{x \mapsto a, y \mapsto b\}$
  $\sigma = \{x \mapsto b, y \mapsto a\}$

- $f(y, f(x, x)) \ll^?_{AC} f(f(f(a, b), f(b, a)), f(b, a))$

Matching modulo AC : examples

Solve the following problems :

- $f(x, y) \ll^{?}_{AC} f(a, b)$
  $\sigma = \{x \mapsto a, y \mapsto b\}$
  $\sigma = \{x \mapsto b, y \mapsto a\}$

- $f(y, f(x, x)) \ll^{?}_{AC} f(f(f(a, b), f(b, a)), f(b, a))$
  $\sigma = \{x \mapsto f(a, b), y \mapsto f(a, b)\}$

Matching modulo *AC* : examples

Solve the following problems :

- $f(x, y) \ll^?_{AC} f(a, b)$
  $\sigma = \{x \mapsto a, y \mapsto b\}$
  $\sigma = \{x \mapsto b, y \mapsto a\}$

- $f(y, f(x, x)) \ll^?_{AC} f(f(f(a, b), f(b, a)), f(b, a))$
  $\sigma = \{x \mapsto f(a, b), y \mapsto f(a, b)\}$
  $\sigma = \{x \mapsto a, y \mapsto f(f(b, b), f(b, a))\}$

Matching modulo *AC* : examples

Solve the following problems :

- $f(x, y) \ll^?_{AC} f(a, b)$
  $\sigma = \{x \mapsto a, y \mapsto b\}$
  $\sigma = \{x \mapsto b, y \mapsto a\}$

- $f(y, f(x, x)) \ll^?_{AC} f(f(f(a, b), f(b, a)), f(b, a))$
  $\sigma = \{x \mapsto f(a, b), y \mapsto f(a, b)\}$
  $\sigma = \{x \mapsto a, y \mapsto f(f(b, b), f(b, a))\}$
  $\sigma = \{x \mapsto b, y \mapsto f(f(a, a), f(b, a))\}$

Matching modulo AC : examples

Solve the following problems :

- $f(x, y) \ll^?_{AC} f(a, b)$
  $\sigma = \{x \mapsto a, y \mapsto b\}$
  $\sigma = \{x \mapsto b, y \mapsto a\}$

- $f(y, f(x, x)) \ll^?_{AC} f(f(f(a, b), f(b, a)), f(b, a))$
  $\sigma = \{x \mapsto f(a, b), y \mapsto f(a, b)\}$
  $\sigma = \{x \mapsto a, y \mapsto f(f(b, b), f(b, a))\}$
  $\sigma = \{x \mapsto b, y \mapsto f(f(a, a), f(b, a))\}$
  $\ldots$

Rewriting modulo : definition

A class rewrite system R/A is composed of a set of rewrite rules R and a set of equalities A, such that A and R are disjoint sets.

$$
\begin{array}{rcl}
x + 0 & \rightarrow & x \\
x + (0 + y) & \rightarrow & x + y \\
x + (-x) & \rightarrow & 0 \\
x + ((-x) + y) & \rightarrow & y \\
- - x & \rightarrow & x \\
-0 & \rightarrow & 0 \\
-(x + y) & \rightarrow & (-x) + (-y)
\end{array}
$$

$$
\begin{array}{rcl}
x + y & = & y + x \\
(x + y) + z & = & x + (y + z)
\end{array}
$$

$\longrightarrow_{R/A}$

t (R/A)-rewrites to t' if $t =_A t_1 \longrightarrow_R t_2 =_A t'$

$\longrightarrow_{R/A}$

t (R/A)-rewrites to t' if $t =_A t_1 \longrightarrow_R t_2 =_A t'$

To be more effective, consider any relation $\longrightarrow_{RA}$ such that :

$$\longrightarrow_R \subseteq \longrightarrow_{RA} \subseteq \longrightarrow_{R/A}$$

$\longrightarrow\!\!\!\triangleright_{R,A}$

A term rewrite system R (a set of rewrite rules) determines a relation on terms denoted $\longrightarrow_{R,A}$ [Peterson & Stickel,81]

$\longrightarrow\!\!\triangleright_{R,A}$

A term rewrite system R (a set of rewrite rules) determines a relation on terms denoted $\longrightarrow_{R,A}$ [Peterson & Stickel,81]

$$u \longrightarrow\!\!\triangleright_{R,A} v$$

iff
there exist $l \longrightarrow\!\!\triangleright r \in R$, an occurrence ω in t, such that

$$u_{|\omega} =_A \sigma(l)$$

and

$$v = u[\sigma(r)]_\omega$$

$\longrightarrow_{R,A}$

A term rewrite system R (a set of rewrite rules) determines a relation on terms denoted $\longrightarrow_{R,A}$ [Peterson & Stickel,81]

$$u \longrightarrow_{R,A} v$$

iff

there exist $l \to r \in R$, an occurrence ω in t, such that

$$u_{|\omega} =_A \sigma(l)$$

and

$$v = u[\sigma(r)]_\omega$$

USUALLY, when defining the rewriting relation, one requires the all rewrite rules satisfy $Var(r) \subseteq Var(l)$.

For example

Let $\cup$ be an AC symbol, such that

$$\{i\} \cup x \rightarrow i$$

$$\begin{array}{ll}
\{1\} \cup \{2\} \cup \{3\} \cup \{4\} \cup \{5\} & =_{AC} \\
\{2\} \cup \{3\} \cup \{4\} \cup \{5\} \cup \{1\} & =_{AC} \\
\ldots & \\
\{5\} \cup \{1\} \cup \{2\} \cup \{3\} \cup \{4\} &
\end{array}$$

Since this term matches the lefthand side of the rewriting rule in 5 different and non *AC*-equivalent ways, the rewrite rule applies in 5 different ways.

Examples

Assume $+$ to be AC (associative and commutative)

$$\boxed{R = \{a + a \rightarrow a\}}$$

Examples

Assume $+$ to be AC (associative and commutative)

$$R = \{a + a \rightarrow a\}$$

R/E -rewrite the term $(a + c) + a$

Examples

Assume $+$ to be AC (associative and commutative)

$$R = \{a + a \rightarrow a\}$$

R/E -rewrite the term $(a + c) + a$ a+c

Examples

Assume $+$ to be AC (associative and commutative)

$$R = \{a + a \rightarrow a\}$$

R/E -rewrite the term $(a + c) + a$ a+c

R, E -rewrite the term $(a + c) + a$

Examples

Assume $+$ to be AC (associative and commutative)

$$R = \{a + a \rightarrow a\}$$

R/E -rewrite the term $(a + c) + a$ a+c

R, E -rewrite the term $(a + c) + a$

$$R = \{a + a \rightarrow a \quad (a + a) + x \rightarrow a + x\}$$

Examples

Assume $+$ to be AC (associative and commutative)

$$R = \{a + a \rightarrow a\}$$

R/E -rewrite the term $(a + c) + a$ a+c

R, E -rewrite the term $(a + c) + a$

$$R = \{a + a \rightarrow a \quad (a + a) + x \rightarrow a + x\}$$

R/E -rewrite the term $(a + c) + a$

Examples

Assume $+$ to be AC (associative and commutative)

$$R = \{a + a \rightarrow a\}$$

R/E -rewrite the term $(a + c) + a$ a+c

R, E -rewrite the term $(a + c) + a$

$$R = \{a + a \rightarrow a \quad (a + a) + x \rightarrow a + x\}$$

R/E -rewrite the term $(a + c) + a$ a+c

Examples

Assume $+$ to be AC (associative and commutative)

$$R = \{a + a \rightarrow a\}$$

R/E -rewrite the term $(a + c) + a$ a+c

R, E -rewrite the term $(a + c) + a$

$$R = \{a + a \rightarrow a \quad (a + a) + x \rightarrow a + x\}$$

R/E -rewrite the term $(a + c) + a$ a+c

R, E -rewrite the term $(a + c) + a$

Examples

Assume $+$ to be AC (associative and commutative)

$$R = \{a + a \rightarrow a\}$$

R/E -rewrite the term $(a + c) + a$ a+c

R, E -rewrite the term $(a + c) + a$

$$R = \{a + a \rightarrow a \quad (a + a) + x \rightarrow a + x\}$$

R/E -rewrite the term $(a + c) + a$ a+c

R, E -rewrite the term $(a + c) + a$ a+c

$\longrightarrow\!\!\triangleright\, RA$

- Huet's approach [JACM80] uses standard rewriting $\longrightarrow\!\!\triangleright_R$ but is restricted to left-linear rules.
- Peterson and Stickel's approach [JACM81] uses *rewriting modulo* A, denoted $\longrightarrow\!\!\triangleright_{R,A}$, and requires matching modulo A.
- Pedersen's approach [Phd84] uses a restricted version of matching modulo A, confined to variables.
- Jouannaud and Kirchner's method [SIAM86] uses standard rewriting with left-linear rules and rewriting modulo A with non-left-linear rules, mixing advantages of the two first methods.

Definitions

The rewriting relation RA is

- Church-Rosser modulo A if

$$=_{R \cup A} \subseteq \xrightarrow{*}_{RA} \circ =_A \circ \ {}_{RA}\xleftarrow{*} .$$

- confluent modulo A if

$$_{RA}\xleftarrow{*} \circ \xrightarrow{*}_{RA} \ \subseteq \ \xrightarrow{*}_{RA} \circ =_A \circ \ {}_{RA}\xleftarrow{*}$$

- locally coherent with R modulo A if

$$_{RA}\longleftarrow \circ \longrightarrow_R \ \subseteq \ \xrightarrow{*}_{RA} \circ =_A \circ \ {}_{RA}\xleftarrow{*}$$

- locally coherent with A modulo A if

$$_{RA}\longleftarrow \circ =_A \subseteq \ \xrightarrow{*}_{RA} \circ =_A \circ \ {}_{RA}\xleftarrow{*}$$

Good news

If R/A is terminating, the following properties are equivalent :

1. $\rightarrow_{RA}$ is Church-Rosser modulo A.
2. $\rightarrow_{RA}$ is confluent modulo A and $\rightarrow_{RA}$ is coherent modulo A.
3. $\rightarrow_{RA}$ is locally confluent with R modulo A and locally coherent with A modulo A.
4. $\forall t, t', t =_{R \cup A} t'$ iff $t \downarrow_{RA} =_A t' \downarrow_{RA}$.

Rewriting and theorem proving, a few examples

- **Boolean algebras and rings** Applications to proof search in first order logic (Hsiang, 1985).
- **Proof of commutativity in specific rings**

$$(\forall x, x^n = x) \Rightarrow \forall x, y, (x * y = y * x)$$

  $n = 3$ (Stickel, 1984), n pair (Kapur,Zhang, 1991).

- **The Robbins conjecture** (McCune, 1996)
  In a Boolean algebra

$$\overline{\overline{\overline{x} + y} + \overline{x + y}} \;=\; y$$

  implies

$$\overline{\overline{x} + \overline{y}} + \overline{x + \overline{y}} \;=\; y$$

References on rewriting modulo

- G. Huet. Confluent reductions : Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4) :797–821, October 1980.
- G. Peterson and M. E. Stickel. Complete sets of reductions for some equational theories. *Journal of the ACM*, 28 :233–264, 1981.
- J.-P. Jouannaud and Hélène Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal of Computing*, 15(4) :1155–1194, 1986.
- Enno Ohlebusch. Church-Rosser Theorems for Abstract Reduction Modulo an Equivalence Relation RTA, pages 17-31, LNCS 1379, 1998.
- Claude and Hélène Kirchner. Rewriting Solving Proving `www.loria.fr/~ckirchne/rsp.ps.gz`

Rewrite rules ...

Rewrite rules describe local transformations

Rewrite derivations are computations

Normal forms are the results

t is in normal form if it cannot be reduced anymore : result of terminating computations

t has a unique normal form if the rewrite system is terminating and confluent.

Paradigm of computation in algebraic languages : ASF+SDF, OBJ, Maude,...

and in functional languages : ML, Haskell,...

... and Strategy

Strategies describe the control of rewrite rule application

- traversals : innermost, outermost, lazy... (Stratego)
- higher-order functions with choice and iteration (ELAN, TOM)

Strategies are ALWAYS needed

1- Even for “good” TRSs
leftmost innermost strategy
i.e. to make clear how the computation is performed

2- To describe the way deduction should be done
Lazy evaluation
Search plans
Action plans
Tactics
User interaction

3- This requires to search for a particular derivation corresponding to the desired strategy.

rewrite rewrite rewrite rewrite rewrite rewrite rewrite rewrite

Logic Programming, Theorem Proving, Constraint Solving are instances of the same deduction schema :

Apply rewrite rules (may be modulo) on formulas with some strategy, until getting specific forms

- Rewrite blindly : implements computations
- Rewrite wisely : implements deduction

Back to Abstract rewrite systems

An **Abstract Rewrite System (ARS)** is a labelled oriented graph $(\mathcal{O}, \mathcal{S})$.
The nodes in $\mathcal{O}$ are called **objects**
The oriented labelled edges in $\mathcal{S}$ are called **steps**.

1. $$\mathcal{A}_{lc} = \begin{tikzcd} a \arrow[r, bend left, "\phi_1"] \arrow[d, "\phi_2"'] & b \arrow[l, bend left, "\phi_3"] \arrow[d, "\phi_4"] \\ c & d \end{tikzcd}$$

2. $$\mathcal{A}_c = \begin{tikzcd} a \arrow[loop above, "\phi_2"] \arrow[loop below, "\phi_1"] \end{tikzcd}$$

Reductions

For a given ARS $\mathcal{A}$:

1. A reduction step is an oriented labelled edge ϕ together with its source *a* and target *b*, written $a \to^{\phi}_{\mathcal{A}} b$.

Reductions

For a given ARS $\mathcal{A}$:

1. A **reduction step** is an oriented labelled edge ϕ together with its source a and target b, written $a \rightarrow^{\phi}_{\mathcal{A}} b$.
2. **$\mathcal{A}$-derivation** : $\pi : a_0 \rightarrow^{\phi_0} a_1 \rightarrow^{\phi_1} a_2 \ldots \rightarrow^{\phi_{n-1}} a_n$ or $a_0 \rightarrow^{\pi} a_n$.
   The **source** of π is a_0 and $dom(\pi) = \{a_0\}$.
   The **target** of π is a_n and $\pi a_0 = \{a_n\}$.

Reductions

For a given ARS $\mathcal{A}$:

1. A **reduction step** is an oriented labelled edge ϕ together with its source a and target b, written $a \rightarrow^{\phi}_{\mathcal{A}} b$.
2. **$\mathcal{A}$-derivation** : $\pi : a_0 \rightarrow^{\phi_0} a_1 \rightarrow^{\phi_1} a_2 \ldots \rightarrow^{\phi_{n-1}} a_n$ or $a_0 \rightarrow^{\pi} a_n$.
   The **source** of π is a_0 and $dom(\pi) = \{a_0\}$.
   The **target** of π is a_n and $\pi a_0 = \{a_n\}$.
3. A derivation is empty when its source and target are the same. The empty derivation issued from a is denoted by id_a.

Reductions

For a given ARS $\mathcal{A}$:

1. A **reduction step** is an oriented labelled edge ϕ together with its source a and target b, written $a \rightarrow^{\phi}_{\mathcal{A}} b$.
2. **$\mathcal{A}$-derivation** : $\pi : a_0 \rightarrow^{\phi_0} a_1 \rightarrow^{\phi_1} a_2 \ldots \rightarrow^{\phi_{n-1}} a_n$ or $a_0 \rightarrow^{\pi} a_n$.
   The **source** of π is a_0 and $dom(\pi) = \{a_0\}$.
   The **target** of π is a_n and $\pi a_0 = \{a_n\}$.
3. A derivation is empty when its source and target are the same. The empty derivation issued from a is denoted by id_a. The set of all derivations is denoted $\mathcal{D}(\mathcal{A})$.

Reductions

For a given ARS $\mathcal{A}$:

1. A reduction step is an oriented labelled edge ϕ together with its source a and target b, written $a \to^{\phi}_{\mathcal{A}} b$.
2. $\mathcal{A}$-derivation : $\pi : a_0 \to^{\phi_0} a_1 \to^{\phi_1} a_2 \ldots \to^{\phi_{n-1}} a_n$ or $a_0 \to^{\pi} a_n$.
   The source of π is a_0 and $dom(\pi) = \{a_0\}$.
   The target of π is a_n and $\pi a_0 = \{a_n\}$.
3. A derivation is empty when its source and target are the same. The empty derivation issued from a is denoted by id_a. The set of all derivations is denoted $\mathcal{D}(\mathcal{A})$.
4. The concatenation of two derivations $\pi_1;\pi_2$ is defined as $a \to^{\pi_1}_{\mathcal{A}} b \to^{\pi_2}_{\mathcal{A}} c$ if $\{a\} = dom(\pi_1)$ and $\pi_1 a = dom(\pi_2) = \{b\}$.
   Then $\pi_1;\pi_2 a = \pi_2\pi_1 a = \{c\}$

Properties : Termination

For a given ARS $\mathcal{A} = (\mathcal{O}, \mathcal{S})$:

- $\mathcal{A}$ is terminating (or strongly normalizing) if all its derivations are of finite length ;

Properties : Termination

For a given ARS $\mathcal{A} = (\mathcal{O}, \mathcal{S})$:

- $\mathcal{A}$ is terminating (or strongly normalizing) if all its derivations are of finite length ;
- An object *a* in $\mathcal{O}$ is normalized when the empty derivation is the only one with source *a* (e.g., *a* is the source of no edge) ;

Properties : Termination

For a given ARS $\mathcal{A} = (\mathcal{O}, \mathcal{S})$:

- $\mathcal{A}$ is terminating (or strongly normalizing) if all its derivations are of finite length ;
- An object *a* in $\mathcal{O}$ is normalized when the empty derivation is the only one with source *a* (e.g., *a* is the source of no edge) ;
- A derivation is normalizing when its target is normalized ;

Properties : Termination

For a given ARS $\mathcal{A} = (\mathcal{O}, \mathcal{S})$:

- $\mathcal{A}$ is terminating (or strongly normalizing) if all its derivations are of finite length ;
- An object *a* in $\mathcal{O}$ is normalized when the empty derivation is the only one with source *a* (e.g., *a* is the source of no edge) ;
- A derivation is normalizing when its target is normalized ;
- An ARS is weakly terminating if every object *a* is the source of a normalizing derivation.

Properties : Confluence

An ARS $\mathcal{A} = (\mathcal{O}, \mathcal{S})$ is **confluent** if

for all objects *a*, *b*, *c* in $\mathcal{O}$, and all $\mathcal{A}$-derivations π_1 and π_2,
when $a \longrightarrow^{\pi_1} b$ and $a \longrightarrow^{\pi_2} c$,
there exist *d* in $\mathcal{O}$ and two $\mathcal{A}$-derivations π_3, π_4 such that
$c \longrightarrow^{\pi_3} d$ and $b \longrightarrow^{\pi_4} d$.

Abstract strategies

For a given ARS $\mathcal{A}$:

1. An abstract strategy ζ is a subset of the set of all derivations (finite or not) of $\mathcal{A}$.
2. $\zeta a = \{b \mid \exists \pi \in \zeta$ such that $a \to^{\pi} b\} = \{\pi a \mid \pi \in \zeta\}$.
   When no derivation in ζ has for source a, we say that the strategy application on a fails.
3. $dom(\zeta) = \bigcup_{\delta \in \zeta} dom(\delta)$
4. The strategy that contains all empty derivations is $Id = \{id_a \mid a \in \mathcal{O}\}$.

Examples

1. $\mathcal{A}_{lc} =$
$$\begin{tikzcd} a \arrow[r, bend left, "\phi_1"] \arrow[d, "\phi_2"'] & b \arrow[l, bend left, "\phi_3"] \arrow[d, "\phi_4"] \\ c & d \end{tikzcd}$$
$\mathcal{D}(\mathcal{A}_{lc}) \supset \{id_a, \phi_1, \phi_1\phi_3, \phi_1\phi_4, \phi_1\phi_3\phi_1, (\phi_1\phi_3)^n, (\phi_1\phi_3)^\omega, \ldots\}$, where ϕ^n denotes the n-steps iteration of ϕ and ϕ^ω denotes the infinite iteration of ϕ ;

Examples

1. $\mathcal{A}_{lc} =$
$$\begin{tikzcd} a \arrow[r, bend left, "\phi_1"] \arrow[d, "\phi_2"'] & b \arrow[l, bend left, "\phi_3"] \arrow[d, "\phi_4"] \\ c & d \end{tikzcd}$$
$\mathcal{D}(\mathcal{A}_{lc}) \supset \{id_a, \phi_1, \phi_1\phi_3, \phi_1\phi_4, \phi_1\phi_3\phi_1, (\phi_1\phi_3)^n, (\phi_1\phi_3)^\omega, \ldots\}$, where ϕ^n denotes the n-steps iteration of ϕ and ϕ^ω denotes the infinite iteration of ϕ;

2. $\mathcal{A}_c =$
$$\begin{tikzcd} a \arrow[loop above, "\phi_2"] \arrow[loop below, "\phi_1"] \end{tikzcd}$$
$\mathcal{D}(\mathcal{A}_c) \supset \{\phi_1, \phi_2, \phi_1\phi_2, \ldots, (\phi_1)^\omega, (\phi_2)^\omega, \ldots\}$.

Examples

$$\mathcal{A}_{lc} = \begin{tikzcd} a \arrow[r, bend left, "\phi_1"] \arrow[d, "\phi_2"'] & b \arrow[l, bend left, "\phi_3"] \arrow[d, "\phi_4"] \\ c & d \end{tikzcd}$$

A few strategies :

1. $\zeta_1 = \mathcal{D}(\mathcal{A}_{lc})$, $\zeta_1 a = \{a, b, c, d\}$.

Examples

$$\mathcal{A}_{lc} = \begin{tikzcd} a \arrow[r, bend left, "\phi_1"] \arrow[d, "\phi_2"'] & b \arrow[l, bend left, "\phi_3"] \arrow[d, "\phi_4"] \\ c & d \end{tikzcd}$$

A few strategies :

1. $\zeta_1 = \mathcal{D}(\mathcal{A}_{lc})$, $\zeta_1 a = \{a, b, c, d\}$.
2. $\zeta_2 = \emptyset$, for all x in $\mathcal{O}_{lc}$, $\zeta_2 x = \emptyset$.

Examples

$$\mathcal{A}_{lc} = \begin{tikzcd} a \arrow[r, bend left, "\phi_1"] \arrow[d, "\phi_2"'] & b \arrow[l, bend left, "\phi_3"] \arrow[d, "\phi_4"] \\ c & d \end{tikzcd}$$

A few strategies :

1. $\zeta_1 = \mathcal{D}(\mathcal{A}_{lc})$, $\zeta_1 a = \{a, b, c, d\}$.
2. $\zeta_2 = \emptyset$, for all x in $\mathcal{O}_{lc}$, $\zeta_2 x = \emptyset$.
3. $\zeta_3 = \{(\phi_1\phi_3)^*\phi_2\}$,
   a always converges to c : $\zeta_3 a = \{c\}$;
   b is not transformed (as well as c and d) : $\zeta_3 b = \emptyset$.

Examples

$$\mathcal{A}_{lc} = \begin{tikzcd} a \arrow[r, bend left, "\phi_1"] \arrow[d, "\phi_2"'] & b \arrow[l, bend left, "\phi_3"] \arrow[d, "\phi_4"] \\ c & d \end{tikzcd}$$

A few strategies :

1. $\zeta_1 = \mathcal{D}(\mathcal{A}_{lc})$, $\zeta_1 a = \{a, b, c, d\}$.
2. $\zeta_2 = \emptyset$, for all x in $\mathcal{O}_{lc}$, $\zeta_2 x = \emptyset$.
3. $\zeta_3 = \{(\phi_1\phi_3)^*\phi_2\}$,
   a always converges to c : $\zeta_3 a = \{c\}$;
   b is not transformed (as well as c and d) : $\zeta_3 b = \emptyset$.
4. The result of $((\phi_1\phi_3)^\omega \; a)$ is the empty set.

Termination under strategy

For a given ARS $\mathcal{A} = (\mathcal{O}, \mathcal{S})$ and strategy ζ :

- $\mathcal{A}$ is ζ-terminating if all derivations in ζ are of finite length ;
- An object *a* in $\mathcal{O}$ is ζ-normalized when the empty derivation is the only one in ζ with source *a* ;
- A derivation is ζ-normalizing when its target is ζ-normalized ;
- An ARS is weakly ζ-terminating if every object *a* is the source of a ζ-normalizing derivation.

Example

Given the strategy ζ defined as

$$a \longrightarrow^{\phi_1} b \longrightarrow^{\phi_4} d$$

b is ζ-normalized since there is no derivation in ζ with source b.

Confluence under strategy (1)

Weak Confluence under strategy

An ARS $\mathcal{A} = (\mathcal{O}, \mathcal{S})$ is weakly confluent under strategy ζ if

for all objects a, b, c in $\mathcal{O}$, and all $\mathcal{A}$-derivations π_1 and π_2 in ζ, when $a \to^{\pi_1} b$ and $a \to^{\pi_2} c$

there exists d in $\mathcal{O}$ and two $\mathcal{A}$-derivations π'_3, π'_4 in ζ such that $\pi'_3 : a \to b \to d$ and $\pi'_4 : a \to c \to d$.

Confluence under strategy (2)

Strong Confluence under strategy

An ARS $\mathcal{A} = (\mathcal{O}, \mathcal{S})$ is strongly confluent under strategy ζ if

for all objects *a*, *b*, *c* in $\mathcal{O}$, and all $\mathcal{A}$-derivations π_1 and π_2 in ζ, when $a \longrightarrow^{\pi_1} b$ and $a \longrightarrow^{\pi_2} c$

there exists *d* in $\mathcal{O}$ and two $\mathcal{A}$-derivations π_3, π_4 in ζ such that :

1. $b \longrightarrow^{\pi_3} d$ and $c \longrightarrow^{\pi_4} d$;
2. $\pi_1; \pi_3$ and $\pi_2; \pi_4$ belong to ζ.

Example

$$\mathcal{A}_{lc} = \begin{tikzcd} a \arrow[r, bend left, "\phi_1"] \arrow[d, "\phi_2"'] & b \arrow[l, bend left, "\phi_3"] \arrow[d, "\phi_4"] \\ c & d \end{tikzcd}$$

Consider the following various strategies :

1. $\zeta_1 = \mathcal{D}(\mathcal{A}_{lc})$: $\mathcal{A}_{lc}$ is neither weakly nor strongly confluent under ζ_1 : $\pi_1 : a \longrightarrow^{\phi_1} b \longrightarrow^{\phi_4} d$ and $\pi_2 : a \longrightarrow^{\phi_2} c$.

Example

$$\mathcal{A}_{lc} = \begin{tikzcd} a \arrow[r, bend left, "\phi_1"] \arrow[d, "\phi_2"'] & b \arrow[l, bend left, "\phi_3"] \arrow[d, "\phi_4"] \\ c & d \end{tikzcd}$$

Consider the following various strategies :

1. $\zeta_1 = \mathcal{D}(\mathcal{A}_{lc})$: $\mathcal{A}_{lc}$ is neither weakly nor strongly confluent under ζ_1 : $\pi_1 : a \longrightarrow^{\phi_1} b \longrightarrow^{\phi_4} d$ and $\pi_2 : a \longrightarrow^{\phi_2} c$.
2. $\zeta_2 = \emptyset$: $\mathcal{A}_{lc}$ is trivially both weakly and strongly confluent under ζ_2.

Example

$$\mathcal{A}_{lc} = \begin{tikzcd} a \arrow[r, bend left, "\phi_1"] \arrow[d, "\phi_2"'] & b \arrow[l, bend left, "\phi_3"] \arrow[d, "\phi_4"] \\ c & d \end{tikzcd}$$

Consider the following various strategies :

1. $\zeta_1 = \mathcal{D}(\mathcal{A}_{lc})$: $\mathcal{A}_{lc}$ is neither weakly nor strongly confluent under ζ_1 : $\pi_1 : a \longrightarrow^{\phi_1} b \longrightarrow^{\phi_4} d$ and $\pi_2 : a \longrightarrow^{\phi_2} c$.
2. $\zeta_2 = \emptyset$: $\mathcal{A}_{lc}$ is trivially both weakly and strongly confluent under ζ_2.
3. $\zeta_3 = \{(\phi_1\phi_3)^*\phi_2\}$: $\mathcal{A}_{lc}$ is also weakly and strongly confluent under ζ_3.

Example

$$\mathcal{A}_{lc} = \begin{tikzcd} a \arrow[r, bend left, "\phi_1"] \arrow[d, "\phi_2"'] & b \arrow[l, bend left, "\phi_3"] \arrow[d, "\phi_4"] \\ c & d \end{tikzcd}$$

Consider the following various strategies :

1. $\zeta_1 = \mathcal{D}(\mathcal{A}_{lc})$: $\mathcal{A}_{lc}$ is neither weakly nor strongly confluent under ζ_1 : $\pi_1 : a \longrightarrow^{\phi_1} b \longrightarrow^{\phi_4} d$ and $\pi_2 : a \longrightarrow^{\phi_2} c$.
2. $\zeta_2 = \emptyset$: $\mathcal{A}_{lc}$ is trivially both weakly and strongly confluent under ζ_2.
3. $\zeta_3 = \{(\phi_1\phi_3)^*\phi_2\}$: $\mathcal{A}_{lc}$ is also weakly and strongly confluent under ζ_3.
4. For a different reason, this is also the case for $\zeta_4 = (\phi_1\phi_3)^\omega$ whose result is the empty set.

Example

Let $\mathcal{O} = \{a, b, c, d\}$ and reduction steps $\phi_1, \phi_2, \phi_3, \phi_4, \phi'_1, \phi'_2, \phi'_3, \phi'_4$.

Example

Let $\mathcal{O} = \{a, b, c, d\}$ and reduction steps $\phi_1, \phi_2, \phi_3, \phi_4, \phi'_1, \phi'_2, \phi'_3, \phi'_4$.

This ARS $\mathcal{A}$ is weakly and strongly confluent under the strategy $\zeta =$

$\{a \rightarrow^{\phi_1} b, a \rightarrow^{\phi_2} c, b \rightarrow^{\phi_3} d, c \rightarrow^{\phi_4} d, a \rightarrow^{\phi_1} b \rightarrow^{\phi_3} d, a \rightarrow^{\phi_2} c \rightarrow^{\phi_4} d\}$

Example

Let $\mathcal{O} = \{a, b, c, d\}$ and reduction steps $\phi_1, \phi_2, \phi_3, \phi_4, \phi'_1, \phi'_2, \phi'_3, \phi'_4$.

This ARS $\mathcal{A}$ is weakly and strongly confluent under the strategy $\zeta =$

$\{a \longrightarrow^{\phi_1} b, a \longrightarrow^{\phi_2} c, b \longrightarrow^{\phi_3} d, c \longrightarrow^{\phi_4} d, a \longrightarrow^{\phi_1} b \longrightarrow^{\phi_3} d, a \longrightarrow^{\phi_2} c \longrightarrow^{\phi_4} d\}$

but is not under

$$\zeta = \{a \longrightarrow^{\phi_1} b, a \longrightarrow^{\phi_2} c, b \longrightarrow^{\phi_3} d, c \longrightarrow^{\phi_4} d\}$$

Example

Let $\mathcal{O} = \{a, b, c, d\}$ and reduction steps $\phi_1, \phi_2, \phi_3, \phi_4, \phi'_1, \phi'_2, \phi'_3, \phi'_4$.

This ARS $\mathcal{A}$ is weakly and strongly confluent under the strategy $\zeta =$

$\{a \to^{\phi_1} b, a \to^{\phi_2} c, b \to^{\phi_3} d, c \to^{\phi_4} d, a \to^{\phi_1} b \to^{\phi_3} d, a \to^{\phi_2} c \to^{\phi_4} d\}$

but is not under

$$\zeta = \{a \to^{\phi_1} b, a \to^{\phi_2} c, b \to^{\phi_3} d, c \to^{\phi_4} d\}$$

$\mathcal{A}$ is weakly but not strongly confluent under the strategy $\zeta =$

$\{a \to^{\phi_1} b, a \to^{\phi_2} c, b \to^{\phi_3} d, c \to^{\phi_4} d, a \to^{\phi'_1} b \to^{\phi'_3} d, a \to^{\phi'_2} c \to^{\phi'_4} d\}$

Strategic rewriting

Given $\mathcal{A} = (\mathcal{O}_R, \mathcal{S}_R)$ generated by a rewrite system R, and a strategy ζ of $\mathcal{A}$,

- A strategic rewriting derivation (or rewriting derivation under strategy ζ) is an element of ζ.
- A strategic rewriting step under ζ is a rewriting step $t \rightarrow_R t'$ that occurs in a derivation of ζ.
  This is also denoted $t \rightarrow_\zeta t'$.

Strategy language

Elementary strategies : *Identity*, *Fail*, *R*, *Sequence*(s_1, s_2) or $s_2; s_1$

Strategy language

Elementary strategies : *Identity*, *Fail*, *R*, $Sequence(s_1, s_2)$ or $s_2; s_1$

- $Choice(s_1, s_2)$ selects the first strategy that does not fail ; it fails if both fail :
  $Choice(s_1, s_2)t = s_1 t$ if $s_1 t$ does not fail, else $s_2 t$.

Strategy language

Elementary strategies : *Identity*, *Fail*, *R*, *Sequence*(s_1, s_2) or $s_2; s_1$

- *Choice*(s_1, s_2) selects the first strategy that does not fail ; it fails if both fail :
  $Choice(s_1, s_2)t = s_1 t$ if $s_1 t$ does not fail, else $s_2 t$.
- On a term t, *All*(s) applies the strategy s on all immediate subterms :

$$All(s)f(t_1, ..., t_n) = f(t'_1, ..., t'_n)$$

if $st_1 = t'_1, ..., st_n = t'_n$; it fails if there exists i such that st_i fails.

Strategy language

Elementary strategies : *Identity*, *Fail*, *R*, *Sequence*(s_1, s_2) or $s_2; s_1$

- *Choice*(s_1, s_2) selects the first strategy that does not fail ; it fails if both fail :
  $Choice(s_1, s_2)t = s_1 t$ if $s_1 t$ does not fail, else $s_2 t$.
- On a term t, *All*(s) applies the strategy s on all immediate subterms :
  $$All(s)f(t_1, ..., t_n) = f(t'_1, ..., t'_n)$$
  if $st_1 = t'_1, ..., st_n = t'_n$; it fails if there exists i such that st_i fails.
- On a term t, *One*(s) applies the strategy s on the first immediate subterm where s does not fail :
  $$One(s)f(t_1, ..., t_n) = f(t_1, ..., t'_i, ..., t_n)$$
  if for all $j < i$, st_j fails, and $st_i = t'_i$; it fails if for all i, st_i fails.

Strategy language

Elementary strategies : *Identity*, *Fail*, *R*, $Sequence(s_1, s_2)$ or $s_2; s_1$

- $Choice(s_1, s_2)$ selects the first strategy that does not fail ; it fails if both fail :
  $Choice(s_1, s_2)t = s_1 t$ if $s_1 t$ does not fail, else $s_2 t$.
- On a term t, $All(s)$ applies the strategy s on all immediate subterms :
  $$All(s)f(t_1, ..., t_n) = f(t'_1, ..., t'_n)$$
  if $st_1 = t'_1, ..., st_n = t'_n$; it fails if there exists i such that st_i fails.
- On a term t, $One(s)$ applies the strategy s on the first immediate subterm where s does not fail :
  $$One(s)f(t_1, ..., t_n) = f(t_1, ..., t'_i, ..., t_n)$$
  if for all $j < i$, st_j fails, and $st_i = t'_i$; it fails if for all i, st_i fails.
- Fixpoint : $\mu x.s = s[x \leftarrow \mu x.s]$

Strategy language

$$
\begin{array}{lcl}
Try(s) & = & Choice(s, Identity) \\
Repeat(s) & = & \mu x.Choice(Sequence(s, x), Identity) \\
OnceBottomUp(s) & = & \mu x.Choice(One(x), s) \\
BottomUp(s) & = & \mu x.Sequence(All(x), s) \\
TopDown(s) & = & \mu x.Sequence(s, All(x)) \\
Innermost(s) & = & \mu x.Sequence(All(x), Try(Sequence(s, x)))
\end{array}
$$

Programming with Rules and Strategies - TOM