Map of the lectures

Saturday, 15 april

14:30 – 15:30 What is rewriting?

16:00 – 18:00 The rewriting calculus and ELAN

Sunday, 16 april

9:00 – 10:00 Deduction modulo

10:30 – 11:30 Application to higher-order logic

What is Rewriting?

Claude Kirchner
LORIA and INRIA
Nancy

Claude.Kirchner@loria.fr

Companion lecture notes: `www.loria.fr/~ckirchne/rsp.ps.gz`

fibonacci

$$
\begin{aligned}
fib(0) &\rightarrow 1\\
fib(1) &\rightarrow 1\\
fib(n) &\rightarrow fib(n-1) + fib(n-2)
\end{aligned}
$$

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

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

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

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

$$\ldots$$

/etc/sendmail.cf

```
#####  @(#)nullrelay.m4 8.19 (Berkeley) 5/19/1998  #####

#
#  This configuration applies only to relay-only hosts.  They send
#  all mail to a hub without consideration of the address syntax
#  or semantics, except for adding the hub qualification to the
#  addresses.
#
# This is based on a prototype done by Bryan Costales of ICSI.
#

######################################################################
######################################################################
#####
##### REWRITING RULES
#####
######################################################################
######################################################################
```

```
############################################
###  Ruleset 3 -- Name Canonicalization  ###
############################################
S3

# handle null input
R$@ $@ <@>

# strip group: syntax (not inside angle brackets!) and trailing semicolon
R$* $: $1 <@>mark addresses
R$* < $* > $* <@>$: $1 < $2 > $3 unmark <addr>
R$* :: $* <@>$: $1 :: $2 unmark node::addr
R:include: $* <@>$: :include: $1 unmark :include:...
R$* : $* <@>$: $2 strip colon if marked
.
.
.
.
.
.
.
```

```
.

##################################################
###  Ruleset 4 -- Final Output Post-rewriting  ###
##################################################
S4

R$* <@>$@ handle <> and list:;

# strip trailing dot off before passing to nullclient relay
R$* @ $+ . $1 @ $2
```

A simple game

The rules of the game:

●● → ○

○○ → ○

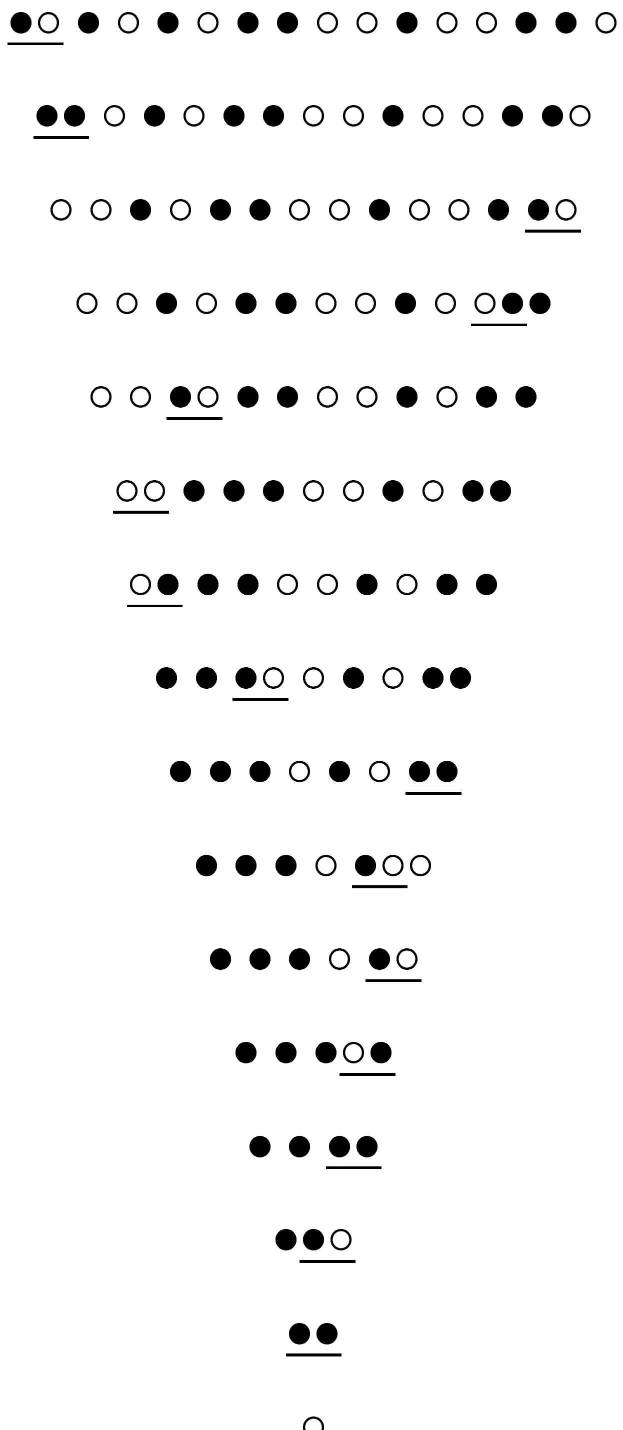
●○ → ●

○● → ●

A starting point:

● ○ ● ○ ● ○ ● ● ● ● ○ ○ ● ○ ○ ● ●○

Who wins? (i.e. put the last white)



Analysing the different cases

Disjoint redexes:

$$\cdots \underline{\otimes\otimes} \cdots \otimes\ \otimes \cdots$$

$$\cdots \otimes \cdots \underline{\otimes\otimes} \cdots$$

$$\cdots \otimes \cdots \otimes \cdots$$

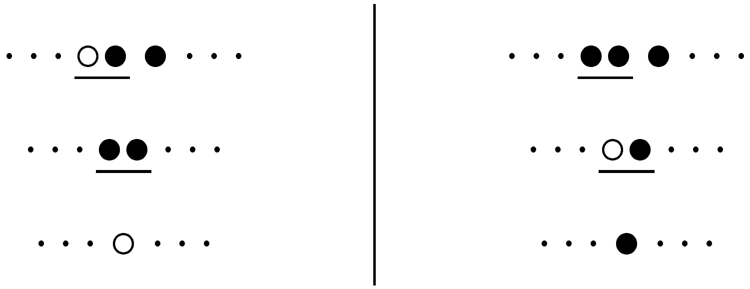
is the same as:

$$\cdots \otimes\ \otimes \cdots \underline{\otimes\otimes} \cdots$$

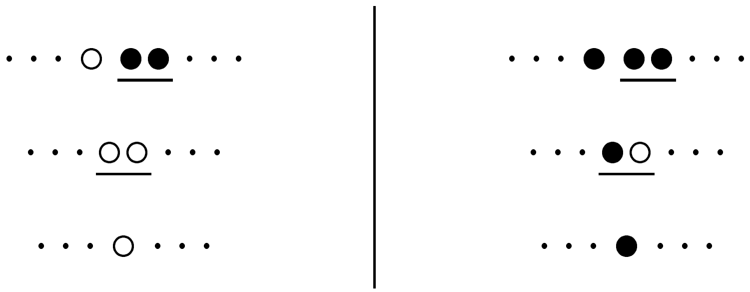
$$\cdots \underline{\otimes\otimes} \cdots \otimes \cdots$$

$$\cdots \otimes \cdots \otimes \cdots$$

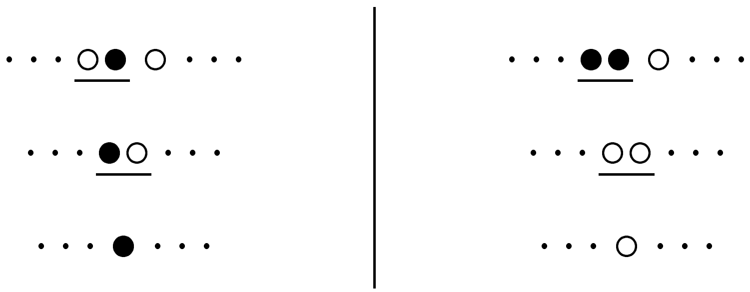
No disjoint redexes (central black):



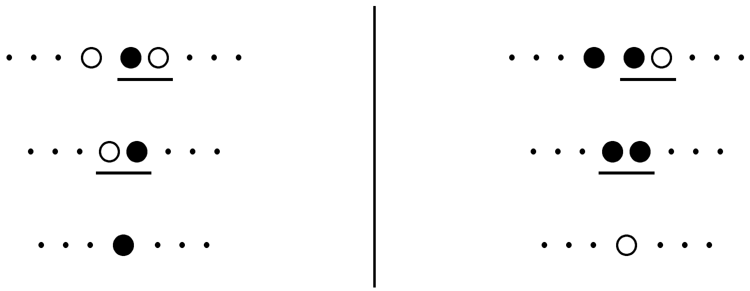
but



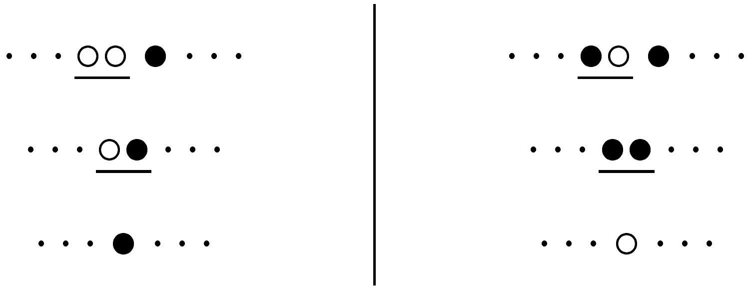
or



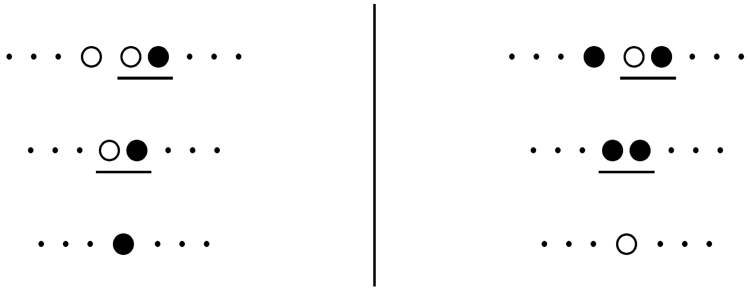
but



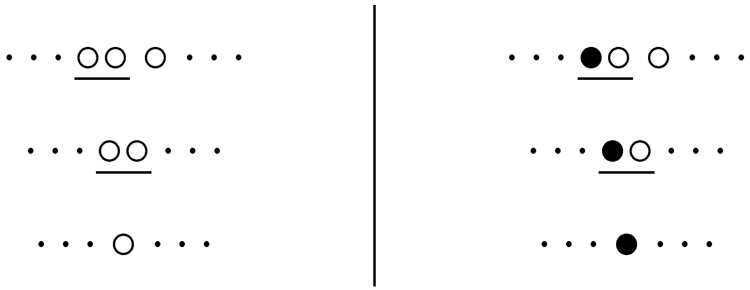
No disjoint redexes (central white):



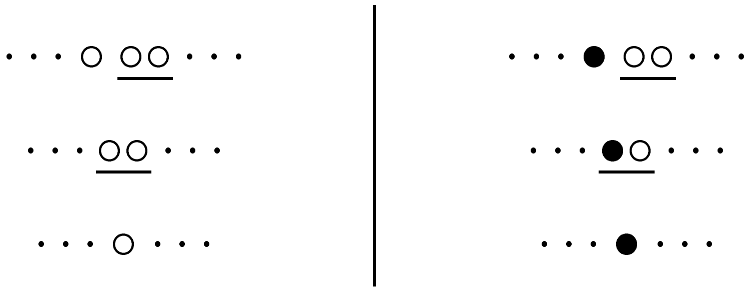
but



or



but



Thus in all the cases:

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

but what about:

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

Rewrite description of a sorting algorithm

```
sorts NeList List ;   subsorts Nat < NeList < List ;
operators
  nil : List ;
  @ @ : (List List) List      [associative id: nil] ;
  @ @ : (NeList List) NeList  [associative] ;
  hd @ : (NeList) Nat ;
  tl @ : (NeList) List ;
  sort @ : (List) List ;
end
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) => ...
```

Alternative Rings

$$
\begin{aligned}
0 + x &= x & 0 * x &= 0 \\
x * 0 &= 0 & i(x) + x &= 0 \\
i(x + y) &= i(x) + i(y) & i(i(x)) &= x \\
x * (y + z) &= (x * y) + (x * z) & (x + y) * z &= (x * z) + (y * z) \\
(x * y) * y &= x * (y * y) & (x * x) * y &= x * (x * y) \\
i(x) * y &= i(x * y) & x * i(y) &= i(x * y) \\
i(0) &= 0 & (x + y) + z &= x + (y + z) \\
x + y &= y + x & &
\end{aligned}
$$

Can we prove by rewriting the Moufang Identities?

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

(R.Moufang, 1933)

(S.Anantharaman and J.Hsiang, JAR 6, 1990)

An additive group G is defined by the set of equalities

$$\begin{aligned} x + e &= x \\ x + (y + z) &= (x + y) + z \\ x + i(x) &= e \end{aligned}$$

How to check that two elements of the group are the same?

$$i(x + y) =?= i(y) + i(x)$$

An equivalent deterministic term rewrite system

$$
\begin{array}{rcl}
x + e & \to & x \\
e + x & \to & x \\
x + (y + z) & \to & (x + y) + z \\
x + i(x) & \to & e \\
i(x) + x & \to & e \\
i(e) & \to & e \\
(y + i(x)) + x & \to & y \\
(y + x) + i(x) & \to & y \\
i(i(x)) & \to & x \\
i(x + y) & \to & i(y) + i(x)
\end{array}
$$

Then

$$
i(x+y) =?= i(y) + i(x) \quad \Leftrightarrow \quad i(x+y) \to \ldots \to \bullet \leftarrow \ldots \leftarrow i(y) + i(x)
$$

Contents

On what objects can rewriting act?

It can be defined on

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

We will "restrict" on terms

How does rewriting act?

Provided a matching condition is satisfied, a rewrite rule is applied

Example: $fib(n) \to fib(n-1) + fib(n-2)$ applies on $fib(3)$ since:

- the matching condition "$fib(n)$ could be equated to $fib(3)$" is true for $n = 3$

then

- we **replace** the $fib(3)$ by the right hand side of the rule where n is replaced by 3, i.e. by $fib(3-1) + fib(3-2)$

This could happens everywhere

In a term like

$$3 + fib(4)$$

a rewrite step may be applied inside, on the sub-term

$$fib(4)$$

This is denoted

$$3 + fib(4) \longrightarrow_{1}^{fib(n) \rightarrow fib(n-1) + fib(n-2)} 3 + fib(4-1) + fib(4-2)$$

Matching

Finding a substitution σ such that

$$\sigma(l) = t$$

is called the matching problem from l to t.

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

It induces a relation on terms called subsumption:

Term subsumption

$$s \leq t \Leftrightarrow \sigma(s) = t$$

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.

$\leq$ is a quasi-ordering on terms called *subsumption*.

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

Theorem: [Huet78]

Up to renaming, the subsumption ordering on terms is well-founded.

Notice that

$$s \le t \quad \not\Rightarrow \quad f(u,s) \le f(u,t)$$

since

$$x \le a \text{ but } f(x,x) \not\le f(x,a)$$

$$s \le t \quad \not\Rightarrow \quad \sigma(s) \le \sigma(t)$$

since

$$x \le a \text{ but } (x \mapsto b)x \not\le (x \mapsto b)a$$

Definition of a rewriting step

It relies on 4 notions:

- terms
- matching
- substitutions
- replacement

and, given a rule, it consists to:

– find a subterm

– that matches the left hand side of the rule

– and replace that subterm by the right hand side of the rule instanciated by the match

Formaly

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

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

This is denoted

$$t \longrightarrow^{p,l \to r} t'$$

Signature and terms

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

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

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

$\mathcal{F} = \{f, a\}$ with $arity(f) = 2$, $arity(a) = 0$, $x \in \mathcal{X}$:
$f(a, a)$ is ground,
$f(x, f(a, x))$ is not linear but
$f(x, f(y, z))$ is.

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

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

position	$\mapsto$	*symbol*
Λ	$\mapsto$	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

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

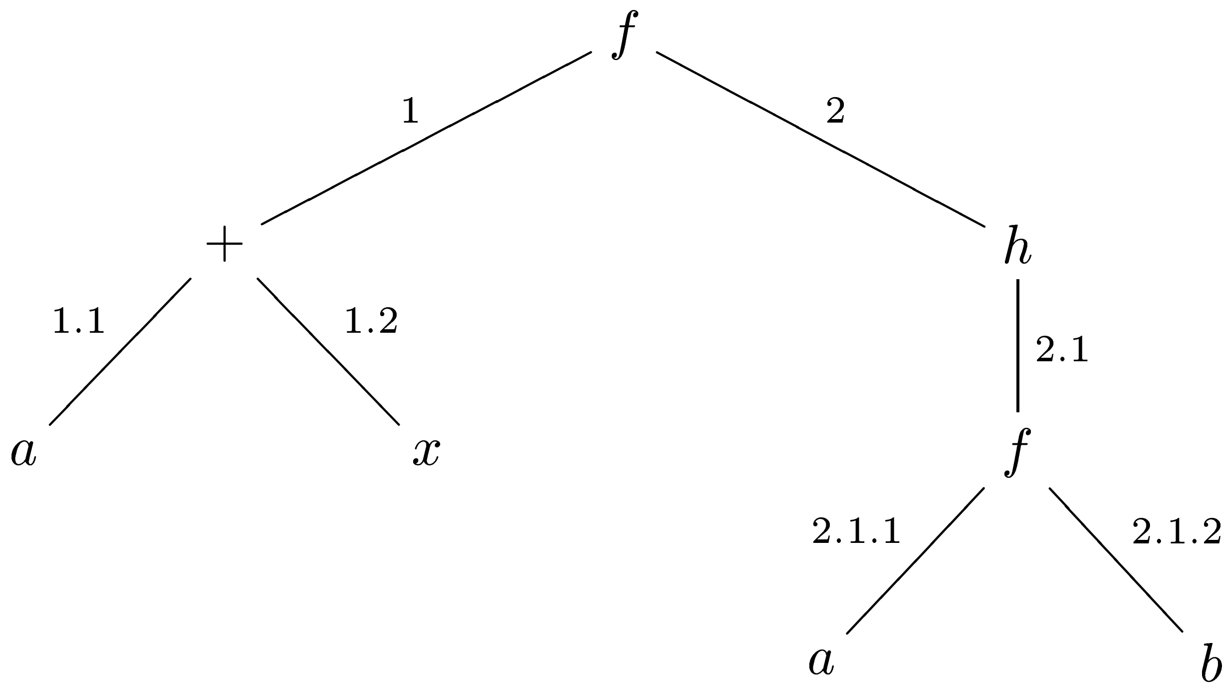
$t[s]_\omega$ denotes the term t with s as **subterm** at **position** (or **occurrence**) ω.

$t|_\omega$ denotes the subterm as 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.

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

Rewrite relation

A term rewrite system $\mathcal{R}$ (a set of rewrite rules) determines a relation on terms denoted $\longrightarrow_{\mathcal{R}}$:

$$t[l\sigma]_\omega \to_R t[r\sigma]_\omega$$

when

$$(l \to r) \in \mathcal{R}$$

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

Abstract rewrite systems

$\mathcal{T}$ a set and	$\longrightarrow$	a binary relation on $\mathcal{T}$
	$\xrightarrow{+}$	its transitive closure
	$\xrightarrow{*}$	its transitive reflexive closure
	$\longleftrightarrow$	its symetric closure

Let $\langle \mathcal{T}, \to \rangle$ be an abstract reduction system.

- 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.
  . $a \to a \quad a \to b$ is weakly terminating
- The relation $\to$ 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 $t = t'$.

Church Rosser (CR)

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

Confluent

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

Localy confluent (LC)

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

A typical result on abstract rewrite systems

[Newman 1942]

Provided the relation $\rightarrow$ is terminating

then

$\rightarrow$ is confluent iff it is locally confluent

Turing power

[Max Dauchet 1989]
A Turing machine can be simulated by the transitive closure of the relation generated by a single rewrite rule

This unique rewrite rule can further be left linear and regular!

... Termination of a rewrite relation

... Expressiveness ease

Conditional rules

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

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

The rule is applicable on a term t provided there exists a matching substitution σ of l on $t|_\omega$ such that

$$c\sigma \xrightarrow{*} true$$

.

Example

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

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

Generalized rules

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

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

where $true := c$ could be written **if** c.

The pattern p_i is often reduced to a variable x.

Applying such a rewrite rule

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

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

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

Rewriting modulo associativity commutativity

f is an AC symbol if it satisfies

$$\forall x, y, z,\ \ f(x, f(y, z)) = f(f(x, y), z) \quad \text{and} \quad \forall x, y,\ f(x, y) = f(y, x).$$

We denote $s =_{AC} t$ the equivalence modulo AC of two terms

For a term rewriting system $\mathcal{R}$ and a rule $l \to r$ in $\mathcal{R}$

$t \longrightarrow^{p, l \to r}_{\mathcal{R}, AC} t'$ iff

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

Example of an AC theory

$\cup$ an AC operator

$$\{i\} \cup s \to i$$

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

Five incomparable solutions ($i = 1, 2, 3, 4, 5$).

AC matching

A nice but (in general) expensive feature

Matching l on t modulo AC is exponential in the size of the terms and the set of matches can also be exponential.

$x + y + z \leq_{AC} a + b + c + d$

$(x \mapsto a, y \mapsto b, z \mapsto c + d) \quad (x \mapsto b, y \mapsto a, z \mapsto c + d)$

$(x \mapsto a, y \mapsto c, z \mapsto b + d) \quad (x \mapsto c, y \mapsto a, z \mapsto b + d)$

$(x \mapsto a, y \mapsto d, z \mapsto c + b) \quad (x \mapsto d, y \mapsto a, z \mapsto c + b)$

$\ldots$

Conditional AC rewriting

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

applied modulo AC on t,

rewrite to any $t' = t[\sigma(r)]_\omega$ such that

$$\sigma(l) =_{AC} t \quad \text{and} \quad \sigma(c)$$

Class rewrite systems

For an equational theory $\mathcal{E}$ and a rewrite system $\mathcal{R}$, two rewrite relations can be defined:

- $\longrightarrow_{\mathcal{R}\mathcal{E}}$ such that

$$t \longrightarrow_{\mathcal{R}\mathcal{E}} t' \text{ iff } t =_{\mathcal{E}} u \to_{\mathcal{R}} v =_{\mathcal{E}} t'$$

- $\longrightarrow_{\mathcal{R},\mathcal{E}}$ such that

$$t \longrightarrow^{p,l\to r}_{\mathcal{R},\mathcal{E}} t' \text{ iff } t_{|p} =_{\mathcal{E}} \sigma(l) \text{ and } t' = t[\sigma(r)]_p$$

Example

$$\mathcal{R} = \{x + x \to x\} \text{ and } \mathcal{E} = AC$$

compare the behavior of $\longrightarrow_{\mathcal{R}\mathcal{E}}$ and $\longrightarrow_{\mathcal{R},\mathcal{E}}$ on the term

$$a + (a + b)$$

More extentions

- Many-sorted and order-sorted rewriting
- Rewriting and lambda-calculus

Term rewriting is classicaly known for its use in computing normal form:

– in languages like ASF+SDF, OBJ, ...

– in saturation based procedures like Knuth-Bendix completion

– for deciding equalities

Generally as an operational tool to deal with equality.

`====>` the rewrite relation is used
`---->` the logical aspects of rewriting are –often– ignored

The rewrite relation

Given a term rewrite system $\mathcal{R}$, its generated rewrite relation ($\mathcal{R}_r$) is the (reflexive and) transitive closure of the elementary rewrite steps.

Rewriting theory

$$\mathcal{R} = (\mathcal{F}, E, N, L, R)$$

- $\mathcal{F}$ is a ranked alphabet of function symbols
- E is a set of $\mathcal{F}$-equalities (structural axioms)
- N is a set of confluent and terminating rewrite rules modulo E
- $A = E \cup N$ and $\langle t \rangle_A$: equational class of t modulo A
- L is a set of labels with arities
- R is a set of labelled conditional rewrite rules

$$\ell : l \Rightarrow r \text{ if } c$$

Rewriting Logic

[J. Meseguer TCS92]

Formulas are sequents of the form

$$\pi : \langle t \rangle_A \Rightarrow \langle t' \rangle_A$$

where π is a proof term, built on $\mathcal{F} \cup L \cup \{;\}$
recording the proof of the sequent.

Models are computation spaces (quotiented set of proof terms).

$$\mathcal{R} \ \vdash \ \pi : \langle t \rangle_A \Rightarrow \langle t' \rangle_A$$

if $\pi : \langle t \rangle_A \Rightarrow \langle t' \rangle_A$ can be obtained by finite application of the following deduction rules.

Reflexivity For any $t \in \mathcal{T}(\mathcal{F})$:

$$\langle t \rangle_A : \langle t \rangle_A \Rightarrow \langle t \rangle_A$$

Congruence For any $f \in \mathcal{F}$ with $arity(f) = n$:

$$\frac{\pi_1 : \langle t_1 \rangle_A \Rightarrow \langle t'_1 \rangle_A \quad \ldots \quad \pi_n : \langle t_n \rangle_A \Rightarrow \langle t'_n \rangle_A}{f(\pi_1, \ldots, \pi_n) : \langle f(t_1, \ldots, t_n) \rangle_A \Rightarrow \langle f(t'_1, \ldots, t'_n) \rangle_A}$$

Replacement For any $\ell : l(x_1, \ldots, x_n) \Rightarrow r(x_1, \ldots, x_n) \in R$,

$$\frac{\pi_1 : \langle t_1 \rangle_A \Rightarrow \langle t'_1 \rangle_A \quad \ldots \quad \pi_n : \langle t_n \rangle_A \Rightarrow \langle t'_n \rangle_A}{\ell(\pi_1, \ldots, \pi_n) : \langle l(t_1, \ldots, t_n) \rangle_A \Rightarrow \langle r(t'_1, \ldots, t'_n) \rangle_A}$$

Transitivity

$$\frac{\pi_1 : \langle t_1 \rangle_A \Rightarrow \langle t_2 \rangle_A \qquad \pi_2 : \langle t_2 \rangle_A \Rightarrow \langle t_3 \rangle_A}{\pi_1 ; \pi_2 \ : \ \langle t_1 \rangle_A \Rightarrow \langle t_3 \rangle_A}$$

$$\mathcal{S} = \quad \{\, nat \,\}$$

$$\mathcal{F} = \quad \left\{ \begin{array}{ll} zero & : \ \mapsto nat, \\ s & : nat \mapsto nat, \\ plus & : nat \times nat \mapsto nat \end{array} \right\}$$

$$A = \quad \{\, plus(X,Y) = plus(Y,X) \,\}$$

$$R = \quad \left\{ \begin{array}{lllll} \ell_0 & : & plus(zero, zero) & \rightarrow & zero, \\ \ell_1 & : & plus(s(X), Y) & \rightarrow & s(plus(X,Y)) \end{array} \right\}$$

How to get a proof of:

$$\langle plus(zero, s(zero)) \rangle_A \rightarrow \langle s(zero) \rangle_A$$

Replacement using the rewrite rule ℓ_1 and commutativity of $plus$:

$$\frac{\langle zero\rangle_A : \langle\langle zero\rangle_A\rangle_A \to \langle\langle zero\rangle_A\rangle_A}{l_1(\langle zero\rangle_A, \langle zero\rangle_A) : \langle\langle plus(zero, s(zero))\rangle_A\rangle_A \to \langle\langle s(plus(zero, zero))\rangle_A\rangle_A}$$

Replacement using the rewrite rule ℓ_0:

$$\frac{\langle zero\rangle_A : \langle\langle zero\rangle_A\rangle_A \to \langle\langle zero\rangle_A\rangle_A}{l_0 : \langle\langle plus(zero, zero)\rangle_A\rangle_A \to \langle\langle zero\rangle_A\rangle_A}$$

Congruence for the symbol s:

$$\frac{l_0 : \langle\langle plus(zero, zero)\rangle_A\rangle_A \to \langle\langle zero\rangle_A\rangle_A}{s(l_0) : \langle\langle s(plus(zero, zero))\rangle_A\rangle_A \to \langle\langle s(zero)\rangle_A\rangle_A}$$

Transitivity:

$$\frac{\begin{array}{l} l_1(\langle zero\rangle_A, \langle zero\rangle_A) : \langle\langle plus(zero, s(zero))\rangle_A\rangle_A \to \langle\langle s(plus(zero, zero))\rangle_A\rangle_A, \\ s(l_0) : \langle\langle s(plus(zero, zero))\rangle_A\rangle_A \to \langle\langle s(zero)\rangle_A\rangle_A \end{array}}{l_1(\langle zero\rangle_A, \langle zero\rangle_A); s(l_0) : \langle\langle plus(zero, s(zero))\rangle_A d\rangle_A \to \langle\langle s(zero)\rangle_A\rangle_A}$$

Model of rewriting logic

Computation space of the rewrite theory $\mathcal{R}$:
quotient set $\{\pi \mid \mathcal{R} \vdash \pi : \langle\langle t \rangle_A\rangle_A \to \langle\langle t' \rangle_A\rangle_A\}/(E \cup \mathcal{A}_{\mathcal{PT}(R)})$.

$\forall \pi_1, \pi_2, \pi_3 \in \mathcal{PT} \quad \pi_1; (\pi_2; \pi_3) = (\pi_1; \pi_2); \pi_3$ Associativity

$\forall \pi : \langle t \rangle_A \to \langle t' \rangle_A, \quad \pi; \langle t' \rangle_A = \pi,$ and $\langle t \rangle_A; \pi = \pi$ Local Identities

For all $f \in \mathcal{F}_n, n \in \mathbf{N}, \quad \forall \pi_1, \ldots, \pi_n, \pi'_1, \ldots, \pi'_n$:
$f(\pi_1; \pi'_1, \ldots, \pi_n; \pi'_n) = f(\pi_1, \ldots, \pi_n); f(\pi'_1, \ldots, \pi'_n)$ Independence

For all $f \in \mathcal{F}_n, n \in \mathbf{N}$:

$f(\langle t_1 \rangle_A, \ldots, \langle t_n \rangle_A) = \langle f(t_1, \ldots, t_n) \rangle_A$ Preservation of A

$\forall \ell : g \to d \in R, \forall \pi_1 : \langle t_1 \rangle_A \to \langle t'_1 \rangle_A, \ldots, \pi_n : \langle t_n \rangle_A \to \langle t'_n \rangle_A$
$\ell(\pi_1, \ldots, \pi_n) = \ell(\langle t_1 \rangle_A, \ldots, \langle t_n \rangle_A); d(\pi_1, \ldots, \pi_n)$ and
$\ell(\pi_1, \ldots, \pi_n) = g(\pi_1, \ldots, \pi_n); \ell(\langle t'_1 \rangle_A, \ldots, \langle t'_n \rangle_A)$
. Parallel Move Lemma

EQUATIONAL AXIOM:

any multiset of two terms $\{s, t\}$ denoted $s = t$.

Name	Definition
Associativity	$f(f(x,y),z) = f(x,f(y,z))$
Commutativity	$f(x,y) = f(y,x)$
Right Distributivity	$f(g(x,y),z) = g(f(x,z),f(y,z))$
Endomorphism	$h(x*y) = h(x)*h(y)$
Anti-endomorphism	$h(x*y) = h(y)*h(x)$
Homomorphism	$h(x*y) = h(x)+h(y)$
Idempotency	$f(x,x) = x$
Involution	$h(h(x)) = x$
Right Inverse	$x*i(x) = e$
Left Inverse	$i(x)*x = e$
Right Simplification	$f(g(x,y),h(y)) = x$
Right Unit	$x*e = x$
Left Unit	$e*x = x$
Right Commutativity	$f(f(x,y),z) = f(f(x,z),y)$
Lie brackets	$(x*y)*z = z*(y*x)$

DEDUCTION SYSTEM

Rewriting deduction + SYMETRIE

1. Reflexivity		$\twoheadrightarrow$	$(t = t)$
2. Symmetry	$(t = t')$	$\twoheadrightarrow$	$(t' = t)$
3. Transitivity	$(t = t'), (t' = t'')$	$\twoheadrightarrow$	$(t = t'')$
4. Congruence	$\{(t_i = t'_i) \| i = 1, \ldots, n\}$	$\twoheadrightarrow$	$(f(t_1, \ldots, t_n) = f(t'_1, \ldots, t'_n))$
			if $f \in \mathcal{F}_n$
5. Substitutivity	$(t_1 = t_2)$	$\twoheadrightarrow$	$(\sigma(t_1) = \sigma(t_2))$
			if $\sigma \in Subst$

Equational Theory

For a given set of equational axioms E the equational theory of E ($TH(E)$), is the set of equalities that can be obtained by applying the previous inference rules starting from E.

Notation: $E \vdash s = t$ if $s = t \in TH(E)$

What is the equational theory of $\{a = b\}$?

Replacement of equal by equal

Given E:

$$
\begin{array}{lll}
s \longleftrightarrow_E t & \text{if} & s_{|w} = \sigma(l) \\
 & & \text{and} \\
 & & t = s[\sigma(r)]_w \\
 & \text{where} & w \in Dom(s) \\
 & & l = r \in E
\end{array}
$$

The provability relation $\stackrel{*}{\longleftrightarrow}_E$ is the

- reflexive
- transitive

closure of $\longleftrightarrow_E$.

$\stackrel{*}{\longleftrightarrow}_E$ is a congruence and $\mathcal{T}(\mathcal{F},\mathcal{X})/_{\stackrel{*}{\longleftrightarrow}_E}$ is simply denoted $\mathcal{T}(\mathcal{F},\mathcal{X})/_E$.

If $E = \{f(x,x) = x\}$ then

- $a \longleftrightarrow_E f(a,a) \longleftrightarrow_E f(f(a,a),a)$
- $g(a,b) \longleftrightarrow_E g(a,f(b,b)) \longleftrightarrow_E g(a,f(f(b,b),b))$

Theorem (Birkhoff) (1935)

$$E \vdash s = t \qquad \Longleftrightarrow \qquad s \stackrel{*}{\longleftrightarrow}_E t$$

Prove that $T(F,X)/_{\stackrel{*}{\longleftrightarrow}_E}$ is a F-algebra.

Birkhoff Theorems

Theorem: a class $\mathcal{C}$ of $\mathcal{F}$-algebras is the class of models of E iff it is a variety.

Theorem:

$$\mathcal{M}od(E) \models s = t$$

iff

$$\mathcal{T}(\mathcal{F}, \mathcal{X})/_E \models s = t$$

iff

$$E \vdash s = t$$

iff

$$s \xleftrightarrow{*}_E t$$

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

Rule system

Deterministic
Decidable
Efficiency

Non Deterministic
Decidable
Finite Search

Non Deterministic
Undecidable
Infinite Search

Computing

Solving

Proving

Programming with Constraints

Theorem Proving with Constraints

Deduction Modulo

Rewriting for computing

$$
\begin{array}{llll}
log(x \times y) & \rightarrow & log(x) + log(y) & \textbf{if} \quad x \geq 0 \ \wedge \ y \geq 0 \\
\hline
\end{array}
$$

$$
\begin{array}{llll}
|x| & \rightarrow & x & \textbf{if} \quad x \geq 0 \\
|x| & \rightarrow & -x & \textbf{if} \quad x < 0 \\
\hline
\end{array}
$$

$$
\begin{array}{llll}
fib(0) & \rightarrow & 1 & \\
fib(1) & \rightarrow & 1 & \\
fib(n) & \rightarrow & fib(n-1) + fib(n-2) & \textbf{if} \quad n > 1
\end{array}
$$

Rewriting for solving

Matching

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

Example: Finding a match

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

$$\Rightarrow_{\mathsf{Decomposition}}\ x \ll^? 1 \ \ \& \ \ y*y = 4*3$$

$$\Rightarrow_{\mathsf{Decomposition}}\ x \ll^? 1 \ \ \& \ \ y \ll^? 4 \ \ \& \ \ y \ll^? 3$$

$$\Rightarrow_{\mathsf{MergingClash}}\ \mathbf{F}$$

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

$$\Rightarrow_{\mathsf{Decomposition}}\ x \ll^? 1 \ \ \& \ \ y*3 = 4*3$$

$$\Rightarrow_{\mathsf{Decomposition}}\ x \ll^? 1 \ \ \& \ \ y \ll^? 4 \ \ \& \ \ 3 \ll^? 3$$

$$\Rightarrow_{\mathsf{Delete}}\ x \ll^? 1 \ \ \& \ \ y \ll^? 4$$

Equation solving

Does it exists x, y, z such that:

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

There is an infinity of solutions

but a most general one

$$x = y = z$$

===> syntactic unification

The rules for syntactic unification

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

Check*		$P \ \& \ \ x_1 =^? s_1[x_2] \ \& \ \ldots$	
		$\ldots \ \& \ x_n =^? s_n[x_1]$	
	$\mapsto\!\!\!\twoheadrightarrow$	$\mathbf{F}$	if $s_i \notin \mathcal{X}$ *for some* $i \in [1..n]$
Merge		$P \ \& \ x =^? s \ \& \ x =^? t$	
	$\mapsto\!\!\!\twoheadrightarrow$	$P \ \& \ x =^? s \ \& \ s =^? t$	if $0 < \lvert s \rvert \leq \lvert t \rvert$
Check		$P \ \& \ x =^? s$	
	$\mapsto\!\!\!\twoheadrightarrow$	$\mathbf{F}$	if $x \in \mathcal{V}ar(s)$ *and* $s \notin \mathcal{X}$
Eliminate		$P \ \& \ x =^? s$	
	$\mapsto\!\!\!\twoheadrightarrow$	$\{x \mapsto s\}P \ \& \ x =^? s$	if $x \notin \mathcal{V}ar(s), s \notin \mathcal{X}, x \in \mathcal{V}ar(P)$

Example

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

$$\Rightarrow_{\mathsf{decompose}}\ x = y \quad \& \quad z*y = x*z$$

$$\Rightarrow_{\mathsf{decompose}}\ x = y \quad \& \quad z = x \quad \& \quad y = z$$

$$\Rightarrow_{\mathsf{coalesce}}\ y = z \quad \& \quad x = z \quad \& \quad z = x$$

$$\Rightarrow_{\mathsf{coalesce}}\ z = x \quad \& \quad y = x \quad \& \quad x = x$$

$$\Rightarrow_{\mathsf{delete}}\ z = x \quad \& \quad y = x$$

Rewriting for Proving

Confluence

When does a computation by rewriting deliver a unique result?

The rewrite system R is **confluent** when:

$$ {}_R\overset{*}{\longleftarrow} \circ \overset{*}{\longrightarrow}_R \subseteq \overset{*}{\longrightarrow}_R \circ {}_R\overset{*}{\longleftarrow} $$

Confluence implies the unicity of normal forms

This is equivalent to:

The Church-Rosser property of R: all equational theorem admits a rewrite proof i.e.:

$$ t =_R t' \text{ iff } t \overset{*}{\longrightarrow}_R \circ {}_R\overset{*}{\longleftarrow} t' $$

- Confluent is indecidable in general. It is decidable for finite and terminating rewrite systems:
- Under the termination of the rewrite relation hypothesis, confluence is equivalent to the confluence of critical pairs.

Critival Pairs

Superposition

$$l_1 \to r_1 \qquad\qquad l_2[u] \to r_2$$

$$l_2[r_1]\sigma = r_2\sigma$$

u is a non-variable sub-term of l_2

σ is the $mgu(u, l_1)$

All critical pair should satisfy:

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

The saturation rules

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

The associated proof transformations

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

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.

Moreover $\overset{*}{\longleftrightarrow}_{P_0 \cup R_0}$ and $\overset{*}{\longleftrightarrow}_{R_\infty}$ coincides.

Some successes of automated deduction

- **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**
  In a Boolean algebra

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

  implies

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

  (McCune, 1996)

Rewriting for verifying

Needham-Schroeder public-key protocol

- proposed in '78 by Needham and Schroeder
- proved insecure in '95 by G. Lowe
- Needham-Schroeder public-key protocol has been already analyzed using several methodologies
  - model-checkers - FDR, Murφ
  - theorem proving - NRL
  - rewriting - Maude

The protocol

The Needham-Schroeder public-key protocol aims to establish a mutual authentication between an initiator and a responder that communicate via an insecure network.

	Initiator	Responder	Net
1. $A \to B$: $\{N_A, A\}_{K(B)}$	A^{SLEEP}	B^{SLEEP}	$\emptyset$
	A^{WAIT}	B^{SLEEP}	$\{N_A, A\}_{K(B)}$
2. $B \to A$: $\{N_A, N_B\}_{K(A)}$	A^{WAIT}	$B^{\{N_A, A\}_{K(B)}}$	$\emptyset$
	A^{WAIT}	B^{WAIT}	$\{N_A, N_B\}_{K(A)}$
3. $A \to B$: $\{N_B\}_{K(B)}$	$A^{\{N_A, N_B\}_{K(A)}}$	B^{WAIT}	$\emptyset$
	A^{COMMIT}	B^{WAIT}	$\{N_B\}_{K(B)}$
4. N_B *is the session key*	A^{COMMIT}	B^{COMMIT}	$\emptyset$

Methodology

Detect possible attacks of the protocol by the symbolic execution of all the possible states.

Undeterministic strategies are needed to ease the description of the search space.

More details after introducing ELAN.

Modelizing with rewrite relation

Confluence of $\mathcal{R}$

Termination of $\mathcal{R}$

Compute normal forms

Find normal forms optimaly, when possible

Modularity properties of rewrite systems

For example for termination

$$f(a, b, x) \quad \to \quad f(x, x, x)$$

is terminating and well as

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

Is the union terminating?

Too bad ...

$$\begin{aligned} f(a,b,x) &\to f(x,x,x) \\ g(x,y) &\to x \\ g(x,y) &\to y, \end{aligned}$$

We have the cycle:

$$\begin{tikzcd} f(g(a,b),g(a,b),g(a,b)) \arrow[r] & f(a,g(a,b),g(a,b)) \arrow[r] & f(a,b,g(a,b)) \arrow[ll, bend right] \end{tikzcd}$$

[Toyama 1986]

What about confluence?

[Toyama 1987] proved that

The disjoint union of two confluent rewrite systems is confluent

What happens if the two rewrite systems share symbols?

1- Introduction of rewriting

2- Definition of a rewrite relation

3- Abstract rewrite systems

4- Rewrite relation expressiveness

5- Rewrite: the relation and the logic points of view

6- Rewriting and equating: a BIG GIB difference

7- Rewriting for Computing, Solving, Proving

8- Typical problems of the field

9- To know more

- The rewriting page:

  `http://www.loria.fr/~vigneron/RewritingHP/index.html`

- The IFIP WG 1.6 working group on rewriting and applications:

- Who is who in rewriting:

  `http://www.math.unipd.it/~max/rew.whoswho.html`

- The RTA conferences

- Books:

  Rewriting and all that

  [Baader Nipkow 1998] (Cambridge University Press)

  Rewriting, Solving, Proving

  [Kirchner Kirchner 1994-2000] (`www.loria.fr/~ckirchne/rsp.ps.gz`)

- Surveyssss

Is that all?.........

Of course not

The main difficulty in using rewrite relations as a modeling tool is that there is no control on it.

We need more that a rewrite relation or a rewrite logic, we need a rewrite calculus giving access to the control of the rewrite application.

More on that topics soon.