

Rule-based computation and deduction

Hélène Kirchner and Pierre-Etienne Moreau
LORIA – CNRS – INRIA
Nancy, France

Rules are ubiquitous in Computer Science

Production rules

Program transformation rules

Grammar rules

Transition rules

Logic programming rules

Constraint manipulation rules

Inference rules

Type checking rules

Functional programming rules ...

and used in many contexts

- Condition-action rules in expert systems: well-suited for **supervision** applications or **production rules** systems.
- Computation of normal forms.
Decision procedures for equational theories.
- Forward chaining rules and simplification rules in **constraint solvers** meta-programming capabilities useful to eliminate redundancies or detect inconsistencies in a constraint store.
- Rule-based systems used to prototype **verification** tools such as **model-checking** algorithms.

Fibonacci numbers

$$fib(0) = 1$$

$$fib(1) = 1$$

$$fib(n) = fib(n - 1) + fib(n - 2)$$

$$fib(0) \rightarrow 1$$

$$fib(1) \rightarrow 1$$

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

$$\begin{aligned} fib(3) &\rightarrow fib(2) + fib(1) \\ fib(2) + fib(1) &\rightarrow fib(2) + 1 \\ fib(2) + 1 &\rightarrow fib(1) + fib(0) + 1 \\ fib(1) + fib(0) + 1 &\rightarrow \dots \end{aligned}$$

sendmail under Unix

[sendmail/doc/intro] "Address rewriting rules"

The heart of address parsing in sendmail is a set of rewriting rules. These are an ordered list of pattern-replacement rules, (somewhat like a production system, except that order is critical), which are applied to each address.

The address is rewritten textually until it is either rewritten into a special canonical form (i.e., a (mailer, host, user) 3-tuple, such as arpanet, usc-isif, postel representing the address "postel@usc-isif"), or it falls off the end. When a pattern matches, the rule is reapplied until it fails.

The configuration file also supports the editing of addresses into different formats. For example, an address of the form:

ucsfcg!tef might be mapped into: **tef@ucsfcg.UUCP** to conform to the domain syntax. Translations can also be done in the other direction.

A simple game

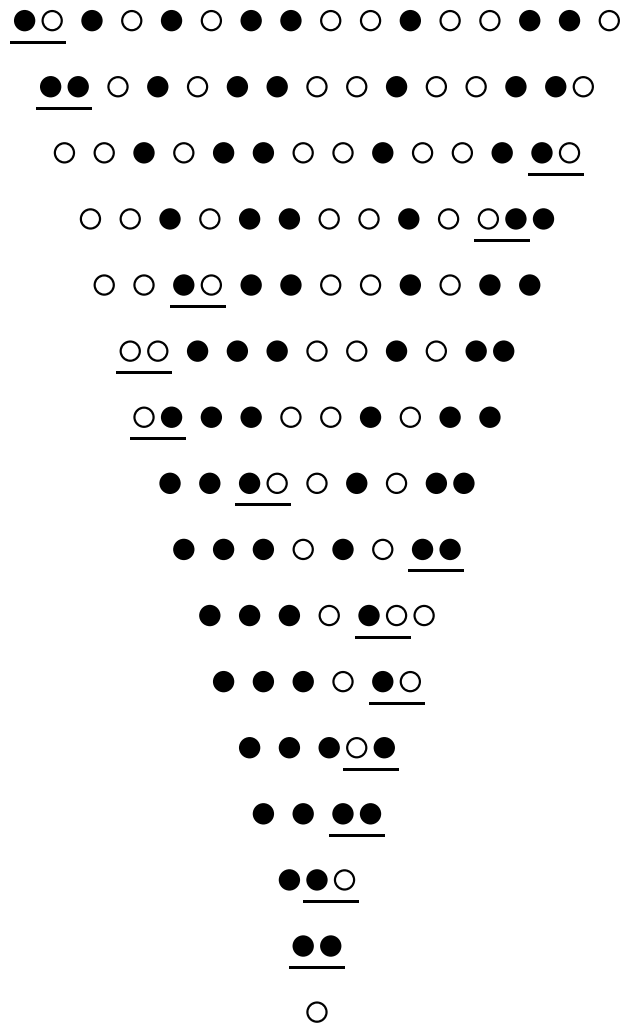
The rules of the game:

●● → ○
○○ → ○
●○ → ●
○● → ●

A starting point:

● ○ ● ○ ● ○ ● ● ● ● ○ ○ ● ○ ○ ● ● ○

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



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

Alternative Rings

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

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)

Two main kinds of rules

Computation rules

$2 + 3$

fib(33)

sort(data-base)

Compute as fast as possible the unique result

Deduction rules

Sequent calculus

Resolution method

Solve($2x + 4y - 3z - u = 0$ in Nat)

The deduction has to be guided

Control

In all applications, it is crucial to be able to express control in a declarative way. This allows **programming search procedures** and specifying how to **explore the search space**.

Search plans

 Action plans

 Tactics

 Lazy evaluation

 Innermost/Outermost reduction

 User interaction

 and more...

Contents/Ojectives of the lecture

- Definition and properties of rewriting
- Logic and calculus for rewriting
- Rule-based programming in ELAN
- Compilation or how to get efficiency?
- Applications and future developments

Rule-based computation and deduction

What is Rewriting?

Definition and Properties

Contents

1- Definition of a rewrite relation

- 2- Abstract rewrite systems
- 3- Rewriting and equating: a BIG GIB difference
- 4- Rewrite relation expressiveness
- 5- To know more

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: $fib(n) \rightarrow fib(n - 1) + fib(n - 2)$ applies on $fib(3)$:

- the matching condition “ $fib(n)$ can be equated to $fib(3)$ ” is true for $n = 3$

then

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

This can happen everywhere

In a term like

$$3 + fib(4)$$

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

$$fib(4)$$

This is denoted

$$3 + fib(4) \longrightarrow 3 + fib(4 - 1) + fib(4 - 2)$$

with $fib(n) \rightarrow fib(n - 1) + fib(n - 2)$.

Definition of a rewriting step

It relies on 4 notions:

- (sub-)terms
- matching
- substitutions
- replacement

Given a rule, it consists of:

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

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:

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

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

Examples

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

$f(a, a)$ is ground,

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

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

Terms as mappings

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

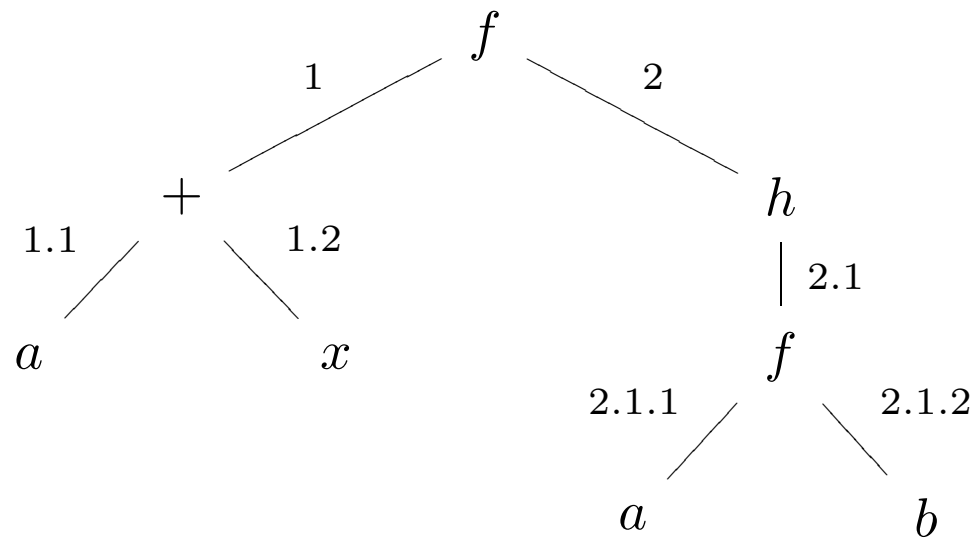
$t = f(a + x, h(f(a, b)))$ is represented by:

<i>position</i>	\mapsto	<i>symbol</i>
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

$$\text{Dom}(t) = \{\Lambda, 1, 1.1, 1.2, 2, 2.1, 2.1.1, 2.1.2\}$$

Terms as trees

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



Notations

$t[s]_p$ denotes the term t with s as **subterm** at **position** (or **occurrence**) p .

$t|_p$ denotes the subterm at position p .

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

$|t|$ 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 .

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

A **substitution** is a mapping from variables to terms

$$\sigma = (x \mapsto t, \dots, y \mapsto u)$$

Matching

Finding a substitution σ such that

$$\sigma(l) = t$$

is called the matching problem from l to t .

Also denoted $l\sigma = t$.

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

This 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

$$s \leq t \not\Rightarrow f(u, s) \leq f(u, t)$$

since

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

$$s \leq t \not\Rightarrow \sigma(s) \leq \sigma(t)$$

since

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

Formal definition of a rewriting step

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

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

This is denoted

$$t[l\sigma]_p \longrightarrow^{p, l \rightarrow r} t[r\sigma]_p$$

Rewrite relation

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

$$t[l\sigma]_p \longrightarrow_{\mathcal{R}} t[r\sigma]_p$$

when

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

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

Contents

1- Definition of a rewrite relation

2- Abstract rewrite systems

3- Rewriting and equating: a BIG GIB difference

4- Rewrite relation expressiveness

5- To know more

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 symmetric closure
 \longleftrightarrow^* its transitive reflexive symmetric closure

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

The rewrite relation

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

- 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.
.
 $a \rightarrow a$ $a \rightarrow b$ is weakly terminating
- The relation \rightarrow has the *unique normal form property* if for any $t, t' \in \mathcal{T}$, $t \xrightarrow{*} t'$ and t, t' are normal forms imply $t = t'$.

Back to the simple game

The rules of the game:

●● → ○
○○ → ○
●○ → ●
○● → ●

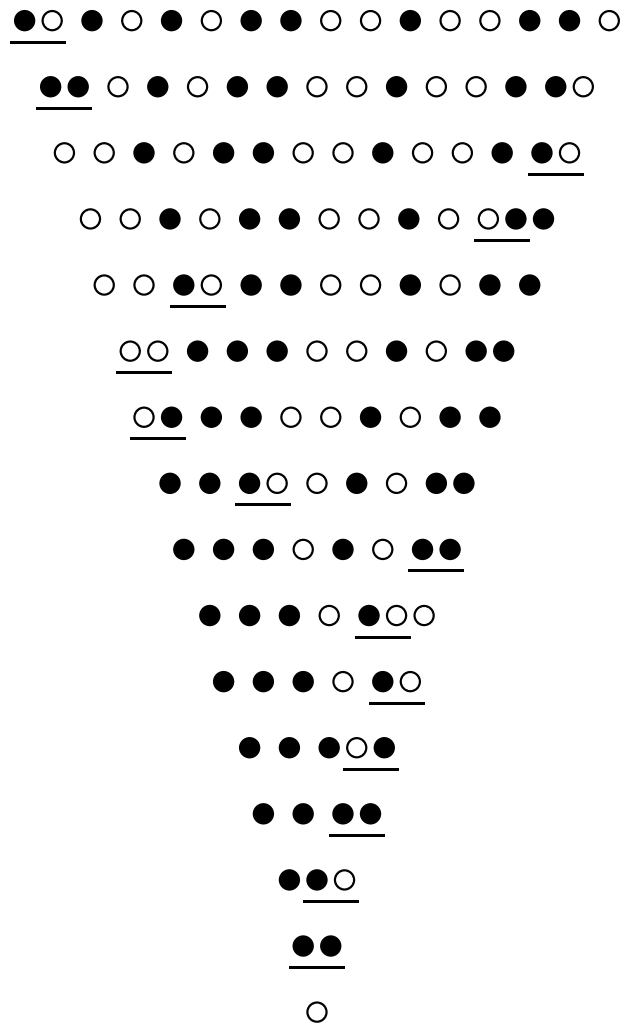
A starting point:

● ○ ● ○ ● ○ ● ● ● ● ○ ○ ● ○ ○ ● ● ○

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

Is there a winner?

Is there a strategy to win?



What can be said?

- The game terminates (existence of a normal form):

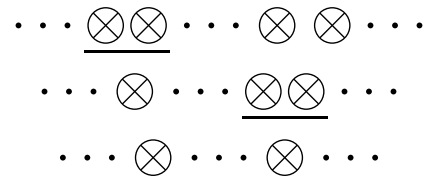
At each step, the number of bullets strictly decreases.

- The result is deterministic (unicity of the normal form):

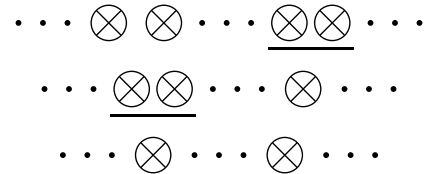
i.e. independent of the rules which are applied.

Analysing the different cases

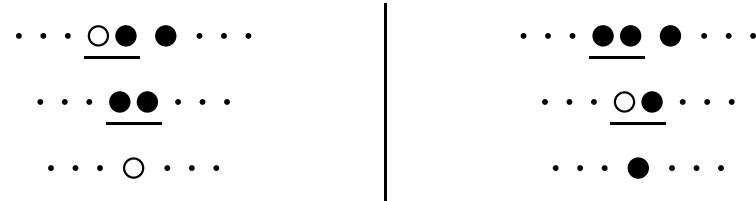
Disjoint redexes:



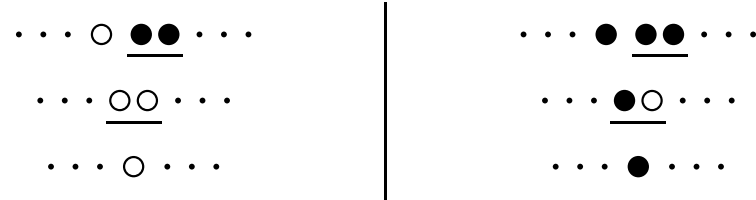
is the same as:



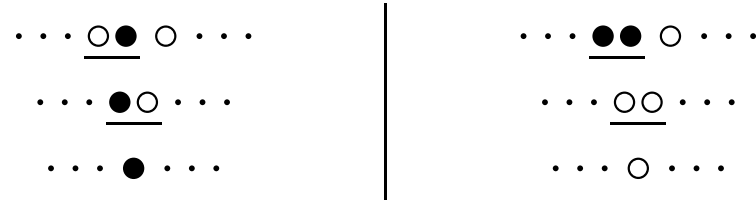
Non disjoint redexes (central black):



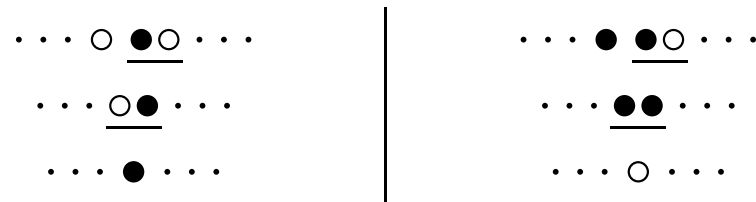
but



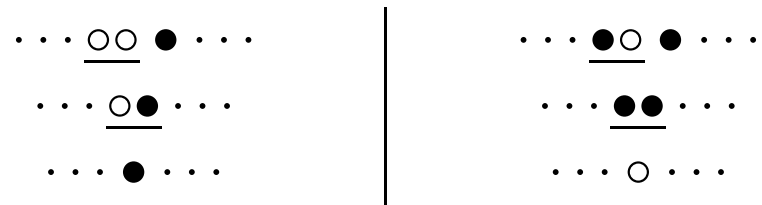
or



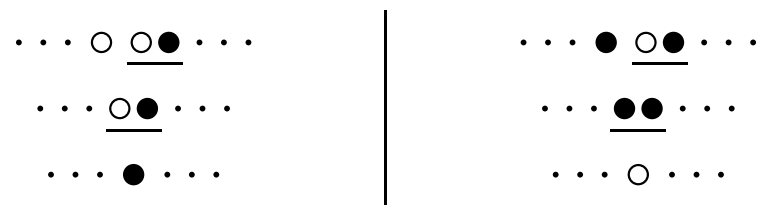
but



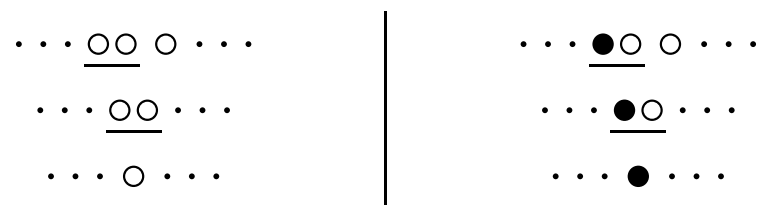
Non disjoint redexes (central white):



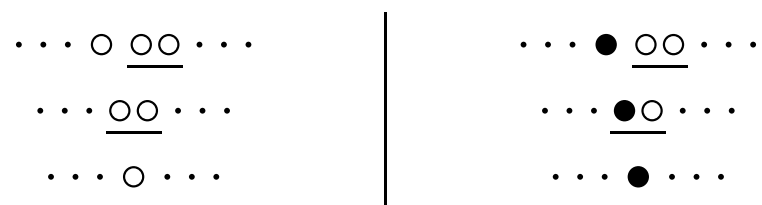
but



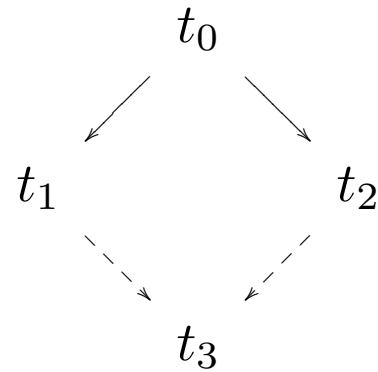
or



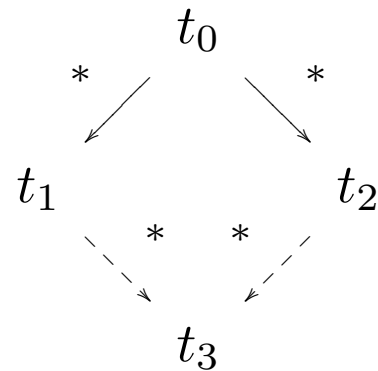
but



Thus in all the cases:

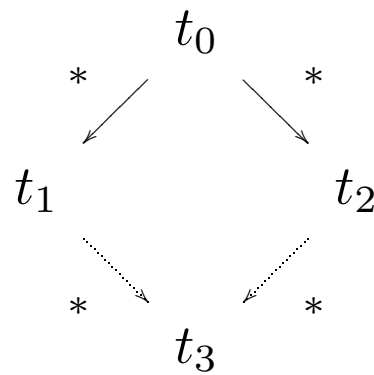


but can we deduce something more globally?

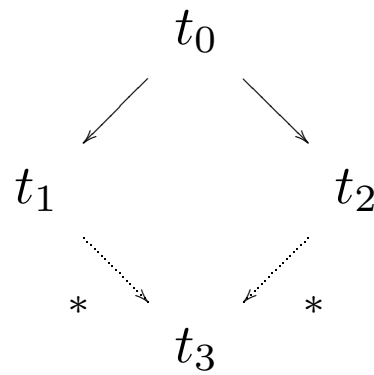


Confluence

Confluent



Locally confluent (LC)



A typical result on abstract rewrite systems

Theorem[Newman 1942]

Provided the relation \rightarrow is terminating, then
 \rightarrow is confluent iff it is locally confluent

Confluence implies the unicity of normal forms

- Confluent is undecidable 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.

Critical Pairs

Superposition

$$\begin{array}{l} l_1 \longrightarrow r_1 \qquad l_2[u] \longrightarrow r_2 \\ l_2[r_1]\sigma = r_2\sigma \end{array}$$

u is a non-variable sub-term of l_2
 σ is the $mgu(u, l_1)$ (i.e. $u\sigma = l_1\sigma$)

All critical pair should satisfy:

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

Contents

1- Definition of a rewrite relation

2- Abstract rewrite systems

3- Rewriting and equating: a BIG GIB difference

4- Rewrite relation expressiveness

5- To know more

Deduction system for Rewriting

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

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

$$\frac{t_1 \rightarrow t'_1 \quad \dots \quad t_n \rightarrow t'_n}{f(t_1, \dots, t_n) \rightarrow f(t'_1, \dots, t'_n)}$$

Substitutivity For any substitution σ , $t, t' \in \mathcal{T}(\mathcal{F})$:

$$\frac{t \rightarrow t'}{\sigma(t) \rightarrow \sigma(t')}$$

Transitivity For any $t_1, t_2, t_3 \in \mathcal{T}(\mathcal{F})$:

$$\frac{t_1 \rightarrow t_2 \quad t_2 \rightarrow t_3}{t_1 \rightarrow t_3}$$

Deduction system for Equality

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

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

$$\frac{t_1 = t'_1 \quad \dots \quad t_n = t'_n}{f(t_1, \dots, t_n) = f(t'_1, \dots, t'_n)}$$

Substitutivity For any substitution σ , $t, t' \in \mathcal{T}(\mathcal{F})$:

$$\frac{t = t'}{\sigma(t) = \sigma(t')}$$

Transitivity For any $t_1, t_2, t_3 \in \mathcal{T}(\mathcal{F})$:

$$\frac{t_1 = t_2 \quad t_2 = t_3}{t_1 = t_3}$$

Symetry For any $t, t' \in \mathcal{T}(\mathcal{F})$:

$$\frac{t = t'}{t' = t}$$

Equational axioms

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

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 inference rules for equality deduction, starting from E .

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

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

Replacement of equal by equal

Given E :

$$s \longleftrightarrow_E t \quad \text{if} \quad \begin{array}{l} s|_p = \sigma(l) \\ \text{and} \\ t = s[\sigma(r)]_p \end{array}$$

where

$$\begin{array}{l} p \in \text{Dom}(s) \\ l = r \in E \end{array}$$

\longleftrightarrow_E is symmetric.

The relation $\overset{*}{\longleftrightarrow}_E$ is the

- reflexive
- transitive

closure of \longleftrightarrow_E .

Provability relation

$\overset{*}{\longleftrightarrow}_E$ is closed under substitution.

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

- $x \overset{*}{\longleftrightarrow}_E f(x, x)$
- $a \overset{*}{\longleftrightarrow}_E f(a, a)$

$\overset{*}{\longleftrightarrow}_E$ is a congruence.

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

- $a \overset{*}{\longleftrightarrow}_E f(a, a) \overset{*}{\longleftrightarrow}_E f(f(a, a), a)$
- $g(a, b) \overset{*}{\longleftrightarrow}_E g(a, f(b, b)) \overset{*}{\longleftrightarrow}_E g(a, f(f(b, b), b))$

Theorem (Birkhoff) (1935)

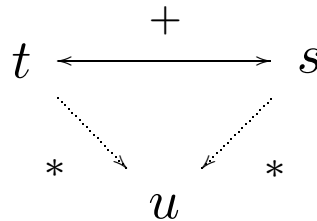
$$E \vdash s = t \quad \iff \quad s \overset{*}{\longleftrightarrow}_E t$$

How to prove equational theorems?

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

$$R^{\leftarrow*} \circ \longrightarrow_R \subseteq \longrightarrow_R \circ R^{\leftarrow*}$$

This is equivalent to the **Church-Rosser (CR)** property of R :



When R is confluent (or CR), any equational theorem admits a rewrite proof:

$$t =_R t' \text{ iff } t \longrightarrow_R \circ R^{\leftarrow*} t'$$

An additive group G is defined by the set of equalities

$$x + e = x$$

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

$$x + i(x) = e$$

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

$$x + e \rightarrow x$$

$$e + x \rightarrow x$$

$$x + (y + z) \rightarrow (x + y) + z$$

$$x + i(x) \rightarrow e$$

$$i(x) + x \rightarrow e$$

$$i(e) \rightarrow e$$

$$(y + i(x)) + x \rightarrow y$$

$$(y + x) + i(x) \rightarrow y$$

$$i(i(x)) \rightarrow x$$

$$i(x + y) \rightarrow i(y) + i(x)$$

To prove an equational theorem

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

$$\Leftrightarrow$$

$$i(x + y) \rightarrow i(y) + i(x)$$

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

$$\Leftrightarrow$$

$$e + y \rightarrow y \leftarrow y + e \leftarrow y + (i(x) + x) \leftarrow (y + i(x)) + x$$

As a consequence

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

- in languages like ASF+SDF, OBJ, ...
- in theorem provers, for simplifying formulas
- 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

Contents

- 1- Definition of a rewrite relation
- 2- Abstract rewrite systems
- 3- Rewriting and equating: a BIG GIB difference

4- Rewrite relation expressiveness

- 5- To know more

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

Conditional rules

$$l \longrightarrow r \quad \mathbf{if} \quad c$$

- $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$,
- 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 $t|_p$ such that

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

.

Example

$$\begin{aligned} \text{even}(0) &\rightarrow \text{true} \\ \text{even}(s(x)) &\rightarrow \text{odd}(x) \\ \text{odd}(x) &\rightarrow \text{true} \quad \mathbf{if} \quad \text{not}(\text{even}(x)) \\ \text{odd}(x) &\rightarrow \text{false} \quad \mathbf{if} \quad \text{even}(x) \end{aligned}$$
$$\text{even}(s(0)) \longrightarrow \text{odd}(0) \longrightarrow \text{false}$$

Generalized rules

$l \rightarrow r$ **where** $p_1 := c_1 \dots$ **where** $p_n := c_n$

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

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

The pattern p_i is often reduced to a variable x .

Applying such a rewrite rule

$$l \rightarrow r \text{ where } p_1 := c_1 \dots \text{ where } p_n := c_n$$

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 $c_i\sigma\mu_1 \dots \mu_{i-1}$, for $i = 1, \dots, n$.

$$\text{move}(S) \rightarrow C(x, y) \text{ where } \langle x, y \rangle := \text{position}(S) \text{ if } x = y$$

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 $l \rightarrow r$ in \mathcal{R}

$$t \longrightarrow_{\mathcal{R}, AC}^{p, l \rightarrow r} t'$$

iff

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

Example of an AC theory

\cup an AC operator

$$\{i\} \cup s \rightarrow i$$

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

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

...

$$\{5\} \cup \{1\} \cup \{2\} \cup \{3\} \cup \{4\}$$

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.

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

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

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

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

...

Conditional AC rewriting

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

applied modulo AC on t ,

rewrite to any $t' = t[\sigma(r)]_p$ 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

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

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

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

Example

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

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

$$a + (a + b)$$

More extensions

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

Contents

- 1- Definition of a rewrite relation
- 2- Abstract rewrite systems
- 3- Rewriting and equating: a BIG GIB difference
- 4- Rewrite relation expressiveness

5- To know more

To know more

- The rewriting page:
<http://rewriting.loria.fr/>
- 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