Techniques de réécriture et transformations

Horatiu CIRSTEA, Pierre-Etienne MOREAU

Session 2008/2009
Techniques de réécriture et transformations

Horatiu CIRSTEA, Pierre-Etienne MOREAU

Session 2008/2009

Documents d’accompagnement :
www.loria.fr/~ckirchne/=master
Mathematics is frequently described as “the science of pattern,” a characterisation that makes more sense than most, both of pure mathematics, but also of the ability of mathematics to connect to the world teeming with patterns, symmetries, regularities, and uniformities.

Jon Barwise
Lawrence Moss
(Some) Additional Recommended Readings

- L’intelligence et le calcul (may be translated to English?)
  Jean-Paul Delahaye
  Look also at his web page

- Term Rewriting Systems
  Terese (M. Bezem, J. W. Klop and R. de Vrijer, eds.)
  Cambridge University press, 2002

- Term Rewriting and all That
  Franz Baader and Tobias Nipkow
  Cambridge University press, 1998

- The Rewriting Calculus Home page
  rho.loria.fr

- Repository of Lectures on Rewriting and Related Topics
  qsl.loria.fr

- The rewriting and IFIP WG1.6 page
  rewriting.loria.fr
Roadmap

1. A smooth introduction

2. Defining term rewriting
   - Terms
   - Substitutions
   - Matching
   - Rewriting
   - On the use of rewriting
   - Extended notions of rewriting

3. Rewriting modulo
   - Matching modulo
A smooth introduction

A simple game

The rules of the game:

•• \rightarrow \circ

○○ \rightarrow ○

•○ \rightarrow ●

○● \rightarrow ●

A starting point:

● ○ ● ○ ● ○ ● ● ● ● ○ ○ ● ○ ○ ● ● ○

Who wins?

Who puts the last white?
May I always win? Does the game terminate? Do we always get the same result?
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:

\[
0 + x = x \\
\text{s}(x) + y = \text{s}(x + y)
\]

What’s the result of \(\text{s}(\text{s}(0)) + \text{s}(\text{s}(0))\)?

\[
\text{s}(\text{s}(0)) + \text{s}(\text{s}(0)) = \text{s}(\text{s}(0) + \text{s}(\text{s}(0))) \\
= \text{s}(\text{s}(0 + \text{s}(\text{s}(0)))) \\
= \text{s}(\text{s}(\text{s}(\text{s}(0)))) \\
= \text{s}(\text{s}(\text{s}(\text{s}(0)))) \\
= 0 + 0 + 0 + \text{s}(\text{s}(\text{s}(\text{s}(0)))) \\
= \ldots
\]

Is there a better result?
Compute a result by turning the equalities into rewrite rules:

\[
\begin{align*}
0 + x & \rightarrow x \\
\text{s}(x) + y & \rightarrow \text{s}(x + y)
\end{align*}
\]

\[
\text{s}(\text{s}(0)) + \text{s}(\text{s}(0)) \rightarrow \text{s}(\text{s}(0) + \text{s}(\text{s}(0))) \\
\text{s}(\text{s}(\text{s}(0))) & \rightarrow \\
\text{s}(\text{s}(\text{s}(\text{s}(0))))
\]

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?

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: \( s(s(0)+s(s(0))) \)

Note that this last entity is a NEW object.
Fibonacci

\[
\begin{align*}
[\alpha] & \quad \text{fib}(0) \rightarrow 1 \\
[\beta] & \quad \text{fib}(1) \rightarrow 1 \\
[\gamma] & \quad \text{fib}(n) \rightarrow \text{fib}(n - 1) + \text{fib}(n - 2)
\end{align*}
\]

\[
\text{fib}(3) \rightarrow \text{fib}(2) + \text{fib}(1)
\]

\[
\text{fib}(2) + \text{fib}(1) \rightarrow \text{fib}(2) + 1
\]

\[
\text{fib}(2) + 1 \rightarrow \text{fib}(1) + \text{fib}(0) + 1
\]

Finally \( \text{fib}(3) = 3, \text{fib}(4) = 5, \ldots \)
A smooth introduction

Graphical Rewriting

\[ F \rightarrow F + F - F - FF + F + F - F \]

L-systems (Lindenmeier)
A smooth introduction

Ecological Rewriting

http://algorithmicbotany.org/
Sorting by rewriting

rules for List
  X, Y : Nat ; L L' L'' : List;
  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) => ...

sorts List ; subsorts Nat < List ;
operators
  nil : List ;
  @ @ : (List List) List [associative id: nil] ;
  sort @ : (List) List ;
end
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.
1 A smooth introduction

2 Defining term rewriting
   - Terms
   - Substitutions
   - Matching
   - Rewriting
   - On the use of rewriting
   - Extended notions of rewriting

3 Rewriting modulo
   - Matching modulo
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
Defining term rewriting

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 \( i \)

\( \mathcal{F} = \bigcup_n \mathcal{F}_n \)

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

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

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

\( \mathcal{T}(\mathcal{F}, \emptyset) = \mathcal{T}(\mathcal{F}) \) is the set of ground terms.
Examples and (some) terminology

With the following signature:
\[ \mathcal{F} = \{ f, a \} \] with \( \text{arity}(f) = 2 \), \( \text{arity}(a) = 0 \), \( x, y, z \in \mathcal{X} \):
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)
Defining term rewriting

Terms as mappings: \( \mathbb{N}, . \) \( \rightarrow \) \( \mathcal{F} \)

\[ t = f(a + x, h(f(a, b))) \]

is represented by:

<table>
<thead>
<tr>
<th>position</th>
<th>symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \wedge )</td>
<td>( f )</td>
</tr>
<tr>
<td>1</td>
<td>+</td>
</tr>
<tr>
<td>1.1</td>
<td>( a )</td>
</tr>
<tr>
<td>1.2</td>
<td>( x )</td>
</tr>
<tr>
<td>2</td>
<td>( h )</td>
</tr>
<tr>
<td>2.1</td>
<td>( f )</td>
</tr>
<tr>
<td>2.1.1</td>
<td>( a )</td>
</tr>
<tr>
<td>2.1.2</td>
<td>( b )</td>
</tr>
</tbody>
</table>

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

\[ \text{let } t[s]_\omega \text{ denote the term } t \text{ with } s \text{ as subterm at position (or occurrence) } \omega. \]
\[ \text{let } t|_\omega \text{ denote the subterm at occurrence } \omega. \]

\[ f(a + x, h(f(a, b)))|_2 = h(f(a, b)) \]
Terms as trees

\[ t = f(a + x, h(f(a, b))) \] is represented by:

\[
\begin{array}{c}
|t| \text{ is the size of } t \text{ i.e. the cardinality of } \text{Dom}(t).

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

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

\mathcal{V}ar(f(a + x, h(f(a, b)))) = \{x\}
\end{array}
\]
Simple questions—

What is $f(f(a, b), g(a))|_{1.1}$? — $a$
What is $f(f(a, b), g(a))|_{\wedge}$? — $f(f(a, b), g(a))$
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
Defining term rewriting

Substitutions

Substitution

A substitution $\sigma$ is a mapping from the set of 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

$\sigma(f(x, f(a, z))) = f(a, f(a, g(z)))$. 
Defining term rewriting

Matching
Defining term rewriting

Matching

Finding a substitution $\sigma$ 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.
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 \( t \)
- \( \sigma \) 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

\[ s \leq t \not\Rightarrow f(u, s) \leq f(u, t) \]
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
\[ x \leq a \text{ but } (x \mapsto b)x \not\leq (x \mapsto b)a \]
Defining term rewriting

Matching: A rule based description

Delete
\[ t \ll ? t \land P \]
\[ \rightarrow P \]

Decomposition
\[ f(t_1, \ldots, t_n) \ll ? f(t'_1, \ldots, t'_n) \land P \]
\[ \rightarrow \land_{i=1,\ldots,n} t_i \ll ? t'_i \land P \]

SymbolClash
\[ f(t_1, \ldots, t_n) \ll ? g(t'_1, \ldots, t'_m) \land P \]
\[ \rightarrow \text{Fail} \]
if \( f \neq g \)

SymbolVariableClash
\[ f(t_1, \ldots, t_n) \ll ? x \land P \]
\[ \rightarrow \text{Fail} \]
if \( x \in X \)

MergingClash
\[ x \ll ? t \land x \ll ? t' \land P \]
\[ \rightarrow \text{Fail} \]
if \( t \neq t' \)
Find a match

\[ x + (y \times 3) \ll ? 1 + (4 \times 3) \]

\[ \Rightarrow \text{Decomposition } x \ll ? 1 \land y \times 3 \ll ? 4 \times 3 \]

\[ \Rightarrow \text{Decomposition } x \ll ? 1 \land y \ll ? 4 \land 3 \ll ? 3 \]

\[ \Rightarrow \text{Delete } x \ll ? 1 \land y \ll ? 4 \]

\[ x + (y \times y) \ll ? 1 + (4 \times 3) \]

\[ \Rightarrow \text{Decomposition } x \ll ? 1 \land y \times y \ll ? 4 \times 3 \]

\[ \Rightarrow \text{Decomposition } x \ll ? 1 \land y \ll ? 4 \land y \ll ? 3 \]

\[ \Rightarrow \text{MergingClash } \text{Fail} \]
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 \( \text{Var}(t) \cap \text{Var}(t') = \emptyset \), 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
XSLT
“pattern matching” in general

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

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

\[ t \xrightarrow{l \rightarrow r}_p t' \]
A term rewrite system \( R \) (a set of rewrite rules) determines a relation on terms denoted \( \rightarrow_R \):

\[
\begin{align*}
U \rightarrow_R V \\
\text{iff} \\
\text{there exist } t, l \rightarrow r \in R, \text{ an occurrence } \omega \text{ in } t, \text{ such that} \\
u = t[\sigma(l)]_\omega \\
\text{and} \\
v = t[\sigma(r)]_\omega \\
\end{align*}
\]

\[
\begin{align*}
t[\sigma(l)]_\omega \rightarrow_R t[\sigma(r)]_\omega 
\end{align*}
\]

USUALLY, when defining the rewriting relation, one requires the all rewrite rules satisfy \( \text{Var}(r) \subseteq \text{Var}(l) \).
Consider a binary relation $\rightarrow$, we define the

- transitive closure: $\rightarrow^+$
- transitive reflexive closure: $\rightarrow^*$
- symmetric closure: $\leftrightarrow$
Simple examples —

Consider the rewrite system $R$:

\[
\begin{align*}
  x + x & \rightarrow x \\
  (a + x) + y & \rightarrow y + x
\end{align*}
\]

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 $\rightarrow_R$? — yes

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

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

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

Why?
[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: rewriting calculus
- Conditional rewriting
- Theorem proving and rewriting
- Rewriting and higher-order features
- Types and rewriting
Extended notions of rewriting
Conditional rules

\[ l \rightarrow r \text{ if } c \]

- \( l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X}) \),
- \( c \) 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 \( \sigma \) allows \( c\sigma \) to reduce to \( \text{true} \).
Applying a conditional rewrite rule

\[
\begin{align*}
even(0) & \rightarrow true \\
even(s(x)) & \rightarrow odd(x) \\
odd(x) & \rightarrow true \quad \text{if} \quad \neg \even(x) \\
odd(x) & \rightarrow false \quad \text{if} \quad \even(x)
\end{align*}
\]

\[
even(s(0)) \rightarrow odd(0) \rightarrow false
\]
Generalized rules

\[ l \rightarrow r \text{ where } p_1 := c_1 \ldots \text{ where } p_n := c_n \]

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

where \( \text{true} := c \) is equivalently written if \( c \).
\( p_i \) is often reduced to a variable \( x \).
Generalized rule application

\[ l \rightarrow r \text{ where } p_1 := c_1 \ldots \text{ where } p_n := c_n \]

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

\[ \text{move}(S) \rightarrow C(x, y) \quad \text{where} \quad < x, y > := \text{position}(S) \quad \text{if} \quad x = y \]
1. A smooth introduction

2. Defining term rewriting
   - Terms
   - Substitutions
   - Matching
   - Rewriting
   - On the use of rewriting
   - Extended notions of rewriting

3. Rewriting modulo
   - Matching modulo
Equality modulo $C$

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

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

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

**Theorem**

$$t_1 + t_2 = C(+) t'_1 + t'_2 \iff (t_1 = C(+) t'_1 \land t_2 = C(+) t'_2)$$

$$\lor$$

$$(t_1 = C(+) t'_2 \land t_2 = C(+) t'_1)$$
Finding a substitution $\sigma$ such that

$$\sigma(l) = t$$

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

Finding a substitution $\sigma$ such that

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

is called the matching problem from $l$ to $t$ (denoted $l \ll^?_E t$).
\( \mathcal{F} = \{a(0), b(0), c(0), f(2), g(2), h(1)\} \)

\( f \) is assumed to be \textbf{commutative} (the other symbols have no property).

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

- \( f(a, b) = f(b, a) \) \hspace{1cm} — yes
- \( g(f(a, b), a) = g(f(b, a), a) \) \hspace{1cm} — yes
- \( g(f(a, b), a) = g(a, f(b, a)) \) \hspace{1cm} — no
- \( f(a, f(a, b)) = f(f(b, a), a) \) \hspace{1cm} — yes
- \( f(a, f(b, c)) = f(f(c, b), a) \) \hspace{1cm} — yes
- \( f(f(a, b), c) = f(a, f(b, c)) \) \hspace{1cm} — no
Solve the following problems:

1. \( f(x, y) \triangleleft_C^? f(a, b) \)
   - \( \sigma = \{ x \mapsto a, y \mapsto b \} \)
   - \( \sigma = \{ x \mapsto b, y \mapsto a \} \)

2. \( f(y, f(x, x)) \triangleleft_C^? f(f(f(a, b), f(b, a)), f(b, a)) \)
   - \( \sigma = \{ x \mapsto f(a, b), y \mapsto f(a, b) \} \)
Matching modulo $C$ : A rule based description

Delete

$t \ll ? t \land P$

$\implies P$

Decomposition

$f(t_1, \ldots, t_n) \ll ? f(t'_1, \ldots, t'_n) \land P$

$\implies \land_{i=1,\ldots,n} t_i \ll ? t'_i \land P$

SymbolClash

$f(t_1, \ldots, t_n) \ll ? g(t'_1, \ldots, t'_m) \land P$

$\implies \text{Fail}$

SymbolVariableClash

$f(t_1, \ldots, t_n) \ll ? x \land P$

$\implies \text{Fail}$

MergingClash

$x \ll ? t \land x \ll ? t' \land P$

$\implies \text{Fail}$
Assume $+$ commutative

\[
\begin{align*}
\text{C-Dec} & \quad t_1 + t_2 \ll_C^{?} t'_1 + t'_2 \land P \\
\iff & \quad (t_1 \ll_C^{?} t'_1 \land t_2 \ll_C^{?} t'_2 \land P) \lor (t_1 \ll_C^{?} t'_2 \land t_2 \ll_C^{?} t'_1 \land P)
\end{align*}
\]
Find a match

\[ x \ast (3 + y) \ll_C^? 1 \ast (4 + 3) \]

⇒ Decomposition

\[ x \ll_C^? 1 \land 3 + y \ll_C^? 4 + 3 \]

⇒ C(+) - Decomposition

\[ x \ll_C^? 1 \land ((3 \ll_C^? 4 \land y \ll_C^? 3) \lor (3 \ll_C^? 3 \land y \ll_C^? 4)) \]

⇒ MergingClash

\[ x \ll_C^? 1 \land (\text{Fail} \lor (3 \ll_C^? 3 \land y \ll_C^? 4)) \]

⇒ Delete

\[ x \ll_C^? 1 \land (\text{Fail} \lor (y \ll_C^? 4)) \]

⇒ Bool

\[ x \ll_C^? 1 \land y \ll_C^? 4 \]
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 $\var(t) \cap \var(t') = \emptyset$, 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 $\bigvee_{k \in K} \bigwedge_{i \in I} x_i^k \ll^? t^k_i$ 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: $t = t'$. 
\( \cup \) is assumed to be an associative commutative (AC) symbol:

\[
\forall x, y, z, \quad \cup(x, \cup(y, z)) = \cup(\cup(x, y), z) \quad \text{and} \quad \forall x, y, \quad \cup(x, y) = \cup(y, x).
\]

\( \{i\} \cup s \preceq \cup \{1\} \cup \{2\} \cup \{3\} \cup \{4\} \cup \{5\} \)

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

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

\( \ldots \)

\( \{5\} \cup \{1\} \cup \{2\} \cup \{3\} \cup \{4\} \)

5 different and non AC-equivalent matches.
Solve the following problems:

1. \( f(x, y) \mathrel{\ll_{AC}} f(a, b) \)
   - \( \sigma = \{ x \mapsto a, y \mapsto b \} \)
   - \( \sigma = \{ x \mapsto b, y \mapsto a \} \)

2. \( f(y, f(x, x)) \mathrel{\ll_{AC}} f(f(f(a, b), f(b, a)), f(b, a)) \)
   - \( \sigma = \{ x \mapsto f(a, b), y \mapsto f(a, b) \} \)
   - \( \sigma = \{ x \mapsto a, y \mapsto f(f(b, b), f(b, a)) \} \)
   - \( \sigma = \{ x \mapsto b, y \mapsto f(f(a, a), f(b, a)) \} \)
   - ...
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.

\[
\begin{align*}
x + 0 & \rightarrow x \\
x + (0 + y) & \rightarrow x + y \\
x + (\neg x) & \rightarrow 0 \\
x + ((\neg x) + y) & \rightarrow y \\
\neg x & \rightarrow x \\
\neg 0 & \rightarrow 0 \\
-(x + y) & \rightarrow (\neg x) + (\neg y)
\end{align*}
\]

\[
\begin{align*}
x + y & = y + x \\
(x + y) + z & = x + (y + z)
\end{align*}
\]
\( t \) \((R/A)\)-rewrites to \( t'\) if \( t = A t_1 \xrightarrow{R} t_2 = A t'\).

To be more effective, consider any relation \( \rightarrow_{RA} \) such that:

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

\[ u \rightarrow_{R,A} v \]

iff

there exist $l \rightarrow r \in R$, an occurrence $\omega$ 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 $\text{Var}(r) \subseteq \text{Var}(l)$. 
For example

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

\[
\{i\} \cup x \rightarrow i
\]

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

\[
\ldots
\]

\[
\{5\} \cup \{1\} \cup \{2\} \cup \{3\} \cup \{4\}
\]

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)

\[ R = \{ a + a \rightarrow a \} \]

\[ R/E\text{-rewrite the term } (a + c) + a \]
\[ R, E\text{-rewrite the term } (a + c) + a \]

\[ R = \{ a + a \rightarrow a \ (a + a) + x \rightarrow a + x \} \]

\[ R/E\text{-rewrite the term } (a + c) + a \]
\[ R, E\text{-rewrite the term } (a + c) + a \]
Huet’s approach [JACM80] uses standard rewriting $\rightarrow R$ but is restricted to left-linear rules.

Peterson and Stickel’s approach [JACM81] uses rewriting modulo $A$, denoted $\rightarrow_{R,A}$, 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.