Rewriting

Rewriting is a technique to :

- automate proofs
- compute terms evaluation,
- do prototyping

Principle of proof of properties :
Orientation of equations $\Rightarrow$ rewrite rules

Problems :

- which direction, is the set of rewriting rules complete ?
- termination
- confluence

Introduction to rewriting principles

- From axiom **not true = false** , we can built the rewrite rule $not(true) \rightsquigarrow_1 false$
- Abstract rewriting system is a proof view of the rewriting process (as opposed to its operational view) defined as relation between equivalent terms.
- Operational mechanism used to do the rewriting process :
  - filtering = choice of the rule by matching the left term to the term to rewrite.
  - substitution of the matching part with the specialised right part of the rule
- Closure of the rewrite rules to build a normal form (irreductible form)

Abstract Rewrite Systems

Definition (Abstract rewrite system)

Let $\Sigma = \langle S, F \rangle$ be a signature and X be a S-sorted set of variables.

- An abstract term rewriting system (ARS) is $A = (T_{\Sigma,X}, \rightarrow)$, where : $\rightarrow \subseteq T_{\Sigma,X} \times T_{\Sigma,X}$

A rewrite step $l \leadsto r \in \rightarrow$ can be completed by the already defined deduction principles. We would like to omit the rules that can introduce non terminating process (for instance symmetry and reflexion)

Closure of Abstract Rewrite Systems

Definition (Closure of Abstract Rewrite system)

Let $\Sigma = \langle S, F \rangle$ be a signature and X be a S-sorted set of variables and an abstract term rewriting system (ARS) $A = (T_{\Sigma,X}, \rightarrow)$, where : We define $Closure(A) = (T_{\Sigma,X}, \rightarrow^*)$: an abstract rewrite system obtained by applying the following rules.

- $\forall \sigma \in X_s \rightarrow (T_{\Sigma,X})_s, (t, t') \in \rightarrow \Rightarrow (\sigma t, \sigma t') \in \rightarrow^*$
- $\forall f \in \Sigma, (t_i, t_i') \in \rightarrow^* \Rightarrow (ft_1, ..., t_n, ft_1, ..., t_n') \in \rightarrow^*$
- $\forall (t, t') and (t', t'') \in \rightarrow^* \Rightarrow (t, t'') \in \rightarrow^*$

From the rewrite rules, a rewrite relation can be computed as extension of the effect of all rewrite rules, according to variable *substitution* and encapsulation in function application (*substitutivity*).

Closure of Abstract Rewrite Systems

Definition (Rewrite rules)

Let $\Sigma = \langle S, F \rangle$ be a signature and X be a S-sorted set of variables.

- We note $Rew_{\Sigma,X} \subseteq T_\Sigma(X) \times T_\Sigma(X)$ a set of rewrite rules for a given signature and variables.

A rewrite rule $l \rightsquigarrow r$ can be derived from axioms $l = r$ by just taking the left and right part of the equality. In general this is not sufficient.

Proof of equalities

Definition (Rewrite theories)

Given $Spec = < \Sigma, X, AX >$ and an abstract term rewriting system $A = (T_{\Sigma,X}, \rightarrow)$, and its closure :
$\forall t_1, t_2 \in T_{\Sigma,X}$,
$t_1 = t_2 \in Th_{\rightarrow}(Spec) \Leftrightarrow \exists t \in T_{\Sigma,X}, t_1 \rightarrow^* t \wedge t_2 \rightarrow^* t$

Validity :

$$Th_{\rightarrow}(Spec) \subseteq Th(Spec)$$

Operational Rewriting of terms

Definition (Rewrite step)

Let $\Sigma = \langle S, F \rangle$ be a signature and X be a S-sorted set of variables and $l \rightsquigarrow r$,$l, r \in T_\Sigma(X)$ a rewrite rule.

- $filter(t, l) =< \sigma, c > \Leftrightarrow \exists \sigma \in X_s \rightarrow (T_{\Sigma,X})_s,\ \exists c,$
  - $t = c[\sigma l]$ [a]
  - $t' = c[\sigma r]$
- $< t, t' > \in Rew_{l \rightsquigarrow r}$ a rewrite step

---

a. $c[\_]$ denotes the context of a term, i.e. a term with a place holder

Considering $\rightsquigarrow^*$ the transitive closure of$\rightsquigarrow$, they are supposed to be confluent and with finite termination, i.e
$\forall t$,$\exists$unique e s.t $t \rightsquigarrow^* e$ and e is not reducible.

Context

Definition (Context and Subterms)

Let $\Sigma = \langle S, \leq, F \rangle$ be an order-sorted signature and X be a S-sorted variable set, let also $\Box \notin F \cup X$ be a special constant symbol called a placeholder.

- A context C of a term $t \in T_{\Sigma,X}$ is a term $(T_{\Sigma \cup \Box, X})_s$
- if $C_t[\Box_1, \ldots, \Box_n]$ is a context with n occurrences of $\Box$ and $t_1, \ldots, t_n$ are terms $\in (T_{\Sigma \cup \Box, X})_s$, then $C_t[t_1, \ldots, t_n]$ is the result of replacing the $\Box_i$ by the t_i.
- A term $st \in (T_{\Sigma,X})_s$ is a *subterm* of $t \in (T_{\Sigma,X})_s$ noted $st \subseteq t$ if there exists a context C of term t denoted $C_t[\Box]$ such that $t = C_t[st]$.

Example : $t = suc(suc(suc(0)))$, $C_t = suc(suc(\Box))$, $st = suc(0)$.

Substitution

Definition (Substitution)

Let $\Sigma = \langle S, F \rangle$ be a signature and X be a S-sorted variable set. A substitution σ is mapping $\sigma : X_s \to \left(T_{\Sigma,X}\right)_s$ where $s \in S$. Every substitution σ extends uniquely to a morphism $\sigma^{\#} : (T_{\Sigma,X})_s \to (T_{\Sigma,X})_s$, where $s \in S$

- $\sigma^{\#}(f(t_1, \ldots, t_n)) = f(\sigma^{\#}(t_1), \ldots, \sigma^{\#}(t_n)))$
- $\sigma^{\#}(f_s) = f_s$ with $f_s \in F_{\epsilon,s}$
- $\sigma^{\#}(x_s) = \sigma(x_s)$

Example : $\sigma : \{x_s \to a_s; y_s \to b_s\}$ with $x_s, y_s \in X_s$, a_s, $b_s \in F_{\epsilon,s}$.
$t = f(x_s, y_s)$, $\sigma^{\#}(t) = f(a_s, b_s)$.

Example of rewriting in algebraic specification

We will provide an example of algebraic specification in order to illustrate rewriting issues.
The example will cover several simple sorts and simple axioms interdependant to each other.
$S = \{nat, bool\}$
$OP = \{+ : nat, nat \rightarrow nat, 0 :\rightarrow nat, suc : nat \rightarrow nat, 1 :\rightarrow nat$
$not : bool \rightarrow bool, true :\rightarrow bool, false :\rightarrow bool, >: nat, nat \rightarrow bool\}$
$X_{nat} = \{x, y, z\}$ and $X_{bool} = \{a, b\}$
The axioms are :
$x + 0 = x\,;\, x + suc(y) = suc(x + y)\,;\, 1 = suc(0)$
$not(true) = false\,;\, not(false) = true$
$0 > x = false\,;\, (suc(x) > 0) = true\,;\, (suc(x) > suc(y)) = x > y$

Example of rewriting algebraic specification terms(2)

The rewrite rules are :

$x + 0 \leadsto_1 x$;

$x + suc(y) \leadsto_2 suc(x + y)$;

$1 \leadsto_3 succ(0)$

$not(true) \leadsto_4 false$;

$not(false) \leadsto_5 true$

$0 > x \leadsto_6 false$;

$(suc(x) > 0) \leadsto_7 true$;

$(suc(x) > suc(y)) \leadsto_8 x > y$

Example of rewriting algebraic specification terms(3)

Rewriting the terms can be computed as follows[1] :

- $1 > 0 \rightsquigarrow_3 suc(0) > 0 \rightsquigarrow_{7,x=0}$ **true**
- $1 + 1 \rightsquigarrow_3 suc(0) + 1 \rightsquigarrow_3 suc(0) + suc(0) \rightsquigarrow_{2,x=suc(0),y=0} suc(suc(0) + 0) \rightsquigarrow_{1,x=suc(0)}$ **suc(suc(0))** [2]
- $(suc(1) + 1) > 1 + 1 \rightsquigarrow_3 (suc(suc(0)) + 1) > 1 + 1 \rightsquigarrow_3 (suc(suc(0)) + suc(0)) > 1 + 1 \rightsquigarrow_{2,x=suc(suc(0)),y=0} suc(suc(suc(0)) + 0) > 1 + 1 \rightsquigarrow_1 suc(suc(suc(0))) > 1 + 1 \rightsquigarrow_{3/3/2/1}$ [3] $suc(suc(suc(0)) > suc(suc(0)) \rightsquigarrow_{8/8} suc(0) > 0 \rightsquigarrow_7$ **true**

1. bold terms are canonical terms
2. reuse of already evaluated terms
3. reuse of several reductions

Proof of equalities

Definition (Rewrite theories)

Given $Spec = < \Sigma, X, AX >$ and a set of rewrite rules defining the relation $\rightsquigarrow$
$\forall t_1, t_2 \in T_{\Sigma,X}$,
$t_1 = t_2 \in Th_{\rightsquigarrow}(Spec) \Leftrightarrow \exists t \in T_{\Sigma,X}, t_1 \rightsquigarrow^* t \wedge t_2 \rightsquigarrow^* t$

Theorem (abstract and operational rules are identical)

Given $Spec = < \Sigma, X, AX >$ and a set of rewrite rules defining the relation $\rightarrow$ and $\rightsquigarrow$.
$Th_{\rightarrow}(Spec) \Leftrightarrow Th_{\rightsquigarrow}(Spec)$

Properties of rewrite rules

Convergence, confluence of a rewrite system :

- Property to reach for all terms a unique normal form, without taking care of the strategy.

Termination of a rewrite system :

- Property to reach for any terms, in a finite number of steps a normal form.

The use of graceful presentation will help in finding a good rewrite system.

Operational view :

Rewriting = rules + application mechanism + strategy

p. 7

Possible strategies :

- left-right-inner-most
- left-right-outer-most

on peut évaluer une opération plus ou moins

There is no optimal strategy (see in the book Rewrite systems, Jouannaud, Dershowitz p. 39).

Strategies : Operational rewriting a la TOM

permet d aider à choisir une stratégie d evaluation à l avance

Based on elementary rewrite rules, we can apply on terms a basic rewrite step.

on programme la strategie !

$Rew_{Ax}[t] : T_\Sigma \rightarrow (T_\Sigma \cup \{fail\})$

t ou fail

regle non applicable

$\exists \sigma,$
$(\sigma(l) = t) \Rightarrow Rew_{Ax \cup \{<l,r>\}}[t] = \sigma(r)$
$Rew_{Ax}[t] = fail$ otherwise

j essaie d applicquer l ecriture sur

This is the application of the rule at the root of the term. can fail if there is no possible rule application.
We use, in the tools, an order on the rules to provide with deterministic behaviours.

Implementation of Strategies

Way to find the context of a rewriting step !

$$Strat(S) : (T_\Sigma \cup \{fail\}) \rightarrow (T_\Sigma \cup \{fail\})$$

If $Strat(s)$ is defined, terms t will be rewritten with :

$$Strat(Rew_{Ax})[t]$$

Obviously :

$$(S)[fail] = fail \quad \checkmark$$

Strategies : Basic operations 1 (TOM)

$$(Identity)[t] = t$$
$$(Fail)[t] = fail$$
$$(s1)[t] = fail \Rightarrow (Sequence(s1, s2))[t] = fail$$
$$(s1)[t] = t' \Rightarrow (Sequence(s1, s2))[t] = (s2)[t']$$

on appilique les strategies en sequence

$$(s1)[t] = t' \Rightarrow (Choice(s1, s2))[t] = t'$$
$$(s1)[t] = fail \Rightarrow (Choice(s1, s2))[t] = (s2)[t]$$

je fais S1 si possible, si S1 échoue, j applique

Strategies 2

$$\frac{(s)[t1] = t1', \ldots, (s)[tn] = tn'}{(All(s))[f(t1, \ldots, tn)] = f(t1', \ldots, tn')}$$

on applique f au

si 1 fail, tout le monde

$$\exists i, (s)[ti] = fail \Rightarrow (All(s))[f(t1, \ldots, tn)] = fail$$

$$(All(s))[cst] = cst$$

$$(s)[ti] = ti' \Rightarrow$$

au hasard ?

stratégie sous 1 sous terme (pas to

$$(One(s))[f(t1, \ldots, tn)] = f(t1, \ldots, ti', \ldots, tn)$$

$$(s)[t1] = fail, \ldots, (s)[tn] = fail \Rightarrow$$

$$(One(s))[f(t1, \ldots, tn)] = fail$$

$$(One(s))[cst] = fail$$

One is non deterministic ! It is not a functional strategy.

TOM : Strategies Library

μ is the recursion operator.

on tente la stratégie S, si ca fail

$$Try(s) = Choice(s, Identity)$$

$$Repeat(s) = \mu x.Choice(Sequence(s, x), Identity())$$

$$OnceBottomUp(s) = \mu x.Choice(One(x), s)$$

$$BottomUp(s) = \mu x.Sequence(All(x), s)$$

$$TopDown(s) = \mu x.Sequence(s, All(x))$$

$$Innermost(s) = \mu x.Sequence(All(Innermost(x)), Try(Sequence(s, x)))$$

Recursion

Fixpoint solution ! !

$eval(\mu x.t) = eval(\mu x.\sigma(t))$ with $\sigma(x) = t$

recursion

je remplace les x par t

Formally it is an infinite possible application of the t pattern.

$Repeat(s) = \mu x.Choice(Sequence(s, x), Identity())$

le x dans le terme est remplacé

$Repeat(s) =$
$\mu x.Choice(Sequence(s, Choice(Sequence(s, x), Identity())), Identity())$

$Repeat(s) =$
$\mu x.Choice(Sequence(s, Choice(Sequence(s, Choice(Sequence(s, x), Identity())), Identit$

Operationnaly, it can be evaluated with lazy procedure.

Rewrite system vs. strategies

Using strategies we can define the previously constructed rewrite operations :
Given a set of rewrite rules Rew and its rewrite relation $\rightsquigarrow^*$.
$t \rightsquigarrow^* t' \Leftrightarrow Innermost(Rew)[t] = t'$

Strategies in Prolog : strategies

```
strataxiom(try(S),choice(S,identity)).
strataxiom(repeat(S),choice(sequence(S,repeat(S)),identity)).
strataxiom(bottomup(S),sequence(all(bottomup(S)),S)).
strataxiom(topdown(S),try(sequence(S,all(topdown(S))))).
strataxiom(innermost(S),sequence(all(innermost(S)),
                        try(sequence(S,innermost(S))))).
identity(T,T).
fail(T,fail).

sequence(S1,S2,T,R):-eval(S1,T,R1),
                     (R1=fail,!,R=fail;
                      eval(S2,R1,R)).
choice(S1,S2,T,R):-eval(S1,T,R1),
                   (R1=fail,!,eval(S2,T,R);
                    R=R1).
all(S,T,R):- T=..[FCT|LP],listeval(S,LP,LR),
       (LR=fail,!,R=fail;R=..[FCT|LR]). /* treatment of f
```

Strategies in Prolog : evaluation

```
/* application of rules
rules from library (last eval rule) can be compiled if more
efficiency is needed (add two parameters
 systematically for terms)*/

eval(axiom,T,R):- (axiom(T,R),!;R=fail),!. /*must be determi
eval(identity,T,R):-identity(T,R),!.
eval(fail,T,R):-fail(T,R),!.
eval(sequence(S1,S2),T,R):-sequence(S1,S2,T,R),!.
eval(choice(S1,S2),T,R):-choice(S1,S2,T,R),!.
eval(all(S),T,R):-all(S,T,R),!.
eval(S,T,R):- strataxiom(S,CORPUS),print((S,T)),nl,
                          eval(CORPUS,T,R).

listeval(S,[],[]).
listeval(S,[T|LP],RES):-
            eval(S,T,R),(R=fail,!,RES=fail;listeval(S,LP,LR),
            (LR=fail,!,RES=fail;RES=[R|LR])).
```

Strategies in Prolog : axioms

```
/* atomic rewrite rules*/

axiom(X+0,X).
axiom(X+s(Y),s(X+Y)).

/*test queries*/

eval(innermost(axiom),s(s(0))+s(0),R).

eval(innermost(axiom), s(s(0)), R).

eval(topdown(axiom), 0, R).

eval(innermost(axiom),s(s(0))+s(s(0)),R).
```

Problems with Rewriting :

- The equality induced by the rewriting process is not the same as the one deduced from the axioms.
- We would obtain a equivalent system generated from the axioms (its not a decidable problem in all generality)
- Solution by orienting the equations, if the resulting system is confluent and terminate it is equivalent to the initial axioms.

Example of rules for boolean

Orientation from left to right :

- 1) $not(true) \leadsto false$;
- 2) $not(false) \leadsto true$;
- 3)$(true\ and\ b) \leadsto b$;
- 4) $(false\ and\ b) \leadsto false$;
- 5) $(true\ or\ b) \leadsto true$;
- 6) $(false\ or\ b) \leadsto b$;
- 7)$(false\ xor\ b) \leadsto b$;
- 8) $(true\ xor\ b) \leadsto not(b)$;

Given a term :

Example : weakness of orientation

```
 sort truc
Operations
    0: -> truc; +: truc truc -> truc; - : truc -> truc;
Axioms
  ax1: 0 + x = x
  ax2: x + (-x) = 0
```

Necessary rewrite rules :

- $0 + x \rightsquigarrow x$
- $x + (-x) \rightsquigarrow 0$
- $-0 \rightsquigarrow 0$? ? ? ?

Orienting is no sufficient i.e. -0 $=$ 0, the proof need ax1 from right to left and axiom 2 from left to right.

Termination

This is the property that for all terms there exist a normal form.
Example : Given the rewrite system, a,b constants, f,g functional symbols and x,y variables :

- 1) $f(a, b, x) \rightsquigarrow f(x, x, x)$;
- 2) $g(x, y) \rightsquigarrow x$;
- 3) $g(x, y) \rightsquigarrow y$;

The sequence :
$f(g(a, b), g(a, b), g(a, b)) \rightsquigarrow f(a, g(a, b), g(a, b)) \rightsquigarrow f(a, b, g(a, b)) \rightsquigarrow f(g(a, b), g(a, b), g(a, b)) \rightsquigarrow$ ... is infinite.
Remark : For a given rewrite system prooving its termination is indecidable. Various proof techniques have been proposed based on the construction of reduction ordering.

Confluence

The confluence property is verified if a rewrite system converge it is to a unique value. Example : Given the rewrite system, a,b,c constants, f,g functional symbols and x variable :

- 1) $f(x,x) \rightsquigarrow a$;
- 2) $f(x,g(x)) \rightsquigarrow b$;
- 3) $c \rightsquigarrow g(c)$;

Is not confluent.
For example, the normal form of f(c,c) is a and b. (c has no normal form).

- 1) $f(c,c) \rightsquigarrow a$;
- 2) $f(c,c) \rightsquigarrow f(c,g(c)) \rightsquigarrow b$;

Properties of rewrite rules

Theorem (validity)

Given $Spec =< \Sigma, X, AX >$, *and a set of rewrite rules obtained by orientation of the axioms defining the relation* $\rightsquigarrow$ *which is confluent and with termination*
$Th_{\rightsquigarrow}(Spec) \subseteq Th(Spec)$

Critical Pairs - Knuth-Bendix theorem

Let $l_1 \rightsquigarrow r_1$ and $l_2 \rightsquigarrow r_2$ be two rules of a term rewriting system. we suppose that these rules have no variables in common. If l_1^{sub} is a subterm (and not a variable) of l_1 (or the term itself) with $l_1^{context}[l_1^{sub}] = l_1$ and there exist a most general unifier σ such that $l_1^{sub}\sigma = l_2\sigma$, then $r_1\sigma$ and $l_1^{context}[r_2\sigma]$are called a critical pair.

The fact that all critical pairs of a term rewriting system can be reduced to the same expression, implies that the system is locally confluent.

Critical Pairs - Knuth-Bendix theorem(2)

The axioms of group theory are :

- $0 + x = x$
- $x^{-1} + x = 0$
- $(x + y) + z = x + (y + z)$

cf. exercices

Critical Pairs - Knuth-Bendix theorem(3)

For instance, if $f(x,x) \rightsquigarrow x$ and $g(f(x,y),x) \rightsquigarrow h(x)$, then $g(x,x)$ and $h(x)$ would form a critical pair because they can both be derived from $g(f(x,x),x)$.

Note that it is possible for a critical pair to be produced by one rule, used in two different ways. For instance, in the string rewrite "AA" $\rightsquigarrow$ "B", the critical pair ("BA", "AB") results from applying the one rule to "AAA" in two different ways. [4]

4. Rowland, Todd ; Sakharov, Alex ; and Weisstein, Eric W. "Critical Pair." From MathWorld–A Wolfram Web Resource.

Critical Pairs - Knuth-Bendix theorem(3)

Theorem (Knuth-Bendix)

Given a set of rewrite rules Rew, If $t \rightsquigarrow t_1$ *and* $t \rightsquigarrow t_2$ *then* $\exists t'$ *such that* $t_1 \rightsquigarrow^* t'$ *and* $t_2 \rightsquigarrow^* t'$ *or* $\exists (c_1, c_2)$ *a critical pair of Rew , a context* $C[]$ *and a substitution* σ *s.t.* $t_1 = C[c_1\sigma]$, $t_2 = C[c_2\sigma]$

Knuth Bendix completion

The Knuth-Bendix completion algorithm attempts to transform a finite set of identities into a finitely terminating, confluent term rewriting system whose reductions preserve identity. [5]

- Identities are equalities of two terms : $t_1 = t_2$. Two terms are equal for all values of variables occurring in them.
- A reduction order is another input to the completion algorithm. Every identity is viewed as two candidates for rewrite rules transforming the left-hand side into the right-hand side and vice versa.

5. This term rewriting system serves a decision procedure for validating identities.

Knuth Bendix completion

The output term rewriting system is used to determine whether $t_1 = t_2$ is an identity or not in the following manner.

- If two distinct terms t_1 and t_2 have the same normal form, then $t_1 = t_2$ is an identity.
- Otherwise, $t_1 = t_2$ is not an identity.

Term rewriting systems that are both finitely terminating and confluent have a unique normal forms for all expressions.

Knuth Bendix completion

Initially, this algorithm attempts to orient input identities according to the reduction order (if $t_1 < t_2$, then $t_1 \rightsquigarrow t_2$ becomes a rule). Then, it completes this initial set of rules with derived ones. The algorithm iteratively detects critical pairs, obtains their normal forms, and adds a new rule for every pair of the normal forms in accordance with the reduction order.

This algorithm may

1. Terminate with success and yield a finitely terminating, confluent set of rules,
2. Terminate with failure, or
3. Loop without terminating.

Exemple of completion

The axioms of group theory are :

- $0 + x = x$
- $x^{-1} + x = 0$
- $(x + y) + z = x + (y + z)$

If only left-right orientation is used then $x + x^{-1}$ is irreductible.
Using the completion, we have :

- $0 + x \rightsquigarrow x$, $x + 0 \rightsquigarrow x$
- $x^{-1} + x \rightsquigarrow 0$, $x + x^{-1} \rightsquigarrow 0$
- $(x + y) + z \rightsquigarrow x + (y + z)$, $0^{-1} \rightsquigarrow 0$
- $x^{-1} + (x + y) \rightsquigarrow y$, $x + (x^{-1} + y) \rightsquigarrow y$
- $(x^{-1})^{-1} \rightsquigarrow x$, $(x + y)^{-1} \rightsquigarrow x^{-1} + y^{-1}$

Summary

- Rewriting is an operational model adapted to the proof of properties on closed terms
- Termination and confluence are desired properties of a rewrite system.
- The use of finitelly generated models wrt to constructors simplify inductive proofs.