Rewrite Proofs and Computations*

Jean-Pierre Jouannaud

Laboratoire de Recherche en Informatique, Bat. 490, Université de Paris Sud,
F-91405 Orsay, France, jouannaud@lri.fr

Abstract. Rewriting is a general paradigm for expressing computations in various logics, and we focus here on rewriting techniques in equational logic. When used at the proof level, rewriting provides with a very powerful methodology for proving completeness results, a technique that is illustrated here. We also consider whether important properties of rewrite systems such as confluence and termination can be proved in a modular way. Finally, we stress the links between rewriting and tree automata. Previous surveys include [21; 18; 37; 12; 45; 46]. The present one owes much to [21].

Keywords. completion, confluence, critical pair, ground reducibility, inductive completion, local confluence, modularity, narrowing, order-sorted algebras, rewrite rule, rewriting, term algebra, termination, tree automata.

1 Introduction

The use of equations is traditional in mathematics. Its use in computer science has culminated with the success of algebraic specifications, a method of specifying software by encapsulating data defined abstractly by means of constructors, operations, and equations defining the action of the operations on the constructor expressions. It is also used for the axiomatic description of various calculi used in computer science.

Equations can be used for reasoning, by using Leibnitz's law of replacing equals by equals, a highly non-deterministic way of using equations. In contrast, rewriting uses the equations in one way, hence eliminating one source of non-determinism. This one way use is achieved by rewriting according to a well-founded ordering on terms, hence it is called a reduction. The other sources of non-determinism, choice of an equation, and of a subterm at which to rewrite can be eliminated as well if every expression has a unique irreducible form. In this case, rewriting defines a functional computation.

* This work was parly supported by the "Greco de programmation du CNRS", and the ESPRIT working group COMPASS.

The theory of rewriting which originates from algebra, theorem proving and computability theory centers therefore around the notion of normal form, an expression that cannot be reduced any further. Rewriting is first illustrated in section 2. The necessary algebraic apparatus is described in section 3. Confluence and termination are investigated in sections 4 and 5. How to obtain normal forms is described in section 6. Section 7 then describes all kinds of logical queries that can be solved by rewriting, provided normal forms exist. Section 8 investigates whether the existence of normal forms is a property closed under union of rewrite systems. The recognizability properties of the language of ground terms in normal form, are studies in section 9. Last but not least, we briefly mention several generalizations of rewriting in section 10.

2 Examples

Before we start the journey, we overview the road by means of a few examples taken from various knowledgeable sources. All of them raise questions which can be answered by using the material of this survey.

2.1 Hercules versus Hydra

This is one of Nachum Dershowitz's favorite examples:

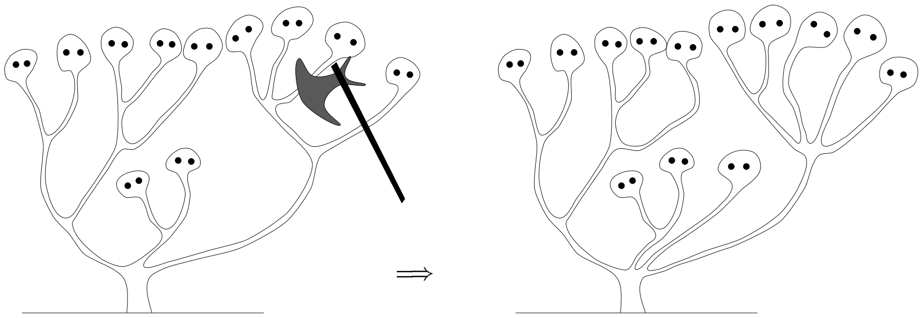


Fig. 1. Hercules versus Hydra

When Hercules chops off a head, finitely many new heads grow out from below: will Hercules eventually defeat Hydra?

This question has been extensively studied by logicians during the century. It is nothing but the uniform termination problem for this game.

2.2 Grecian Urn

This example is again taken from Dershowitz who took it from the greeks.

$$\bullet\bullet \;\to\; \circ \qquad \circ\circ \;\to\; \bullet \qquad \circ\bullet \;\to\; \circ \qquad \bullet\circ \;\to\; \circ$$

The Grecian urn contains 150 black beans, and 75 white beans. Greeks come to the urn, take out two beans and replace them by a single one according to the above rules. Can the colour of the last bean be predicted?

Uniform termination is obvious here. This game is related to an additional property: uniqueness of the resulting bean. The underlying property of the game is called confluence.

2.3 Chamelion Island

This example is due to Claude Marché, who took it from a famous mathematical contest, the "Olympiades".

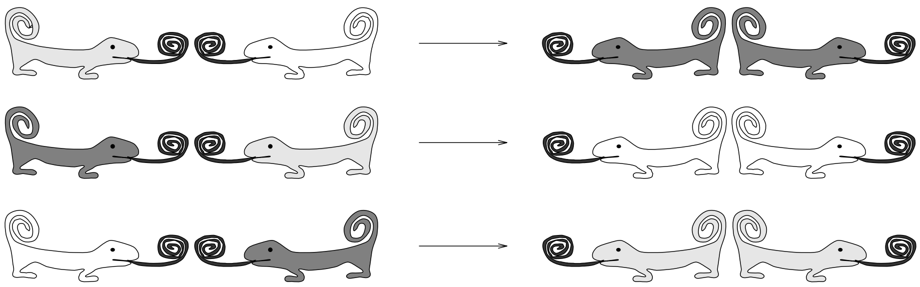


Fig. 2. Chameleons' Island

15 red, 14 blue, and 13 yellow chamelions are randomly wandering on chamelion island, and when two of them meet, they change their colour according to the above rules. Will they reach a stable state, all sharing the same colour?

Uniform termination is no more true for this game: starting from three chamelions of different colours terminates and results into three chamelions of the same colour; in contrast, starting with three pairs of chamelions, each pair of the same colour may result into an infinite sequence of changes, but may also result into a finite sequence ending in six chamelions of the same colour. The question is therefore whether there exist such an end. This is called weak termination.

2.4 Groups

Groups are usually defined by a set of three equations:

$$(x \cdot y) \cdot z \;=\; x \cdot (y \cdot z) \qquad x \cdot x^- \;=\; 1 \qquad x \cdot 1 \;=\; x$$

or, better, by a set of rewrite rules defining normal forms:

$$\begin{array}{rlrl}
(x \cdot y) \cdot z & \rightarrow x \cdot (y \cdot z) & \qquad (x \cdot y)^- & \rightarrow y^- \cdot x^- \\
x \cdot x^{-1} & \rightarrow 1 & x^- \cdot x & \rightarrow 1 \\
x \cdot 1 & \rightarrow x & 1 \cdot x & \rightarrow x \\
x^{--} & \rightarrow x & 1^- & \rightarrow 1 \\
x \cdot (x^- \cdot y) & \rightarrow y & x^- \cdot (x \cdot y) & \rightarrow y
\end{array}$$

This time, we have a set of rules which is both uniformly terminating and also confluent, which allows to compute a unique normal form for any expression, allowing us to compare two expressions by checking their respective normal forms for equality. This procedure is clearly effective, in contrast with replacing equals by equals which is clearly not.

Obtaining the set of rules automatically from the set of equations is called completion. The algorithm is due to Knuth and Bendix as well as the above set of rules for groups [47].

2.5 Abelian Groups

Abelian Groups are commutative groups:

$$\begin{array}{rlrl}
x \cdot y & = y \cdot x & \qquad (x \cdot y) \cdot z & = x \cdot (y \cdot z) \\
x \cdot x^- & = 1 & x \cdot 1 & = x
\end{array}$$

but can also be defined by a set of rules defining again normal forms up to associativity and commutativity. Indeed, these two equations cannot be turned into rules together, since otherwise normal forms could no more be computed in finite time by repeatedly replacing instances of left-hand sides by the corresponding right-hand sides.

$$x \cdot y \;=\; y \cdot x \qquad\qquad (x \cdot y) \cdot z \;=\; x \cdot (y \cdot z)$$

$$\begin{array}{rlcrl}
x \cdot x^{-1} & \rightarrow 1 & & (x \cdot y)^- & \rightarrow x^- \cdot y^- \\
x \cdot 1 & \rightarrow x & \qquad 1^- \rightarrow 1 \qquad & x^{--} & \rightarrow x
\end{array}$$

This kind of rewriting performs *modulo* associativity and commutativity, two axioms very common in practice. Completion modulo associativity and commutativity is due to Peterson and Stickel [63], and was later studied in more details by Jouannaud and Kirchner [38].

2.6 Commutative Rings

Rings use an additional operation, addition. The ring is commutative when so is its multiplication, as here. We give the rules only.

$$x + y = y + x \qquad (x + y) + z = x + (y + z)$$
$$x \cdot y = y \cdot x \qquad (x \cdot y) \cdot z = x \cdot (y \cdot z)$$

$$x + x^{-1} \to 0 \qquad x + 0 \to x$$
$$0^- \to 0 \qquad x^{--} \to x$$
$$(x + y)^- \to x^- + y^- \qquad x \cdot (y + z) \to (x \cdot y) + (x \cdot z)$$
$$x \cdot 1 \to x \qquad x \cdot y^- \to (x \cdot y)^- \qquad x \cdot 0 \to 0$$

2.7 Boolean Rings

The axioms are given in three parts. The first set expresses the usual connectives in terms of *and* and *exclusive or*. The next four equations state that both connectives are associative and commutative. Then, the remaining rules allow to compute normal forms built up from *and* and *exclusive or*. Note that there is no set of rules defining unique normal forms based on the usual *or* connective.

$$\neg x \to x \oplus \top \qquad x \vee y \to (x \wedge y) \oplus x \oplus y \oplus \bot$$
$$x \Rightarrow y \to (x \wedge y) \oplus x \oplus \top \qquad x \Leftrightarrow y \to (x \wedge y) \oplus x \oplus y$$

$$x \wedge y = y \wedge x \qquad (x \wedge y) \wedge z = x \wedge (y \wedge z)$$
$$x \oplus y = y \oplus x \qquad (x \oplus y) \oplus z = x \oplus (y \oplus z)$$

$$x \oplus \bot \to x \qquad x \oplus x \to \bot$$
$$x \wedge (y \oplus z) \to (x \wedge y) \oplus (x \wedge z) \qquad (x \oplus y) \wedge z \to (x \wedge z) \oplus (y \wedge z)$$
$$x \wedge \top \to x \qquad x \wedge \bot \to \bot \qquad x \wedge x \to x$$

The above convergent rewrite system for Boolean Rings was first given by Hsiang, based on previous work by Zegalkin [78], and Stone [71].

2.8 Combinatory Logic

$$I \cdot x \;\to\; x \qquad (K \cdot x) \cdot y \;\to\; x \qquad ((S \cdot x) \cdot y) \cdot z \;\to\; (x \cdot z) \cdot (y \cdot z)$$

From this example, it can then be easily predicted that rules can encode Turing machines. Actually, one single rule suffices to encode a Turing machine [14]. As a consequence, all useful properties of rewrite rules are undecidable, even when restricted to a single rule.

2.9 Programming

The following is a *rewrite program* to sort a list of natural numbers by inserting elements one by one into position, with lists represented in LISP notation and natural numbers in successor notation. *cons* and *nil* are therefore the list constructors, and

s and 0 are the constructors for natural numbers. We will use x and y for arbitrary natural numbers, and l for an arbitrary list of natural numbers.

$$\begin{aligned} max(0,x) &\to x & \qquad min(0,x) &\to 0 \\ max(x,0) &\to x & min(x,0) &\to 0 \\ max(s(x),s(y)) &\to s(max(x,y)) & min(s(x),s(y)) &\to s(min(x,y)) \end{aligned}$$

$$\begin{array}{rcl} sort(nil) & \to & nil \\ sort(cons(x,l)) & \to & insert(x, sort(l)) \\ insert(x,nil) & \to & cons(x,nil) \\ insert(x,cons(y,l)) & \to & cons(max(x,y), insert(min(x,y),l)) \end{array}$$

This program sorts numbers in decreasing order. If one wishes to sort them in ascending order, then the definition of *insert* must be rewritten. We therefore write a higher-order definition that will cover both cases by instantiating the two higher-order variables F and G (of type $nat \to nat \to nat$) by appropriate lambda expressions in each function *ascending*_sort and *descending*_sort.

$$\begin{aligned} ascending\_sort(l) &\to new\_sort(l, \lambda xy.min(x,y), \lambda xy.max(x,y)) \\ descending\_sort(l) &\to new\_sort(l, \lambda xy.max(x,y), \lambda xy.min(x,y)) \\ new\_sort(nil,F,G) &\to nil \\ new\_sort(cons(x,l),F,G) &\to insert(x, new\_sort(l,F,G),F,G)) \\ insert(x,nil,F,G) &\to cons(x,nil) \\ insert(x,cons(y,l),F,G) &\to cons(Fxy, insert(Gxy,l,F,G)) \end{aligned}$$

The question is whether confluence and termination proofs for these sets of rules can be performed in a modular way. This is especially important for practical applications, where rules defined in different modules should be proved confluent and terminating in their respective modules. The free use of lambda-expressions for instantiating higher-order variables of course complicates the question. Further complications arise if we assume a type system rich enough to accommodate polymorphic types and if one wants to prove properties of such programs like $new\_sort(l, \lambda xy.x, \lambda xy.y) = l$.

Here, assuming a monomorphic type discipline, we can deduce the termination and confluence properties of the whole set of rules from the termination proof of each set by showing that the interactions between all these sets of rules do not falsify these properties.

3 Equations and Rewrite Rules

We recall here standard notions about terms, substitutions, rewrite relations, equations and rewrite rules.

3.1 Terms

We start with a *signature* $\mathcal{F} = \cup_{n\geq 0}\mathcal{F}_n$ of function symbols. $\mathcal{F}_0 = \mathcal{C}$ is the set of constant symbols. Assuming the sets $\mathcal{F}_n$ are disjoint, each symbol f has the *arity* n if and only if $f \in \mathcal{F}_n$. We also assume a (denumerable) set $\mathcal{X}$ of variable symbols.

The set of (first-order) *terms* $\mathcal{T}(\mathcal{F},\mathcal{X})$ over $\mathcal{F}$ and $\mathcal{X}$ is the smallest set containing $\mathcal{X}$ such that $f(t_1,\ldots,t_n)$ is in $\mathcal{T}(\mathcal{F},\mathcal{X})$ whenever $f \in \mathcal{F}_n$ and $t_i \in \mathcal{T}(\mathcal{F},\mathcal{X})$ for $i = 1,\ldots,n$. Variable-free terms are called *ground*; the set $\mathcal{T}(\mathcal{F},\emptyset)$ of ground terms is also written $\mathcal{G}(\mathcal{F})$.

A term t in $\mathcal{T}(\mathcal{F},\mathcal{X})$ may be viewed as a finite ordered tree, the leaves of which are labeled with variables (from $\mathcal{X}$) or constants (from $\mathcal{C}$) and the internal nodes of which are labeled with function symbols (from $\mathcal{F}-\mathcal{C}$) of positive arity, with outdegree equal to the arity of the label. A *position* within a term may be represented—in Dewey decimal notation—as a sequence of positive integers, describing the path from the outermost *root* symbol to the head of the subterm at that position. By $t|_p$, we denote the *subterm* of t rooted at position p, and by $t[s]_p$ the term obtained by replacing t_p by s in t. As a consequence, $t = t[t_p]_p$. We use Λ for the root position, $\mathcal{P}os(t)$ for the set of positions of t, and $\mathcal{FP}os(t)$ for the set of non-variable positions of t.

We use $|t|$ for the size (number of nodes) of t, and $\mathcal{V}ar(t)$ for its set of variables; we say that t is *linear* if no variable occurs more than once in t.

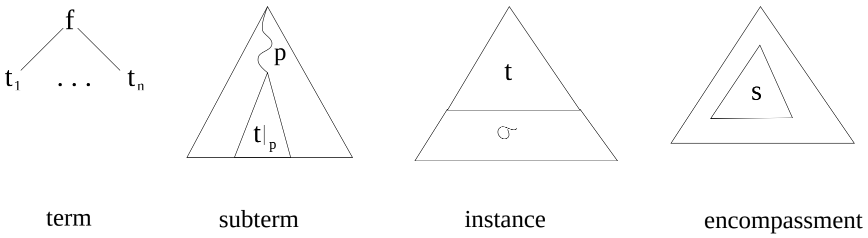


Fig. 3. Tree Representation of Terms.

3.2 Substitutions

A *substitution* is a special kind of replacement operation, uniquely defined by a mapping from variables to terms, and written out as $\{x_1 \mapsto s_1,\ldots,x_m \mapsto s_m\}$ when there are only finitely many variables x_i not mapped to themselves. Formally, a substitution σ is a function from $\mathcal{X}$ to $\mathcal{T}(\mathcal{F},\mathcal{X})$, extended to a function again written σ from $\mathcal{T}$ to itself for which we use postfix notation in such a way that $f(t_1,\ldots,t_n)\sigma = f(t_1\sigma,\ldots,t_n\sigma)$, for each $n \in N, f \in \mathcal{F}_n, i \in [1..n],$ *and* $t_i \in \mathcal{T}$. By a ground substitution, we mean a substitution γ such that $x\gamma$ is ground for all x in $\mathcal{X}$. We use postfix juxtaposition for composition of substitutions, and Σ (resp. Γ) for the set of all (resp. ground) substitutions.

A term s is *more general than* (*subsumes*) a term t, writen $s \mathrel{\underline{\gtrdot}} t$, if $s\sigma = t$ for some substitution σ called a *match* from s to t; we also say that t is an *instance* of s. The relation $\mathrel{\underline{\gtrdot}}$ is a quasi-ordering on terms, called *subsumption*, whose associated equivalence $\doteq$ is called *renaming* (*conversion*).

Subsumption and the subterm ordering are special cases of the *encompassment* quasi-ordering, in which $s \trianglelefteq t$ if a subterm of t is an instance of s, that is $s\sigma = t|_p$ for some $p \in \mathcal{P}os(t)$ and $\sigma \in \Sigma$.

We say that a substitution σ unifies two terms s and t iff $s\sigma = t\sigma$. A substitution σ is *more general* than (*subsumes*) a substitution τ, or τ is an *instance* of σ, and we write $\sigma \lesssim \tau$, if there exists a substitution θ such that $\sigma\theta = \tau$.

The following key result is due to Herbrand [30] and Robinson [67]:

Theorem 1. *If two terms s and t are unifiable, then they have a most general unifier $mgu(s,t)$ which is unique up to renaming.*

3.3 Lexicography

To fix nomenclature, the letters a through h will be used for function symbols; l, r, and s through w will denote arbitrary terms; x, y, and z will be reserved for variables; p and q, for positions; lower case Greek letters, for substitutions.

Binary relations will frequently be denoted by arrows of one kind or another. If $\rightarrow$ is a binary relation, then $\leftarrow$ is its inverse, $\leftrightarrow$ is its symmetric closure ($\leftarrow \cup \rightarrow$), $\xrightarrow{=}$ is its reflexive closure ($\rightarrow \cup =$), $\xrightarrow{*}$ is its reflexive-transitive closure ($\rightarrow \circ \cdots \circ \rightarrow$) and $\xrightarrow{+}$ is its transitive closure ($\rightarrow \circ \xrightarrow{*}$).

3.4 Rewrite relations

A binary relation $\rightarrow$ over a set of terms $\mathcal{T}$ is a *rewrite relation* if it is closed both under context application and instantiation, that is

$$\forall s,t,u \in \mathcal{T}, \forall p \in \mathcal{P}os(t), \forall \sigma \in \Sigma, s \rightarrow t \Rightarrow u[s\sigma]_p \rightarrow u[t\sigma]_p.$$

The inverse, symmetric closure, reflexive closure, and transitive closure of any rewrite relation are also rewrite relations.
An ordering on terms which satisfies the above property will be called a *rewrite ordering*. An equivalence on terms which satisfies the above property is *congruence*.

Note that encompassment lies at the heart of the above definition of a rewrite relation, since a term u is rewritable iff it encompasses some rewritable term s.

Rewrite relations on terms can be generated by equations, that is unordered rewritable pairs of terms, if they are symmetric or else by rewrite rules, that is ordered rewritable pairs of terms. So, rewrite rules really differ from equations in their use.

3.5 Equations

Equation are pairs of terms written $l \simeq r$. Let E be a set of equations.

A term s rewrites to a term t at position p using an equation $l \simeq r \in E$, if there exist a substitution σ such that $s|_p = l\sigma$, and $t = s[r\sigma]_p$, or $s|_p = r\sigma$, and $t = s[l\sigma]_p$. The *equational proof step* between s and t is written $s \xleftrightarrow[l\simeq r]{p} t$ in the first case, and $s \xleftrightarrow[r\simeq l]{p} t$ in the second. We may drop the subscripts or superscripts, use E as a subscript if the actual equation is not important, as well as describe a position by a predicate if necessary. The sequence $s \xleftrightarrow[E]{*} t$ of equational proof steps between

s and t is called a proof, and $\xleftrightarrow[E]{*}$ is therefore the congruence relation generated by E.

For example, given the signature $0 \in \mathcal{F}_0, S \in \mathcal{F}_1, + \in \mathcal{F}_2$ for specifying the natural numbers with addition, and the usual equations for defining the addition of two natural numbers, we have the equational proof:

$$S(x) + 0 \xleftrightarrow[x+0 \simeq x]{\Lambda} S(x) \xleftrightarrow[x+0 \simeq x]{1} S(x+0) \xleftrightarrow[x+S(y) \simeq S(x+y)]{\Lambda} x + S(0)$$

The semantics of equations is a particular case of the semantics of first-order predicate logic in which the only predicate symbol is interpreted as the equality. Call an $\mathcal{F}$-*Algebra* $\mathcal{A}$ a pair made of a *domain* (*carrier*) A and a set of *operations* $\{f_A\}_{f \in \mathcal{F}}$ on A. A homomorphism from an $\mathcal{F}$-*Algebra* $\mathcal{A}$ to another $\mathcal{F}$-*Algebra* $\mathcal{B}$ is any application from A to B which preserves the operations, that is it satisfies the property $h(f_A(e_1, \ldots, e_n)) = f_B(h(e_1), \ldots, h(e_n))$. Homomorphisms from $\mathcal{A}$ to $\mathcal{A}$ are called *endomorphisms*, of which bijective ones are called *isomorphisms*.

A particular $\mathcal{F}$-algebra can be constructed from the signature $\mathcal{F}$ which plays a crucial role, and is called the *term* $\mathcal{F}$-*algebra* over $\mathcal{X}$. Its domain is the set of term $\mathcal{T}(\mathcal{F}, \mathcal{X})$ (hereafter written $\mathcal{T}$), and its operations are defined by $f_{\mathcal{T}}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$. Endomorphisms of the term algebra are indeed substitutions. Ground substitutions are homomorphisms from the term algebra to the ground term algebra, denoted by $\mathcal{G}(\mathcal{F})$ whose domain is the set of ground terms.

The term algebra $\mathcal{T}$ is also called the *free* algebra. It enjoys an important property: any application from $\mathcal{X}$ to the domain A of any $\mathcal{F}$-algebra $\mathcal{A}$ can be uniquely extended into a homomorphism from the free algebra $\mathcal{T}$ to $\mathcal{A}$. As a consequence, there is a unique homomorphism from the ground term algebra $\mathcal{G}$ (also called the *initial* $\mathcal{F}$-algebra) to any $\mathcal{F}$-algebra $\mathcal{A}$.

Homomorphisms allow to interpret formulae in a particular algebra. We say that $\mathcal{A}$ *satisfies* (is a *model* of) the equation $u = v$ iff $h(u) = h(v)$. We use the notation $\mathcal{A} \models u = v$, and denote by $\mathcal{M}(\mathcal{E})$ the class of models of E, that is the class of algebras which are models of each equation in E.

We now construct a particular model of E, called the *quotient algebra* and denoted $\mathcal{T}/\xleftrightarrow[E]{*}$, whose domain $\{[t]_E\}_{t \in \mathcal{T}}$ is made of the congruence classes defined in the term algebra by the congruence $\xleftrightarrow[E]{*}$ generated by the set E of equations, and whose operations are defined as expected by $f_{\mathcal{T}/\xleftrightarrow[E]{*}}([t_1]_E, \ldots, [t_1]_E) = [f(t_1, \ldots, t_n)]_E$. The quotient algebra is the free algebra (in the above sense) of $\mathcal{M}(\mathcal{E})$, and its subalgebra $\mathcal{G}/\xleftrightarrow[E]{*}$ is the initial one, also called the *standard model*. Satisfaction in the free algebra is actually equivalent to satisfaction in $\mathcal{M}(\mathcal{E})$, a result of Birkhoff [6]:

Theorem 2. $\mathcal{M}(\mathcal{E}) \models u = v$ *iff* $\mathcal{T}/\xleftrightarrow[E]{*} \models u = v$.

Let us again consider the previous example of natural numbers with addition, and define two algebras which are models of the axioms for addition:

For $\mathcal{A}$, we take the standard model, in which the domain is the set $\{0, 1, 2, \ldots\}$ of natural numbers, and the operations are $\{zero, succ, plus\}$ defined as expected.
For $\mathcal{B}$, we take a non-standard model, whose domain $\{0, 1, 2, \ldots, \infty, *\}$ includes the set of natural numbers, and whose operations $\{zero, succ, plus\}$ need the additional definitions $succ(\infty) = \infty, succ(*) = *, plus(x, \infty) = *, plus(x, *) = \infty, plus(\infty, y) = \infty$, and $plus(*, y) = *$.
It can be routinely checked that both $\mathcal{A}$ and $\mathcal{B}$ are models of the axioms $x + 0 \simeq x$ and $x + S(y) \simeq S(x + y)$. On the other hand, the standard model satisfies the commutativity axiom for $+$, while the other one does not. Indeed, it can be proved that the set of equations valid in the standard model is exactly the set of equations which can be proved by using replacement of equals by equals and structural induction over ground terms. We therefore call *inductive equalities* the equations in the following set:

$$Ind(E) = \{u = v : \mathcal{G}/\underset{E}{\overset{*}{\longleftrightarrow}} \models u = v\}$$

3.6 Rules

Rewrite rule are pairs of terms written $l \rightarrow r$. Let R be a set of rewrite rules.

A term s rewrites to a term t at position p using a rule $l \rightarrow r$ if there exist a substitution σ such that $s|_p$ is the *redex* $l\sigma$, and $t = s[r\sigma]_p$. We write $s \xrightarrow[l\rightarrow r]{p} t$ or $t \xleftarrow[l\rightarrow r]{p} s$. We may also write $s \underset{R}{\longleftrightarrow} t$ for $s \underset{R}{\longrightarrow} \cup \underset{R}{\longleftarrow} t$, hence using rules as equations. The *derivation relation* $\xrightarrow[R]{*}$ is the reflexive transitive closure of the above rewrite relation, while $\underset{R}{\overset{*}{\longleftrightarrow}}$ is the congruence relation generated by R. The sequence of rewrite steps $s \underset{R}{\overset{*}{\longleftrightarrow}} t$ is called a *proof.*

For example, given the signature $0 \in \mathcal{F}_0, S \in \mathcal{F}_1, + \in \mathcal{F}_2$ for specifying the natural numbers with addition, and the usual rules for defining the addition of two natural numbers, we have the proofs:

$$S(x) + 0 \xrightarrow[x+0\rightarrow x]{\Lambda} S(x) \xleftarrow[x+0\rightarrow x]{1} S(x + 0) \xleftarrow[x+S(y)\rightarrow S(x+y)]{\Lambda} x + S(0)$$

Definition 3. We say that s and t are *convertible* when there is a proof $s \underset{R}{\overset{*}{\longleftrightarrow}} t$. Proof shapes are depicted by appropriate names: $s \xrightarrow[R]{*} u \xleftarrow[R]{*} t$ is called a *valley* or *rewrite proof,* in which case we say that s and t are *joinable*, $s \xrightarrow[R]{=} u \xleftarrow[R]{=} t$ is a *flat* or *diamond proof,* $u \xleftarrow[R]{} s \xrightarrow[R]{} v$ is a *peak*, and $u \xleftarrow[R]{*} s \xrightarrow[R]{*} v$ is a *mountain.*

Note that a rewrite proof is simply a proof with no peak. The proof in the previous example is a rewrite proof.

We say that a term s is reducible if it contains a redex, and *irreducible* (in *normal form*) otherwise. We denote by $s\downarrow$ a normal form for s, that is an irreducible term t such that $s \xrightarrow[R]{*} t$, and we say that R is functional (*normalizing*) if every term has a unique computable R-normal form.

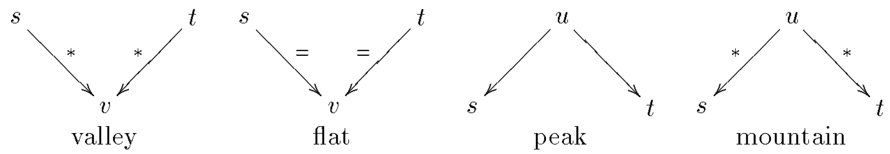


Fig. 4. Proof Shapes.

When R is functional, all terms belonging to a same class of $\xleftrightarrow[R]{*}$ have the same normal form, as can be easily shown by induction on the length of proofs. This normal form can be used as a representative of the class. We can therefore construct a variant of the quotient algebra, called the *normal-form algebra* and denoted $\mathcal{T}{\downarrow_{\mathcal{R}}}$, whose domain is the set $\{t{\downarrow}\}_{t \in \mathcal{T}}$ of terms in normal form, and whose operations are defined as expected by $f_{\mathcal{T}\downarrow_{\mathcal{R}}}(t_1{\downarrow}, \ldots, t_n{\downarrow}) = f(t_1, \ldots, t_n){\downarrow}$. This algebra is clearly isomorphic to the quotient algebra, hence

$$\mathcal{M}(\mathcal{E}) \models u = v \;\; iff \;\; \mathcal{T}/_R \models u = v \;\; iff \;\; u{\downarrow} = v{\downarrow} \; .$$

Considering now the subalgebra $\mathcal{G}{\downarrow_{\mathcal{R}}}$ whose domain is the set of ground terms in normal form, it is therefore isomorphic to the initial algebra defined by R:

$$\mathcal{G}/_{\xleftrightarrow[R]{*}} \models u = v \;\; iff \;\; \mathcal{G}{\downarrow_R} \models u = v \;\; iff \;\; u{\downarrow} = v{\downarrow} \; .$$

As a consequence, rewriting is the main paradigm used for prototyping algebraic specifications.

4 Confluence Properties

Functionality appears therefore crucial. Unfortunately, it is undecidable and indeed very difficult to obtain. We will therefore approximate it, either from above by strengthening it, or from below, by relaxing it.

Approximating it from above yields the theory of terminating rewriting systems, in which every term posseses at least one normal form by assuming termination:

Definition 4. A relation $\longrightarrow$ is said to be *terminating* (*strongly normalizing*) if there exists no infinite sequence of related elements $t_1 \longrightarrow t_2 \longrightarrow \ldots \longrightarrow t_i \longrightarrow \ldots$.

Approximating it from below yields the theory of combinatory reduction systems, of which the lambda-calculus is a particular case. Here, we will simply require that every term that posseses a normal form possesses a unique one. But some terms may have no normal form at all, and of course, there may be non-terminating sequences of rewrites issued from a term that possesses a normal form.

In both cases, it is useful to study first abstract properties of relations, which we do now.

4.1 Abstract confluence properties

The uniqueness of normal forms, whether they exist or not, is related to a confluence property of some kind depicted on figure 5:

Definition 5. A relation $\longrightarrow$ is said to be:
Church-Rosser if every proof enjoys a rewrite proof: $\forall s,t\ s.t.\ s \overset{*}{\longleftrightarrow} t,\ \exists v\ s.t.\ s \overset{*}{\longrightarrow} v \overset{*}{\longleftarrow} t$,

confluent if every mountain enjoys a rewrite proof: $\forall u,s,t\ s.t.\ s \overset{*}{\longleftarrow} u \overset{*}{\longrightarrow} t,\ \exists v\ s.t.\ s \overset{*}{\longrightarrow} v \overset{*}{\longleftarrow} t$,

locally confluent if every peak enjoys a rewrite proof: $\forall u,s,t\ s.t.\ s \overset{*}{\longleftarrow} u \overset{*}{\longrightarrow} t,\ \exists v\ s.t.\ s \overset{*}{\longrightarrow} v \overset{*}{\longleftarrow} t$,

strongly confluent if every peak enjoys a diamond proof: $\forall u,s,t\ s.t.\ s \overset{*}{\longleftarrow} u \overset{*}{\longrightarrow} t,\ \exists v\ s.t.\ s \overset{*}{\longrightarrow} v \overset{*}{\longleftarrow} t$.

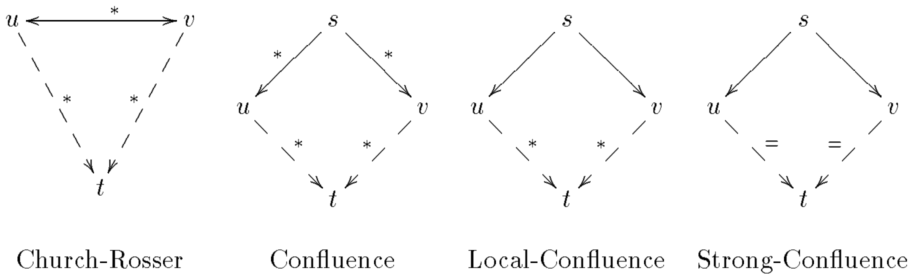


Fig. 5. Confluence Properties.

These properties are intimately related. First of all, the Church-Rosser property and confluence are equivalent as can be easily shown by induction on the length of an arbitrary proof (or better on $\longrightarrow_{mul}$ which allows a single use of the induction argument). Confluence and local confluence are not equivalent in general, as shown on the picture 6. However, they are equivalent when the relation $\longrightarrow$ is terminating, a result known as Newmann's lemma [59], whose proof by induction on $\longrightarrow$ is pictured below.

Theorem 6. *ı Assume $\longrightarrow$ is terminating. Then $\longrightarrow$ is confluent iff it is locally confluent.*

The alternative to termination is the use of strong confluence, a result due to Rosen [68] which can be proved by using a tiling argument as shown on picture 6.

Theorem 7. *A relation $\longrightarrow$ is confluent if it is strongly confluent.*

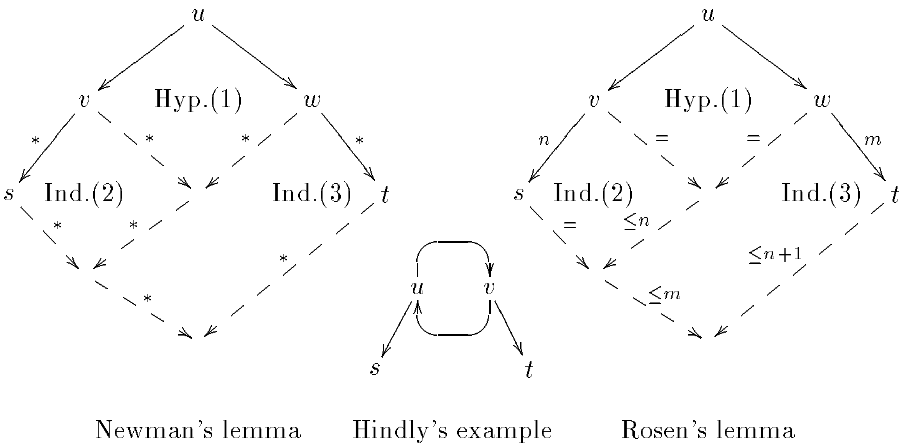


Fig. 6. Newman's and Rosen's Lemmas.

Although strong confluence appears to be very special, it is often used to prove the confluence of a relation $\longrightarrow$ by applying it to another relation $\Rightarrow$ having the same transitive closure as $\longrightarrow$. This applies in particular to the pure lambda-calculus, and more generally to combinatory reduction systems in the sense of Klop [46].

4.2 Confluence properties of rewrite relations

In practice, what really matters is to have functionality for ground terms. We therefore will distinguish two kinds of confluence properties of a rewrite relation, on $\mathcal{T}$ called *Church-Rosser* and *confluence*, and on $\mathcal{G}$ called *ground Church-Rosser* and *ground confluence*. Ground confluence, however, is undecidable even for terminating relations. In contrast, confluence can be more easily grasped, and we will see that it is indeed decidable for terminating rewrite relations.

Orthogonal Systems

Definition 8. A rewrite system R is *linear* if the left-hand side of each rule in R is linear, *overlapping* if a left-hand side unifies with a renamed non-variable subterm of any other left-hand side or with a renamed proper subterm of itself, and *orthogonal* if it is both left-linear and non-overlapping.

The importance of orthogonal systems lies in the following result:

Theorem 9 [32]. *Every orthogonal system is confluent.*

What is really shown is the strong confluence of the parallel rewrite relation, in which all redexes in a term are rewritten at once thanks to the absence of overlappings and the left-linearity of rules. Obviously, this relation has the same transitive closure as the starting one. The strong confluence proof is similar to the local confluence proof depicted on 8, but convertible terms s and t can now always be joined in at most one step from each by rewriting all redexes at once since the rewrite system is orthogonal.

Terminating Systems In this section, we restrict our attention to terminating relations. Terminating confluent systems are of course functional: any rewriting strategy will lead to the unique normal form. Besides, local confluence can itself be restricted to finitely many distinguished peaks, as shown now:

Definition 10. Let $l \to r$ and $g \to d$ be two rewrite rules (with variables renamed apart) and σ a most general unifier of g and a nonvariable subterm $l|_p$ of l. Then the peak $r\sigma \xleftarrow[l\to r]{\Lambda} l\sigma \xrightarrow[g\to d]{p} l\sigma[d\sigma]_p$ is said *critical*, and the equation $r\sigma \simeq l\sigma[d\sigma]_p$ is called a *critical pair*. We use $cp(R)$ for the set of all critical pairs between rules in R.

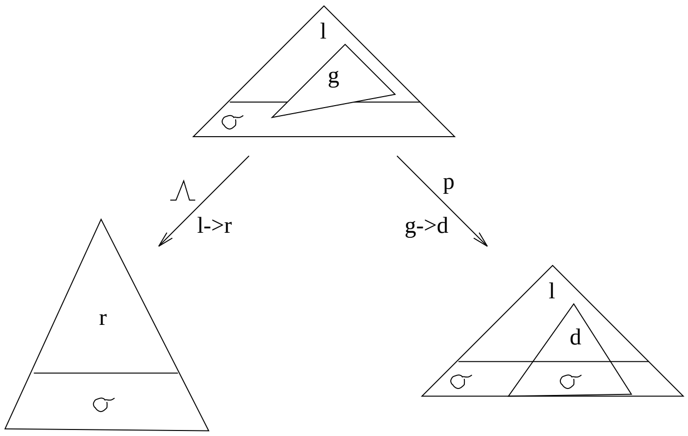


Fig. 7. Critical Pair.

Note that a finite rewrite system R has a finite set $cp(R)$ of critical pairs, and that there are "trivial" critical pairs in the above definition, when overlapping a rule with itself at the root.

Theorem 11 [47]. *A rewrite system is locally confluent if and only if all its critical pairs are joinable.*

The proof is depicted on figure 8

Therefore, a terminating system is confluent if and only if all its critical pairs are joinable, a decidable property.

We show next two examples of local-confluence proofs. For the Grecian urn which comes first figure 9, all critical pairs are joinable, hence the Grecian urn is confluent

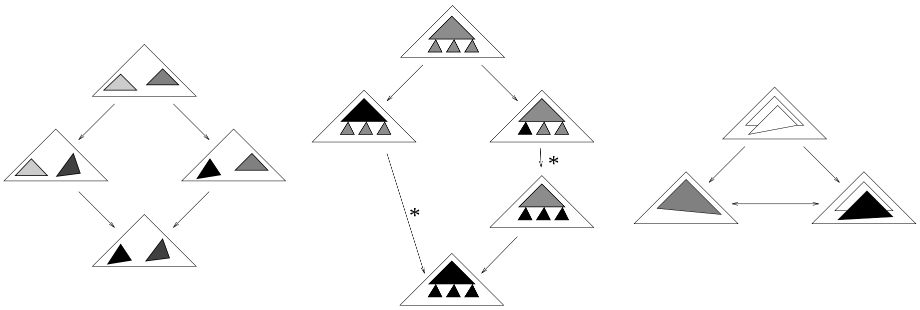


Fig. 8. Critical Pair Lemma.

since it is trivially terminating. For the group theory which comes second figure 10, there are four critical pairs of which two are confluent and the other two are not. Note how associativity overlaps itself yielding one of the two joinable critical pairs.

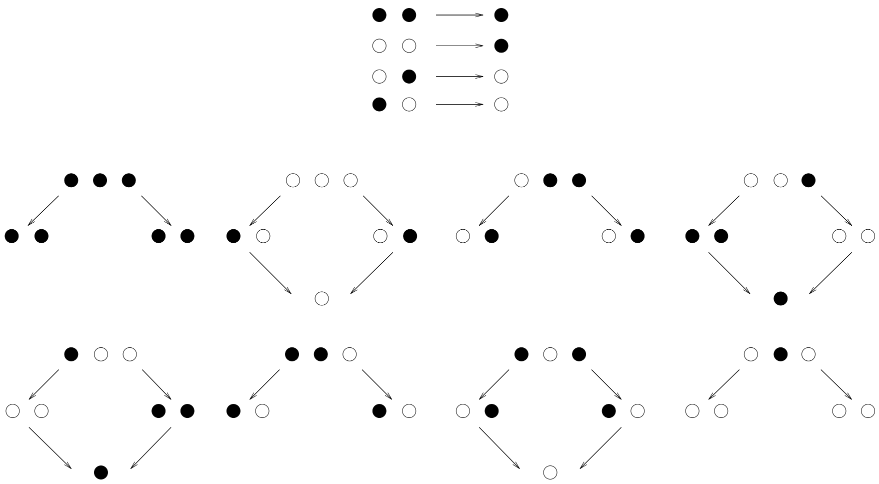


Fig. 9. Critical Pairs of the Grecian Urn

Canonical systems Rewrite systems which are both terminating and confluent are called *convergent*. We now consider an additional property in order to characterize a rewrite system under a given order.

Definition 12. A system R is *reduced* if the following terms are irreducible:

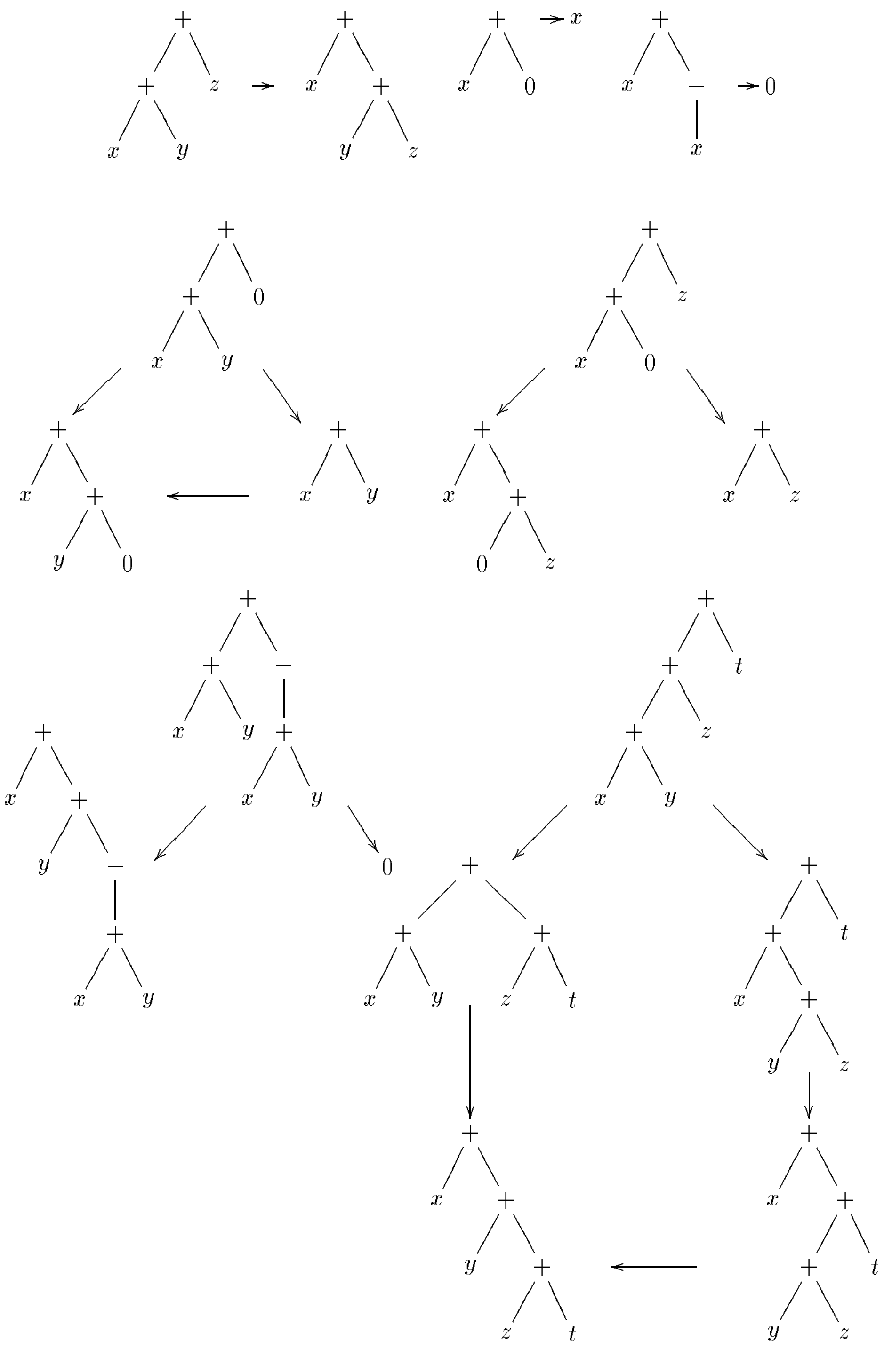


Fig. 10. Critical Pairs of Group theory.

the right-hand side r of each rule $l \to r \in R$, and
any term s properly encompassed into a left-hand side l of a rule $l \to r \in R$. A confluent, terminating, and reduced rewrite system is called *canonical*.

In particular, a strict subterm of a left-hand side must be irreducible in a reduced system. Note also that a reduced system may contain two rules $l \to r$ and $l' \to s$ such that l' is a renaming of l. In contrast, this is not possible in a canonical system due to confluence.

Canonical systems have an interesting uniqueness property, an observation first made by Lankford:

Theorem 13 [55]. *Suppose R and S are two canonical (not necessarily finite) rewrite systems having the same equational theory. Suppose further that the combined system $R \cup S$ is terminating. Then R and S must be the same (up to renaming of variables).*

This result has important consequences to be developed in section 6. Its proof is routinely obtained by expressing that each rule in R has a rewrite proof in S and vice versa.

5 Termination

In general, it is undecidable whether a system is terminating, even if both sides of all rules are monadic [35] or if it has only one left-linear rule [14]. For ground systems, however, termination is decidable [35].

In order to concentrate the termination proof on the finite number of rewrite rules rather than on the infinite number of possible rewrites, we need a special kind of ordering:

Definition 14. A *reduction ordering* on a set of terms $\mathcal{T}$ is any well-founded rewrite ordering of $\mathcal{T}$.

Termination is assured if each of the rules in R is contained in a reduction ordering; conversely, if R is terminating, then the relation $\xleftrightarrow[R]{+}$ itself is a reduction ordering.

Rewrite ordering are necessary for proving termination of rules which may apply at subterms. Rules applying always at the top, of which PROLOG is a particular case do not require well-founded orderings closed under replacement. Similarly, ground rules do not require well-founded orderings closed under instantiation. This remark is used in section 6.3.

5.1 Interpretation Orderings

We therefore need to provide with powerful enough reduction orderings that will enable us to solve the practical cases we may face with. A popular method for designing reduction orderings is by interpreting terms in some well-founded set, the set of terms being a particular case. The interpretation mapping will be a homomorphism τ from the ground terms $\mathcal{G}(\mathcal{F})$ to the well-founded set seen as an $\mathcal{F}$-algebra $\mathbf{A}$ equipped with a well-founded ordering $\succ$. The homomorphism and ordering are constrained to satisfy the following monotonicity condition: $f_\tau(\ldots x \ldots) \succ f_\tau(\ldots y \ldots)$ whenever $x \succ y$, for all f in $\mathcal{F}$ and all x, y, etc. in A. Then, the ordering $\succ_\tau$, under which $s \succ t$ if $\tau(s) \succ \tau(t)$, for s and t in $\mathcal{G}$, is well-founded.

The previous method allows us to construct reduction orderings on ground terms, which can then be lifted to open terms. By defining $s \succ t$ for arbitrary terms s and t iff $s\sigma \succ t\sigma$ for all ground substitutions σ, one obtains the finest possible such extension, but the obtained ordering may not be decidable. By adding the variables to the domain A of $\mathcal{A}$, we can map the variables in $\mathcal{X}$ to their copy in A. Then, let $s \succ_\tau t$ only if $\tau(s) \succ \tau(t)$ for all assignments of values in A to the variables in $\tau(s)$ and $\tau(t)$. A system is terminating iff such $\mathcal{A}$, τ, and $\succ$ exist.

For example, the system below, which computes the disjunctive normal form of formulae, can be shown terminating with an exponential mapping into the natural numbers:

$$\begin{aligned}
not(not(x)) &\rightarrow x \\
not(or(x,y)) &\rightarrow and(not(x), not(y)) \\
not(and(x,y)) &\rightarrow or(not(x), not(y)) \\
and(x, or(y,z)) &\rightarrow or(and(x,y), and(x,z)) \\
and(or(y,z), x) &\rightarrow or(and(y,x), and(z,x))
\end{aligned}$$

The mapping $\tau : \mathcal{T} \rightarrow \{2, 3, \ldots\}$ is defined by:

$$\begin{array}{llll}
or_\tau(x,y) & = x + y + 1 & \qquad not_\tau(x) & = 2^x \\
and_\tau(x,y) & = x \times y & \qquad c_\tau & = 2
\end{array}$$

for all x and y in $\{2, 3, \ldots\}$ and constant c in $\mathcal{F}_0$, with numbers compared under their natural ordering $>$. The second rule, for instance, always decreases the interpretation of a term, since $\tau(not(or(x,y))) = 2^{x+y+1}$ is greater than $\tau(and(not(x), not(y))) = 2^{x+y}$, for all x and y.

The use, in particular, of *polynomial interpretations* was developed in [51; 52]. Here a multivariate integer polynomial $f_\tau(x_1, \ldots, x_n)$ of degree n is associated with each n-ary symbol f in $\mathcal{F}_n$, for all n. The choice of coefficients must satisfy the monotonicity condition and ensure that terms are mapped into nonnegative integers only, as is the case, for example, when all coefficients are positive. Each rule $l \rightarrow r$ must be shown to be reducing; the polynomial $\tau(l) - \tau(r)$ must be positive for all values of variables greater than the minimal value of a ground term.

5.2 Extension Orderings

We will insist here in *divide and conquer* methods for building orderings, based on the use of functionals that preserve well-founded orderings. There are three major ones.

Lexicographic extension The *lexicographic extension* transforms n well-founded orderings $>_1, \ldots, >_n$ on n sets $S_1, \ldots, S_n$ into a well-founded ordering $(>_1, \ldots, >_n)_{lex}$ on the product $S_1 \times \ldots \times S_n$, by letting:

$$(a_1, \ldots, a_n)(>_1, \ldots, >_n)_{lex}(b_1, \ldots, b_n) \text{ if}$$

$$a_1 >_1 b_1 \text{ or } a_1 = b_1 \text{ and } (a_2, \ldots, a_n)(>_2, \ldots, >_n)_{lex}(b_2, \ldots, b_n)$$

We use to write $>_{lex}$ when all orderings $>_1, \ldots, >_n$ are the same ordering $>$. This definition extends to quasi-orderings, by using the equivalences associated to the quasi-orderings instead of the syntactic equality.

Example: $\gtrdot$ and $\trianglerighteq$ are well-founded orderings on terms.

To prove the claim, we interpret a term t by the pair $\langle |s|, |\{p \in \mathcal{P}os(t) : t|_p \in \mathcal{X}\} - \{\mathcal{V}ar(t)|\rangle$, and compare pairs in the lexicographic extension $(>_N)lex$.

Multiset Extension The *multiset extension* transforms any well-founded ordering $>$ on a set S into a well-founded ordering $\geq_{mul}$ on multisets on S. It is defined as the reflexive transitive closure of the relation between two multisets, say M_1 and M_2 such that M_2 is obtained from M_1 by replacing any element x by a finite number (possibly zero) of elements strictly smaller than x in the ordering on S.

This ordering is not yet powerful enough to show that Hercules will eventually defeat the Hydra. Actually, the Hydra is a kind of *nested multiset*. Defining the appropriate extension is left to the reader. Note however that a nested multiset is a term on a vocabulary of one varyadic symbol.

Term Extensions The *term extension*, also called *recursive path ordering* transforms any well-founded ordering $\succ$ (called a *precedence*) on an alphabet $\mathcal{F}$ of function symbols into a rewrite-ordering $\succ_{rpo}$ on terms in $\mathcal{T}(\mathcal{F})$. To define it, we must first assign a status, lexicographic or multiset to all function symbols in $\mathcal{F}$, therefore split into two subsets Lex and Mul. The induced ordering is as follows:

$$s = f(s_1, \ldots, s_m) \succeq_{rpo} g(t_1, \ldots, t_n) = t$$

if either of the following (1 or 2 or 3) hold:

(1) $s_i \succeq_{rpo} t$ for some s_i, $i \in [1..m]$
(2) $f = g$ and
case a: $f \in Lex$, and $\begin{cases} s \succ_{rpo} t_1, \ldots, s \succ_{rpo} t_n \\ (s_1, \ldots, s_n)(\succeq_{rpo})_{lex}(t_1, \ldots, t_n) \end{cases}$
case b: $f \in Mul$, and $(s_1, \ldots, s_n)(\succeq_{rpo})_{mul}(t_1, \ldots, t_n)$.
(3) $f \succ g$ and $s \succ_{rpo} t_i$ for all $i \in [1..n]$;

where $s \succ t$ if $s \succeq t$, but $s \not\preceq t$. When the precedence on function symbols is total, then the recursive path ordering is total. The ordering is strict if there are no multiset symbols, and its equivalence is generated by Malcev's permutative axioms for the multiset symbols otherwise.

See [20] for a more general version of this ordering mixing syntactic and semantic considerations.

As previously discussed for interpretation orderings, there are two ways to extend the recursive path ordering to a rewrite ordering on terms with variables. The finest extension turns out to be decidable [11; 42] but NP-hard. The other (polynomial) one is obtained by considering the variables as new constants which do not compare with other function symbols nor between themselves in the precedence.

For example, we can prove the termination of the Ackerman's function by using the lexicographic version of the recursive path ordering:

$$\begin{aligned} ack(0, y) &\to succ(y) \\ ack(succ(x), 0) &\to ack(x, succ(0)) \\ ack(succ(x), succ(y)) &\to ack(x, ack(succ(x), y)) \end{aligned}$$

For example, the third rule is contained in $>_{lpo}$ since x occurs in $succ(x)$ and $ack(succ(x), succ(y))$ is lexicographically greater than $ack(succ(x), y)$.

We can also prove termination of the first set of rules of section 2.9 by letting: $sort \succ insert \succ cons \succ max \succ min \succ S$. It is actually not necessary to compare $cons$, S and 0, nor is it actually necessary to compare max and min.

Trees are Ordinals This suggestive title is again due to Dershowitz [19]. Since the term ordering is total on $\mathcal{G}$ when all statuses are lexicographic, and the precedence is total, then the set of ground terms becomes totally ordered, hence is isomorphic to some initial segment of the ordinals. This correspondence between trees and ordinals is pictured in figure 11 in the case of a vocabulary of three symbols $*$ (binary), S (unary) and 0 (a constant) ordered by decreasing arity.

Although the order-type of this ordering is rather small [22], it can be made bigger by using trees as nodes, and so on ... It can be made even bigger by using infinite regular trees. However, it never gets really big[2], and hence, it is far from being huge[3].

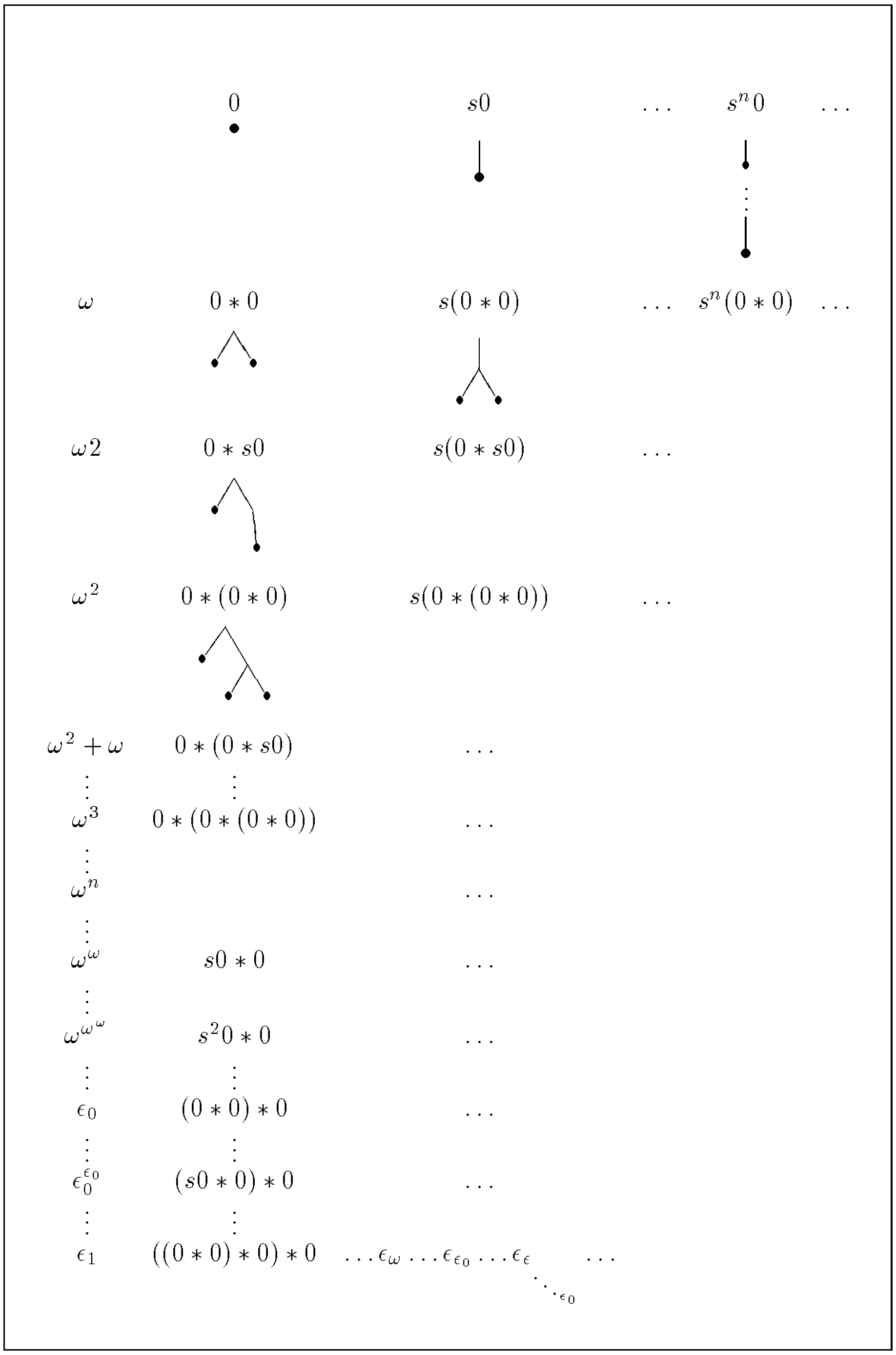
Simplification Orderings The recursive path ordering which we just discussed is a particular case of a simplification ordering:

Definition 15. A simplification ordering is a rewrite ordering $\succ$ which has the so-called *subterm property*: $\forall s \forall p \in \mathcal{P}os(s)\ p \neq \Lambda \Rightarrow s \succ s|_p$.

Note that a term is always strictly bigger than any of its strict subterms in any reduction ordering, provided they compare. Otherwise, termination would be violated: if $s|_p \succ s$, then, since $\succ$ is a rewrite ordering, $s_p \succ s = s[s_p]_p \succ s[s]_p$, and so on ... As an important consequence, total rewrite orderings are indeed simplification orderings. There is a sort of converse due to Dershowitz [17]:

Theorem 16. *Any simplification ordering on $\mathcal{G}(\mathcal{F})$ is a reduction ordering if $\mathcal{F}$ is finite.*

[2] The notion of a big ordinal was told to me by Stan Wainer.
[3] The notion of a huge ordinal was told to me by Mitsuhiro Okada.



$>_{\text{rpo}}$ with $* > s > 0, * \in \text{Lex}$

Fig. 11. Trees are ordinals [19]

Virtually all orderings used in rewriting-systems termination profs are reduction orderings. For example, the recursive path ordering is a simplification ordering, which proves it is a reduction ordering.

The proof of the above theorem is based on a powerful combinatorial result known as Kruskal's theorem, which says that the set of terms (provided $\mathcal{F}$ is finite) is well ordered by the tree-embedding relationship, in which s is *embedded* in t if it can be obtained from t by repeatedly replacing a subterm u of t by any of the strict subterms of u. It is easy to see that any simplification ordering contains the embedding. Since, by definition, an infinite sequence $\{t_i\}_{i \in N}$ of terms is well-ordered iff it contains two indices $i < j$ such that t_i is embedded in t_j, were this sequence a strictly decreasing sequence for some simplification ordering $\succ$, then we would have $t_j \succeq t_i$ by the above remark, a contradiction.

A quite elegant proof of Kruskal's theorem is due to Nash-Williams, and based on a minimal counter-sequence argument, see, e.g., [15; 17]. The finiteness condition on $\mathcal{F}$ can be relaxed by choosing particular simplification orderings, e.g. the recursive path ordering.

5.3 Union Orderings

The most powerful techniques cannot suffice in general, since termination is undecidable. When all fails, including mimicking arguments borrowed from your colleagues, it is time for trying a few tricks. Among all of them, let us stress a one.

Say that $\underset{1}{\longrightarrow}$ commutes over $\underset{2}{\longrightarrow}$ if $s \underset{1}{\longrightarrow} t \underset{2}{\longrightarrow} u$ implies $s \underset{2}{\longrightarrow} \circ (\underset{1}{\longrightarrow} \cup \underset{2}{\longrightarrow})^* u$.

Lemma 17. *Assume that* $\underset{1}{\longrightarrow}$ *commutes over* $\underset{2}{\longrightarrow}$ *and that both are terminating relations. Then* $\underset{1}{\longrightarrow} \cup \underset{2}{\longrightarrow}$ *is terminating.*

The proof is simply obtained by inductively pushing the $\longrightarrow_2$ steps first in any mixed derivation. This lemma is quite useful as the basis for a divide and conquer argument. For example, it allows us to prove that given any reduction ordering $\longrightarrow$, the relation $(\longrightarrow \cup \rhd)^+$ is a well-founded ordering. It is not a reduction ordering, though, since closure under substitution fails. In contrast, replacing encompassment by subterm yields a reduction ordering.

6 Completion of a set of rules

When a terminating set of rules is not locally confluent, non-joinable critical pairs may be simplified, then oriented and added as new rules. Rules may be simplified as well in order to obtain a canonical set. This process is called Knuth-Bendix completion [47]. In completion, the axioms used are therefore in a constant state of flux; these changes are expressed as inference rules, which add a dynamic character to establishing the existence of rewrite proofs. This view of completion, then, has two main components: an inference system, used in the completion process to generate new rewrite rules, and a rewrite relation on proofs, that shows how any proof can be normalized to a rewrite proof, as long as the appropriate rules have been generated.

So, adding new rules by computing critical pairs is simply a tool used to achieve our goal: to transform every proof into a rewrite proof using interreduced rules by rewriting (minimal) subproofs which do not obey the above property. According to [23], we will break the tradition by displaying the proof transformations first, and the inference rules second. Let $\succ$ be our reduction ordering used to orient rules, and $\gg$ the well-founded ordering on rules defined as follows: $s \to t \gg l \to r$ if (i) $s \trianglerighteq l$ under the encompassment ordering, or else (ii) $s \doteq l$ and $t \succ r$.

6.1 Completion Proof Rewrite Rules

$$
\begin{array}{rcl}
u \longleftarrow s \longrightarrow v & \Rightarrow & u \xrightarrow{*} \xleftarrow{*} v \\
 & & if\ u \longleftarrow s \longrightarrow v \text{ is not a critical peak} \\
u \xleftarrow[l\to r]{q} s \xrightarrow[g\to d]{q\cdot p} v & \Rightarrow & u \underset{r\sigma\simeq l\sigma[d\sigma]_p}{\overset{q}{\longleftrightarrow}} v \\
 & & if\ u \longleftarrow s \longrightarrow v \text{ is a critical peak} \\
u \underset{l\simeq r}{\overset{p}{\longleftrightarrow}} v & \Rightarrow & u \xrightarrow[l\to r]{p} v \\
 & & if\ l \succ r \\
u[l\sigma]_p \underset{l\simeq r}{\overset{p}{\longleftrightarrow}} u[r\sigma]_p & \Rightarrow & u[l\sigma]_p \xrightarrow[g\to d]{p\cdot q} u[l'\sigma]_p \underset{l'\simeq r}{\overset{p}{\longleftrightarrow}} u[r\sigma]_p \\
 & & if\ l \xrightarrow[g\to d]{q} l' \\
u \underset{s\simeq s}{\longleftrightarrow} u & \Rightarrow & u \\
u[l\sigma]_p \xrightarrow[l\to r]{p} u[r\sigma]_p & \Rightarrow & u[l\sigma]_p \xrightarrow[l\to r']{p} u[r'\sigma]_p \xleftarrow[g\to d]{p\cdot q} u[r\sigma]_p \\
 & & if\ r \xrightarrow[g\to d]{q} r' \\
u[l\sigma]_p \xrightarrow[l\to r]{p} u[r\sigma]_p & \Rightarrow & u[l\sigma]_p \xrightarrow[g\to d]{p\cdot q} u[l'\sigma]_p \underset{l'\simeq r}{\overset{p}{\longleftrightarrow}} u[r\sigma]_p \\
 & & if\ l \xrightarrow[g\to d]{q} l' \ \ and\ \ l \to r \gg g \to d
\end{array}
$$

In the sequel, we will use $\Rightarrow_{KB}$ for the rewrite relation on proofs generated by the above rules. These proof rewrite rules are built up in a very natural way. The first two express the critical pair lemma. The third is the orientation of an equation into a rule. Note that the critical pair may not yet be an axiom, so it will have to be *generated*. The fourth is the simplification of equations, which again *produces* a new equation while it eliminates an old one making the underlying process of equation and rule generation non-monotonic. The fifth eliminates equations that have become trivial after some simplifications. The sixth reduces rules on the right, in order to achieve canonicity. The last reduces rules on the left, yielding new equations that must be again checked for orientation. The extra condition for the application of this rule is required for proving termination of these proof rewrite rules. So, all proof rewrite rules except the first two change the set of axioms, hence give rise to appropriate inference rules.

6.2 Completion Inference Rules

An *inference rule* (for our purposes) is a binary relation between pairs $(E; R)$, where E is a set of equations and R is a set of rewrite rules. We define the following set KB of six inference rules:

Deduce:	$(E;R) \vdash (E \cup \{s \simeq t\}; R)$	if $s = t \in \mathbf{cp}(R)$
Orient:	$(E \cup \{s \simeq t\}; R) \vdash (E; R \cup \{s \rightarrow t\})$	if $s \succ t$
Simplify:	$(E \cup \{s \simeq t\}; R) \vdash (E \cup \{s \simeq u\}; R)$	if $t \rightarrow_R u$
Delete:	$(E \cup \{s \simeq s\}; R) \vdash (E; R)$	
Compose:	$(E; R \cup \{s \rightarrow t\}) \vdash (E; R \cup \{s \rightarrow u\})$	if $t \rightarrow_R u$
Collapse:	$(E; R \cup \{s \rightarrow t\}) \vdash (E \cup \{u \simeq t\}; R)$	if $s \xrightarrow[l \rightarrow r]{p} u$ with $s \rightarrow t \gg l \rightarrow r$

The role of these inference rules is to make sure that if a left-hand side of a proof rewrite rule matches a particular proof, then that proof can be rewritten. This may not be the case unless E and R contain the additional rules and equations which are needed in the right-hand side of the proof rewrite rule. Adding these equations and rules is the role of the inference rules.

We write $(E;R) \vdash_{KB} (E';R')$ if the latter may be obtained from the former by one application of a rule in KB.

6.3 Termination

Given a starting set E of equations, let us call $R*$ and $E*$ the smallest set of equations and rules which is closed under the above inference rules. To this set, we can associate the corresponding infinite set of proof rewrite rules defined in section 6.1. The role of these rules is to normalize proofs. Hence, their termination is a crucial property. To this end, let us define an ordering on proofs in which a proof is interpreted as a multiset of elementary proof steps, and a proof step is in turn interpreted by a pair of a multiset and a term as follows:

the equational proof step $s \longleftrightarrow t$ is interpreted by the pair $\langle \{s,t\}, \perp \rangle$;

the rewrite proof step $s \xrightarrow[l \rightarrow r]{} t$ is interpreted by the pair $\langle \{s\}, l \rightarrow r \rangle$.

Pairs are compared lexicographically, using the multiset ordering $\succ_{mul}$ induced by the given reduction ordering $\succ$ for the first component, and the ordering $\gg$ on rules for the second. Multisets of pairs, measuring the complexity of proofs, are compared by the multiset extension of the ordering on pairs. The ordering on proofs is therefore

$$((\succ_{mul}, \gg)_{lex})_{mul}.$$

Note how this ordering considers the justification of proofs, and not just the terms in it. Since the ordering is built up from well-founded orderings by means of extension that preserve well-founded orderings, the obtained ordering is well-founded. Checking the rules is left as a usefull exercise.

6.4 Completion Sequences

In practice, the completion rules are usually applied in some order, yielding a *completion sequence* $(E_0; \emptyset) \vdash_{KB} (E_1; R_1) \vdash_{KB} \cdots \vdash_{KB} (E_n; R_n) \cdots$ which may be infinite (completion *diverges*) or finite in case there are no possible non-redundant inferences after some finite step n. In the latter case, completion *succeeds* if $E_n = \emptyset$, and *fails* otherwise.

For example, completion succeeds when given the following fragment of group theory:

$$x \cdot 1 \simeq x, 1 \cdot x \simeq x, x^{-} \cdot (x \cdot y) \simeq y,$$

by generating the eight-rule canonical system

$$\begin{array}{rcr} 1 \cdot x \to x & \qquad & x \cdot 1 \to x \\ x^{-} \cdot x \to 1 & & x \cdot x^{-} \to 1 \\ 1^{-} \to 1 & & (x^{-})^{-} \to x \\ x^{-} \cdot (x \cdot y) \to y & & x \cdot (x^{-} \cdot y) \to y \end{array}$$

using size as the reduction ordering. A completion sequence, in which *Simplification*, *Compose* and *Collapse* are applied eagerly is shown in table 1. In contrast, completion diverges when given the single following equation $fgfx \simeq gfx$, where f and g are both unary (prefix) operator symbols, by generating the following equations and rules:

$$\begin{array}{rclr} fgfx & \underset{1}{\to} & gfx & Orient \\ fggfx & \simeq & ggfx & Deduce(1,1) \\ fggfx & \underset{2}{\to} & ggfx & Orient \\ fgggfx & \simeq & gggfx & Deduce(2,1) \\ & \ldots & & \end{array}$$

Divergence is of course undecidable [44] Note that no trick can circumvent divergence except changing the completion ordering, since by canonicity of the obtained set of rules, there cannot exist both a finite and an infinite set of rules defining the same congruence and included in the same well-founded ordering on terms. On the other hand, there are methods for circumventing failure mentioned in section 10.

6.5 Fairness

The role of the rewrite rules on proofs is to show that every proof is eventually normalized, this is why termination of these rules is crucial, and that normal form proofs are indeed rewrite proofs. For this latter property to hold, an hypothesis is needed, called *fairness*. Let us call *persisting* a rule which is never reduced on the left nor on the right once it has been generated:

$$E_\infty = \bigcup_{i \geq 0} \bigcap_{j \geq i} E_j$$

[4] The first field, i, indicates the number of completion steps. The second field is the set of rules at step i. At step 3 for example, R_3 is made of all rules in R_2 and of the oriented rule at step 4 which is displayed below R_2. Since the set of rules does not change at step 4, steps 3 and 4 are displayed between the same horizontal lines. The third field displays the set of equations in E_i on the same line(s) as the step number i. The last field indicates the name of the inference rule used at step i.

i	R_i	E_i	inference
0		$1 \cdot x \simeq x$ $x \cdot 1 \simeq x$ $x^- \cdot (x \cdot y) \simeq y$	
1	$x \cdot 1 \to x$	$1 \cdot x \simeq x$ $x^- \cdot (x \cdot y) \simeq y$	orient
2	R_1 $1 \cdot x \to x$	$x^- \cdot (x \cdot y) \simeq y$	orient
3	R_2		orient
4	$x^- \cdot (x \cdot y) \to y$	$x^- \cdot x \simeq 1$	deduce (1,3)
5	R_3		orient
6	$x^- \cdot x \to 1$	$1^- \simeq 1$	deduce (1,5)
7	R_5		orient
8	$1^- \to 1$	$(x^-)^- \cdot y \simeq x \cdot y$	deduce (3,3)
9	R_7		orient
10	$(x^-)^- \cdot y \to x \cdot y$	$1^- \cdot y \simeq 1 \cdot y$	deduce (7,9)
11	R_9 $1^- \cdot y \to 1 \cdot y$		orient
12	R_9 $1^- \cdot y \to y$		compose (11,2)
13	R_9	$1 \cdot y \simeq y$	collapse (12,7)
14		$y \simeq y$	simplify (2)
15			delete
16		$(x^-)^- \simeq x$	deduce (1,9)
17	R_9 $(x^-)^- \to x$		orient
18	R_7	$x \cdot y \simeq x \cdot y$	collapse (9,17)
19	$1^- \cdot y \to y$		delete
20	$(x^-)^- \to x$	$x \cdot (x^- \cdot y) \simeq y$	deduce (3,17)
21	R_{18}		orient
22	$x \cdot (x^- \cdot y) \to y$	$x \cdot x^- \simeq 1$	deduce (1,21)
23	R_{21} $x \cdot x^- \to 1$		orient

Table 1. A successful completion sequence for a fragment of group theory[4]

$$R_\infty = \bigcup_{i \geq 0} \bigcap_{j \geq i} R_j$$

Then, a completion sequence is *fair* if the critical peaks formed with the persisting rules are all reducible with respect to the rewrite rules on proofs. This is in particular so, if the corresponding critical pairs have been computed, but this may not be always necessary.

If success occurs after a finite number n of steps, then the resulting system is a decision procedure for E_0. In case of divergence, a semi-decision procedure can be obtained if the completion sequence is *fair*:

Theorem 18 [33]. *Given two terms s and t such that $s \xleftrightarrow[E_0]{*} t$, and a fair completion sequence $(E_0; \emptyset) \vdash_{KB} (E_1; R_1) \vdash_{KB} \cdots \vdash_{KB} (E_n; R_n) \cdots$ such that E_∞ is empty, there exists an integer i such that s and t have the same normal form with respect to R_j for all $j \geq i$.*

The strategy described below, by a regular expression of the inference rules names, is fair:

$$((\textbf{Collapse}^*;\ \textbf{Compose}^*;\ \textbf{Simplify}^*;\ \textbf{Delete}^*; \textbf{Orient}^*)^*;\ \textbf{Deduce})^*$$

Efficiency of completion is strongly related to the possibility of eliminating redundant critical pair computations, especially in the context of completion modulo. See [2; 77] for various critical pair criteria and [4; 61; 53] for completion strategies which eliminate redundancy.

6.6 Limitations of rewriting techniques

There are two kinds of limitations. First, completion may fail in finding a canonical rewrite system even if one exists. This is so because orienting non-persisting equations may require an ordering more powerful than the one in use (which may be powerful enough to orient the set of persisting rules). Second, an equation may be generated, like commutativity, that cannot be oriented without loosing the termination property. These questions are briefly addressed in section 10.

Although rewriting appears as a very general and powerful technique for solving such problems in a uniform manner, it may not always apply: there are computable (with a decidable word problem) algebras, for which the word problem cannot be decided by means of (plain) rewriting in any presentation of the algebra, a result of Squier [70]. However, Squier's example can be solved in a conservative extension of the algebra, by choosing a different set of generators as recently shown by Lafont at the "Ecole de Printemps d'Informatique Théorique". This can of course always be done, since rewrite rules can encode Turing machines [54].

7 Solving queries

A convergent set R of rewrite rules for some equational theory E allows one to solve various kinds of queries. A query will be a logical formula built over the equality predicate, to be interpreted in some (possibly quotient of) term algebra ($\mathcal{T}(\mathcal{F})$ or $\mathcal{T}(\mathcal{F}, \mathcal{X})$) by the congruence generated by the set E (equivalently R) of axioms. Accordingly, the query will be solved in the quotient under consideration (we may simply say in $\mathcal{T}(\mathcal{F})$ or $\mathcal{T}(\mathcal{F}, \mathcal{X})$ in the following, leaving the quotient implicit). We will distinguish queries after their logical complexity. Since conjunctions and disjunctions do not cause particular problems, we will simplify our notations by forgetting about these two connectives, hence consider quantified atomic formulae.

7.1 Word problems and universal queries

Query: $s = t$, where s and t are ground.

This query is solved by computing the normal forms of s and t and comparing them. Such a query is usually called a word problem. Since the convergence property of the set R of rules yields a decision procedure for equality, we can also answer to negated such queries:

Query: $s \neq t$, where s and t are ground.

Word problems generalize easily to universal queries :

Query: $\forall \mathcal{V}ar(s,t) \;\; s = t$

and their negation:

Query: $\exists \mathcal{V}ar(s,t) \;\; s \neq t$

This is of course done as above by considering the variables as new constants. This can be justified by looking at the latter query, where existential variables can be first skolemized into constants, and we are back to a ground query. This remark applies to the former query, by using a double negation. Note that the equality predicate is implicitely interpreted here in $\mathcal{T}(\mathcal{F}, \mathcal{X})$, an algebra with infinitely many free constants. Adding new constants in an arbitrary algebra may not be sound: there are indeed pathological algebras with a decidable word problem for which universal queries are undecidable.

7.2 Existential queries

These queries may have free variables as well as existentially quantified ones:

Query: $\exists X \;\; s = t$, where $X \subseteq \mathcal{V}ar(s,t)$

The goal here is to find (ground) values for the free variables that make the formula true in the quotient of the Herbrand universe by the congruence generated by a convergent set of rules R. Variables in X are auxiliary variables (*parameters*) whole value is useless. The values of the free variables are supposed to be enumerated by the resolution process, or represented by means of a so-called *solved form*, see [37]. This will be achieved by a set of non-deterministic transformation rules that simplify the query until a solved form is obtained on each branch.

Due to the existence of a rewrite proof for an arbitrary valid equation, instead of solving a goal

$$\exists X \;\; s = t,$$

one solves a pair of "directed" goals,

$$\exists X \exists z \;\; s \rightarrow z \wedge t \rightarrow z, \text{ for some new variable } z.$$

The coming rules transform such a query into a substitution encoded as a formula of the form $x_1 \rightarrow u_1 \wedge \ldots \wedge x_n \rightarrow u_n$, where the x_i are distinct free variables of the query, and do not belong to any u_j.

$$
\begin{array}{lll}
\text{Decompose:} & f(\vec{s}) \to f(\vec{t}) \Rightarrow \bigwedge_{i=1}^{i=n} s_i \to t_i & \\
\text{Delete:} & s \to s \Rightarrow T & \\
\text{Eliminate:} & x \to s \wedge P \Rightarrow x \to s \wedge P\{x \mapsto s\} & \\
 & \qquad \text{if } x \in \mathcal{X}, \text{ and } x \notin \mathcal{V}ar(s) & \\
\text{Mutate:} & f(\vec{s}) \to t \Rightarrow s_1 \to l_1 \wedge \ldots \wedge s_n \to l_n \wedge r \to t & \\
 & \qquad \text{if } f(l_1, \ldots, l_n) \to r \text{ is a copy of a rule in } R \text{ with} & \\
 & \qquad \text{its variables renamed apart } \mathcal{V}ar(P, t, s_1, \ldots, s_n) & \\
\text{Imitate:} & f(\vec{s}) \to y \Rightarrow \exists y_1, \ldots, y_n \;\; y \to f(y_1, \ldots, y_n) \wedge \bigwedge_{i=1}^{i=n} s_i \to y_i & \\
 & \qquad \text{if } y \in \mathcal{X}, \;\; y_i \notin \mathcal{V}ar(y, s_1, \ldots, s_n) & \\
\text{Parameters:} & \exists X, y \;\; y \to u \wedge P \Rightarrow \exists X \;\; P & \\
 & \qquad \text{if } y \notin \mathcal{V}ar(P, u) & \\
 & \exists X, y \;\; u \to y \wedge P \Rightarrow \exists X \;\; P & \\
 & \qquad \text{if } y \notin \mathcal{V}ar(P, u) & \\
 & \exists X, y \;\; P \Rightarrow \exists X \;\; P & \\
 & \qquad \text{if } y \notin \mathcal{V}ar(P) &
\end{array}
$$

For example, if R is $\{f(x, x) \to c(x), a \to b\}$, then the query $f(y, a) = c(y)$ can be solved by the following sequence of rewrites:

$$\exists z \;\; f(y, a) \to z \wedge c(y) \to z \Rightarrow_{Mutate}$$

$$\exists z, x \;\; y \to x \wedge a \to x \wedge c(x) \to z \wedge c(y) \to z \Rightarrow_{Eliminate}$$

$$\exists z, x \;\; y \to x \wedge a \to x \wedge c(x) \to z \Rightarrow_{Imitate}$$

$$\exists z, x \;\; y \to x \wedge x \to a \wedge c(x) \to z \Rightarrow_{Eliminate}$$

$$\exists z, x \;\; y \to a \wedge x \to a \wedge c(a) \to z \Rightarrow_{Parameters}$$

$$\exists z \;\; y \to a \wedge c(a) \to z \Rightarrow_{Parameters}$$

$$y \to a$$

The solution $\{y \mapsto b\}$ can be obtained with an application of *Mutate* replacing the application of *Imitate*:

$$\exists z, x \;\; y \to x \wedge a \to x \wedge c(x) \to z \Rightarrow_{Mutate}$$

$$\exists z, x \;\; y \to x \wedge x \to b \wedge c(x) \to z \Rightarrow_{Eliminate}$$

$$\exists z, x \;\; y \to b \wedge x \to b \wedge c(b) \to z \Rightarrow_{Parameters}$$

$$\exists z \;\; y \to b \wedge c(b) \to z \Rightarrow_{Parameters}$$

$$y \to b$$

Theorem 19 [21]. *Let R be a convergent rewrite system and Q the existential query $\exists X, z \;\; s \to z \wedge t \to z$. Then, for every solution γ of Q, there exists a sequence of rewrites from Q whose result is a substitution more general than γ.*

By exhibiting more primitive transformation rules, this set of rules subsumes *narrowing*, as used for this purpose in [25]:

Definition 20. A term s *narrows* to a term t, via substitution $\sigma = mgu(s|_p, l)$, at position p using the rule $l \rightarrow r$, symbolized $s \underset{l\rightarrow r,\sigma}{\overset{p}{\leadsto}} t$, if $t = s\sigma[r\sigma]_p$.

By encoding each narrowing step by appropriate sequences of the rules, the above theorem can now be reformulated as follows: For every solution γ of the above existential query Q, there exists a narrowing derivation from $s = t$:

$$s_0 = t_0 \underset{\sigma_0}{\leadsto} s_1 = t_1 \underset{\sigma_1}{\leadsto} \ldots \underset{\sigma_{n-1}}{\leadsto} s_n = t_n$$

such that $s = s_0$, $t = t_0$, $\theta = mgu(s_n, t_n)$ and $\sigma = \sigma_1 \ldots \sigma_{n-1}\theta$ is more general than γ.

Taking for example the rules for defining the addition over natural numbers $x + 0 \rightarrow x$, $x + S(y) \rightarrow S(x + y)$, the query $S(0) + y = S(S(x))$ originates in the narrowing tree depicted on figure 12.

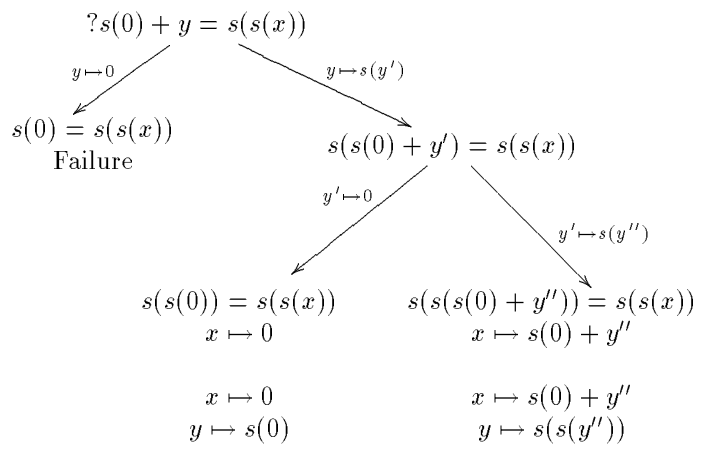


Fig. 12. Narrowing Tree.

The narrowing process resembles completion as pointed out by Dershowitz, but less critical pairs are computed. Variations on narrowing include: *normal narrowing* [25] (in which terms are normalized via $\rightarrow^!_R$ before narrowing), *basic narrowing* [36] (in which the substitution part of prior narrowings is not subsequently narrowed), and their combination [62]. Basic narrowing can be elegantly formulated by the following two inference rules operating on pairs of equations $s = t$ (assuming $=$ commutative) and substitutions θ, with $s = t, \emptyset$ as the initial pair:

Narrowing: $s = t \; ; \; \theta \;\; \Rightarrow \;\; \exists \mathcal{V}ar(l,r) s[r]_p = t \; ; \; \theta\sigma$
if σ is the most general unifier of s_p and l
Success: $s = t \; ; \; \theta \;\; \Rightarrow \;\; T \; ; \; \theta\sigma$
if σ is the most general unifier of s and t

7.3 Negative queries

The above technique extends to negative queries, by making use of ground reducibility tests:

Query: $\exists X \;\; s \neq t$, where $X \subseteq \mathcal{V}ar(s,t)$

Definition 21. An equation $s \simeq t$ is *ground reducible* iff for all ground substitutions γ, either $s\gamma = t\gamma$ or one of $s\gamma, t\gamma$ is reducible. In case $s\eta$ and $t\eta$ are different and irreducible, $s = t$ is not ground reducible *because of* η.

Narrowing: $s \neq t \; ; \; \theta \;\; \Rightarrow \;\; s' \neq t' \; ; \; \theta\sigma$
if σ is the most general unifier of s_p and l
Success: $s \neq t \; ; \; \theta \;\; \Rightarrow \;\; T \; ; \; \theta\eta$
if $s = t$ is not ground reducible because of η

Ground reducibility is indeed decidable [65; 43; 9; 7; 8], and substitutions because of which an equation is not ground reducible are enumerable [27]. The test is however NP-hard. The algorithm given in [8] has an exponential complexity with respect to the height of terms, which is proved optimal.

7.4 Inductive queries

These queries have all their variables universally quantified over ground terms. Inductive queries do not differ from universal queries at the syntactic level, but the equality predicate is interpreted in the quotient of the ground term algebra. Answering to an inductive query normally requires an induction scheme (over the term algebra). This induction scheme can actually be hidden in a completion process.

Query: $\forall X \in \mathcal{T}(\mathcal{F}) \;\; s = t$, where $X \subseteq \mathcal{V}ar(s,t)$

For convergent systems R_0, any equation between distinct ground-normal forms is considered to be inconsistent with R_0. The observation that an equation $s = t$ is valid in the initial algebra associated with R_0 iff no inconsistency follows from $R_0 \cup \{s \simeq t\}$ is the basis of the method of inductive theorem proving pioneered by Musser [58]: if there exists a ground-convergent system with the same ground normal forms as R_0, and which presents the same equational theory as $R_0 \cup \{s \simeq t\}$, then inconsistency is precluded.

Solving these queries will therefore be done by adding the equation $s = t$ to the algebra, and completing the set $R_0, \{s \simeq t\}$. To ensure that normal forms are not changed during completion, it will be enough to check the left hand sides of the added rules for ground-reducibility [16; 40].

Definition 22. A term s is *ground reducible* by R at a set of *covering positions* S iff all its ground instances by irreducible substitutions are reducible by R at a position in S.

For example, $s(s(x))$ is ground reducible by $s(s(0)) \to 0$ at the set $\{\Lambda, 1\}$ of positions.

Since we are only interested in ground-normal forms, ground-convergence will actually be enough. As a consequence, not all critical pairs are actually needed. Only those obtained by overlapping with R_0 at a set of covering positions of the generated rules are necessary. In the rules below, R denotes the set of added rules only:

$$
\begin{array}{ll}
\text{Delete:} & (E \cup \{s \simeq s\}; R) \vdash (E; R) \\
\text{Compose:} & (E; R \cup \{s \to t\}) \vdash (E; R \cup \{s \to u\}) \text{ if } t \to_{R \cup R_0} u \\
\text{Simplify:} & (E \cup \{s \simeq t\}; R) \vdash (E \cup \{s \simeq u\}; R) \text{ if } t \to_{R \cup R_0} u \\
\text{Orient:} & (E \cup \{s \simeq t\}; R) \vdash (E; R \cup \{s \to t\}) \\
& \qquad \text{if } s \succ t \text{ and } s \text{ is ground-reducible by } R_0 \\
\text{Disprove:} & (E \cup \{s \simeq t\}; R) \vdash \text{False} \\
& \qquad \text{if } s \succ t \text{ and } s \text{ is not ground-reducible by } R_0 \\
\text{Collapse:} & (E; R \cup \{s \to t\}) \vdash (E \cup \{u \simeq t\}; R) \\
& \qquad \text{if } s \longrightarrow^{p}_{l \to r} u \text{ with } s \to t \gg l \to r \\
\text{Deduce:} & (E; R) \vdash (E \cup \{s \simeq t\}; R) \\
& \qquad \text{if } s \simeq t \text{ is a critical pair of } R_0 \text{ at a covering position in } R
\end{array}
$$

Theorem 23. *An equation $s \simeq t$ is not an inductive consequence of a ground-convergent set R_0 of rules iff inductive completion returns false, provided it does not fail.*

The above result follows from [16; 40] and [28]. The associativity proof of the addition over natural numbers defined previously is classical. Noting that the position Λ in $(x+y)+z$ is a covering position (1 is another one), the associativity rule $(x+y)+z \to x+(y+z)$ generates two critical pairs with the starting rules whose confluence is shown on figure 13

Choosing the other covering position yields a divergent completion. This must of course be the case in general since membership to the inductive theory of a set of equations is co-semi-decidable.

8 Modularity

Assume given two modules, each one specifying a set $\mathcal{D}_i$ $(i = 1, 2)$ of *defined functions* with respect to a given set $\mathcal{C}_i$ of *constructors* by means of a set R_i of rewrite rules, the left-hand sides of which are headed by symbols in $\mathcal{D}_i$. The set $\mathcal{F}_i = \mathcal{C}_i \cup \mathcal{D}_i$ is called the *signature* of the module. Putting these two modules together results in the union of the two rewriting systems R_1 and R_2.

If $\mathcal{F}_1 \cap \mathcal{F}_2 = \emptyset$, their *disjoint union* is denoted by $R_1 \oplus R_2$.

If $\mathcal{F}_1 \cap \mathcal{F}_2 = \mathcal{C}_1 \cap \mathcal{C}_2 \neq \emptyset$, their *combined union* is denoted by $R_1 \cup R_2$.

If $\mathcal{F}_1 \cap \mathcal{D}_2 = \emptyset$, but $\mathcal{F}_2 \cap \mathcal{D}_1 \neq \emptyset$, their *hierarchical union* is denoted by $R_1 + R_2$. R_1 is the base, and R_2 the enrichment. There may be several successive enrichments in a hierarchical union.

Note that defined symbols are never shared.

$$R_0 \begin{cases} x + 0 & \rightarrow \quad x \\ x + s(y) & \rightarrow \quad s(x + y) \end{cases}$$

$$\underset{\substack{\uparrow \\ \text{covering position}}}{(x + y)} + z \rightarrow x + (y + z)$$

$$\begin{tikzcd}[column sep=small]
 & (x+y)+0 \arrow[dl] \arrow[dr] & \\
x+y & & x+(y+0) \arrow[ll, dashed]
\end{tikzcd}
\qquad
\begin{tikzcd}[column sep=small]
 & (x+y)+s(z) \arrow[dl] \arrow[dr] & \\
s((x+y)+z) \arrow[d, dashed] & & x+(y+s(z)) \arrow[d, dashed] \\
s(x+(y+z)) & & x+s(y+z) \arrow[ll, dashed]
\end{tikzcd}$$

Fig. 13. Associativity of +.

Since obtaining a confluent and terminating rewriting system from a set of equations may be a hard task, it is desirable that both properties are closed under these unions. A property of term rewriting systems is *modular* if it is closed under union. We are interested in particular in the modularity of termination and confluence. Of course, neither property is closed under arbitrary union. However, it is well known that confluence is modular for disjoint union, a celebrated result of Toyama [74]. This is not true of combined unions, as pointed out by Kurihara and Ohuchi [49]:

$$R_1 = \{f(x, x) \rightarrow a,\ f(x, c(x)) \rightarrow b\}, \quad R_2 = \{g \rightarrow c(g)\}$$

Both systems are confluent, but the term $f(g, g)$ has two different normal forms, a and b, in the union. The same happens with hierarchical systems.

In contrast, termination is not modular, even for disjoint unions [73]:

$$R_1 = \{f(0, 1, x) \rightarrow f(x, x, x)\}, \quad R_2 = \{g(x, y) \rightarrow x,\ g(x, y) \rightarrow y\}$$

Indeed, there is the following infinite reduction sequence:

$$f(g(0, 1), g(0, 1), g(0, 1)) \rightarrow$$

$$f(0, g(0, 1), g(0, 1)) \rightarrow f(0, 1, g(0, 1)) \rightarrow$$

$$f(g(0, 1), g(0, 1), g(0, 1)) \ldots$$

Contradicting hopes, the above example can be refined so as to have confluent definitions for both rewrite systems [24]:

$$R_1 = \{f(0,1,x) \to f(x,x,x),\ f(x,y,z) \to 2,\ 0 \to 2,\ 1 \to 2\}$$

$$R_2 = \{g(x,y,y) \to x,\ g(y,y,x) \to x\}$$

for which there is again an infinite reduction sequence originating from $f(g(0,2,1), g(0,2,1), g(0,2,1))$.

8.1 Modular properties of alien-decreasing unions

It turns out that the *existence* of unique normal forms for all terms is a modular property of disjoint unions [56; 57]. So, if two rewrite systems are both confluent and terminating, their union enjoys unique normal forms even when there are infinite reduction sequences. For example, reducing all subterms of the above sequence to 2 yields the normal form $f(2,2,2)$. Using implicitely a result of Rusinowitch [69], Kurihara and Ohuchi pointed out that a very simple strategy allows to compute this normal form: *shared rewriting* in which different occurrences of a same variable in a right-hand side of rule are shared. This is also true of combined unions [49], and it generalizes to even more complex situations [57; 26].

Definition 24. Let R_1 and R_2 be two rewrite systems over the respective signatures $\mathcal{F}_1 = \mathcal{C}_1 \cup \mathcal{D}_1$ and $\mathcal{F}_2 = \mathcal{C}_2 \cup \mathcal{D}_2$ such that $\mathcal{D}_1 \cap \mathcal{D}_2 = \emptyset$. A term in the union $\mathcal{T}(\mathcal{F}_1 \cup \mathcal{F}_2, \mathcal{X})$ is *homogeneous* if it uses symbols of $\mathcal{F}_1 - \mathcal{D}_2$ or $\mathcal{F}_2 - \mathcal{D}_1$ exclusively. Derivations using rules in $R_1 \cup R_2$ are *homogeneous* if all terms in the sequence are homogeneous with respect to the same signature.

Non-homogeneous derivations may be homogenized by computing *caps* and *aliens*. The cap of a term is its topmost homogeneous part obtained by abstracting different aliens by different variables, and its multiset of aliens subterms is made of its maximal subterms whose root does not belong to the signature of the cap. We then say that a relation $\to$ on terms in the union is *alien-decreasing* if $s \to t$ in a cap position implies
$aliens(t)\ \rhd_{mul} aliens(s)$, or
$aliens(t) =_{mul} aliens(s)$ and $cap(s) \to^{+} \cup \blacktriangleright cap(t)$.

This seemingly complex definition captures many different ones found in the literature, like conservative disjoint unions [69], in which a variable must have less occurrences in the right hand-side of a rule than in its left-hand side, conservative combined unions, shared-rewriting of combined unions [49], crosswise disjoint unions in which the function symbols occurring in the left-hand sides of one system do not occur in the right-hand sides of the other [66].

Theorem 25. *Let R_1, R_2 be terminating term rewriting systems over $T(\mathcal{D}_1 \cup \mathcal{C}, \mathcal{X})$ and $T(\mathcal{D}_2 \cup \mathcal{C}, \mathcal{X})$ respectively, where $\mathcal{D}_1 \cap \mathcal{D}_2 = \emptyset$. If $\to_{R_1}$, $\to_{R_2}$ are alien-decreasing then $\to_{R_1 \cup R_2}$ is terminating.*

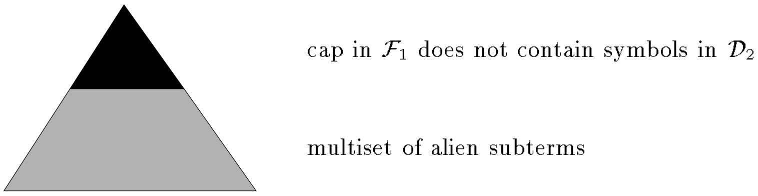


Fig. 14. Caps and Aliens

The proof is interesting by being based on a induction argument traditional in proof theory, from which we borrow some vocabulary: *For any term t and strongly normalizable substitution γ, $t\gamma$ is strongly normalizable.* The theorem follows directly, taking the identity substitution for γ. Now, γ being assumed strongly normalizable, we can use it for constructing a well-founded ordering on which the induction will operate: a term $t\gamma$ is interpreted by the pair $I(t) = \langle t, \{\gamma\}\rangle$, where $\{\gamma\}$ denotes the multiset $\{x\gamma \mid x \in Var(t)\}$. These pairs are compared in the well-founded ordering $(\trianglerighteq, (\longrightarrow_{R_1 \cup R_2} \cup \triangleright)_{mul})_{lex}$. The proof needs three cases, according to the position where $t\gamma$ reduces to $t\gamma'$, see [26]. The proof uses the property that a term may have several decompositions: given one, finding a lesser one allows to apply the induction hypothesis to the term.

Assume now that both rewrite systems R_1 and R_2 are *simply terminating*, that is, can be shown terminating by use of a simplification ordering. Let P_f be the rewrite system made of the projection rules $f(\ldots, x_i, \ldots) \to x_i$ for all $1 \leq i \leq n$. Since a set R of rules is simplifying iff $R \cup \bigcup_{f \in \mathcal{F}} P_f$ is terminating, it suffices to note that the obtained set of rules is alien-decreasing when the starting set R is, in order to prove that simple termination is a modular property of alien-decreasing unions, a result obtained by Kurihara and Ohuchi for combined unions [50].

8.2 Modular properties of hierarchical unions

We now investigate the case of hierarchical unions $R_0, R_1, \ldots, R_n$, *the base* R_0 of which is assumed alien-decreasing, while the *incremental developments* $R_1, \ldots, R_n$ define functions by generalized primitive recursive schema introduced by Jouannaud and Okada in the context of higher-order definitions [41]:

Definition 26. Let f a function symbol not belonging to the signature $\mathcal{F}$ of the rewrite system R, defined by a rewrite system R_f whose rules follow the so-called *general scheme*, that is are of the form: $f(t_1, \ldots, t_n) \to v$
where $t_1, \ldots, t_n \in T(\mathcal{F}, \mathcal{X})$, $v \in T(\mathcal{F} \cup \{f\}, \mathcal{X})$, and for all subterms $f(u_1, \ldots, u_n)$ of v, $\{t_1, \ldots, t_n\} \triangleright_{mul} \{u_1, \ldots, u_n\}$.

The general scheme imposes two restrictions on the incremental developments:

1. Since the definition of a function f uses only the symbols previously defined and f itself, mutually recursive definitions are forbidden. This restriction can be lifted by pairing mutually recursive definitions together.
2. The arguments of f in the right-hand side of a rule must be strict subterms of the arguments of f in the corresponding left-hand side.

Termination is a modular property of hierarchical systems:

Theorem 27. *Let $R = R_0 + R_1 + \ldots + R_n$ be a hierarchical system such that R_0 is terminating and alien-decreasing, and $R_1, \ldots, R_n$ are incremental developments satisfying the general scheme. Then $\rightarrow_R$ is terminating.*

The proof of the above result uses a technique similar to the one employed in the previous section. This result, however, is not as strong as we would like it to be due to the second restriction pointed out above. When it is not satisfied, tricks can be used in order to apply the theorem above in an indirect fashion, see [26].

The above results extend to higher-order definitions like those given in section 2.9. Besides higher-order rewrite rules, the combination may contain typed-lambda terms, or even polymorphically typed lambda-terms [41], or even terms of the calculus of constructions [5]. Note that such combinations are usual in proof theory: Gödel's system T is a particular case of a higher-order rewrite system following the general schema combined with the simply typed lambda-calculus. Its most powerful generalization, the Calculus of Constructions with Inductive types, is another such combination, in which the higher-order rules may *not* follow the general schema. Since this calculus has been shown confluent and terminating [76], we may think that there is still room for improving these results.

9 Rewriting and Automata

The language of ground terms in normal form plays a key role for proving inductive theorems and evaluating negated equational queries through ground reducibility. Describing the language of ground term in normal form for a given rewrite system appears therefore to be very important. A natural tool for this purpose is provided by tree automata theory, although tree grammars have also been used [9].

Inductive reducibility is also the main concept underlying the transformation of a quotient of a ground term algebra into a free order-sorted algebra by exhibiting the underlying constructors when possible [10]. Order-sorted algebras are many sorted algebras equipped with an ordering on sorts interpreted as the subset relationship on the carrier. Comon pointed out that this is equivalent to interpret sorts as recognizable subsets of the set of terms. Indeed, many-sorted signatures and non-deterministic bottom-up tree automata are different views of the same objects in which sorts are identified to states and function declarations to transitions.

Assume given the following OBJ-like [29] specification for Boolean expressions:

$$\begin{array}{ll} sorts: & N, Z, B \\ subsorts: & N < Z \\ operation: & id :\rightarrow N \\ operation: & + : Z \times Z \rightarrow Z \\ operation: & >: Z \times Z \rightarrow B \\ operation: & \vee : B \times B \rightarrow B \\ operation: & \wedge : B \times B \rightarrow B \\ operation: & \neg : B \rightarrow B \end{array}$$

This specification can be seen as the tree automaton given figure 15, which can in turn be expressed by the rules given figure 16.

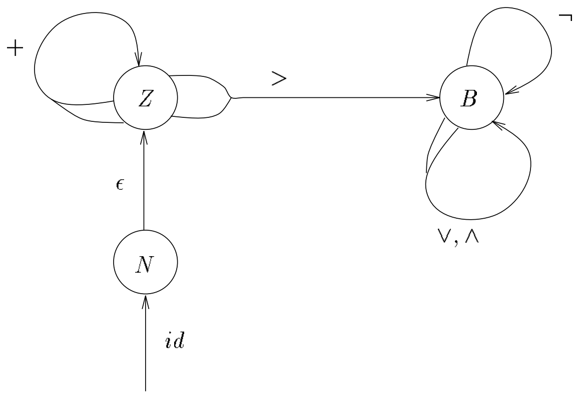


Fig. 15. Automaton for Boolean expressions

This example shows that (non-deterministic) bottom-up tree automata can also be viewed as a set of ground rules $s \rightarrow s'$ or $f(s_1, \ldots, s_n) \rightarrow s'$. The rules $s \rightarrow s'$ correspond to empty-transitions in the usual jargon of formal language theory, and to subsorts in the jargon of algebraic specifications. Non-determinism is obtained when there are critical pairs between pairs of rules of any kind.

So, bottom-up tree automata are particular rewrite systems able to type-check terms with respect to a signature. We will say that a ground term is *recognized* by an automaton if it reduces to a final state by using the rules, and that a set of terms is *regular* if it is recognized by a bottom-up tree automaton.

Bottom-up tree automata are a faithful generalization of automata on words. Non-determinism can be reduced, they can be minimized, emptiness can be decided efficiently by using cleaning techniques, pumping applies, they are closed under Boolean operations and projection, hence enjoy a logical characterization "à la Buchi", see e.g. [72] for a survey on this question.

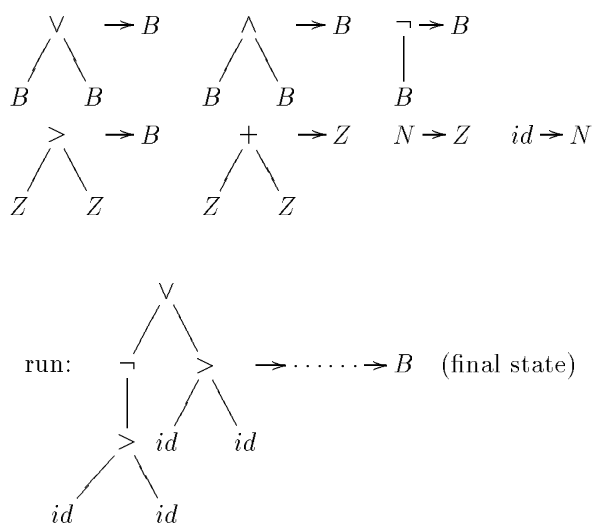


Fig. 16. Rules for recognizing Boolean expressions

Going back to the language of ground terms in normal form, it is easy to see that it is recognizable if the rewrite system is left-linear. The automaton will simply use for states the subterms of the left-hand sides of rules in order to store the information that a left-hand side of rule has been partially recognized.

For example, with the rewrite system $\{ssx \rightarrow x\}$ describing natural numbers modulo 2, we get the following automaton, whose states are $\{zero, one, reducible\}$, with $\{zero, one\}$ as final states:

In contrast, the language may not be recognizable when the rewrite system is not left-linear. For example, $\{x + x \rightarrow 0\}$ defines a non-regular language of ground normal-forms defined on figure 17.

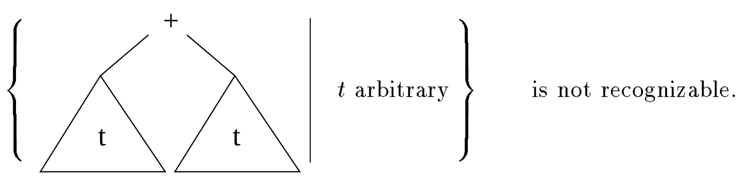


Fig. 17. A non-recognizable set of terms

The usual proof is based on a pumping lemma for regular tree languages. Adding the new rule $\{x + y \rightarrow 0\}$ makes the language regular. Fortunately, whether a finite rewrite system defines a regular laguage of ground terms in normal form is decidable [48].

We are now able to examplify the already mentioned interesting application of these automata. Consider the specification of integers as given on figure 18. This specification has an important drawback: the constructor symbols *succ* and *pred* are not free, they appear on top od left-hand sides of rules. But, since the rules are left-linear, we can indeed give an equivalent presentation obtained by computing the automaton recognizing the language of ground terms in normal form. This automaton is shown on the same figure. The state *red* in which all reducible terms are recognized is of course not final. Besides, the rules for the defined operations, if any, of the starting signature can be mechanically transformed into new rules for the transformed defined operations [10].

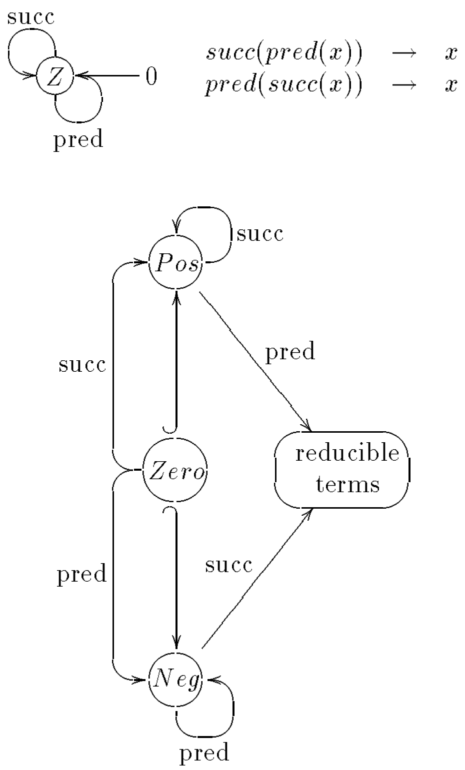


Fig. 18. Specification and automaton for the integers

Ground reducibility refers to the language of terms in normal form which are instances of a given term t. The entire set of ground terms in normal form is of course a particular case, taking a variable for t. Describing this language is the key to a decision procedure for ground reducibility.

We will first show that we can answer to our question when the rules are left-linear, even if the term t is not left-linear. By definition, t is ground-reducible iff R and $R \cup \{t \rightarrow\}$ define the same language of ground terms in normal form. If the latter is not regular, which is decidable as we have seen, then t is not ground-reducible. If it is regular, then we can compute the minimal automaton of both languages and compare them. Ground reducibility is therefore decidable when the normal form algebra has a regular carrier.

Answering the question in the general case is much more difficult. We first remark that ground reducibility can be expressed by means of a logical formula, whose set of solutions is the set of ground terms in normal form. Given $R = \{l_i \rightarrow r_i\}_{i \in [1..n]}$ such that $\mathcal{V}ar(l_i) \cap \mathcal{V}ar(l_j) = \emptyset \; if \; i \neq j$, a term t is not ground-reducible by R iff the formula

$$\exists \mathcal{V}ar(t, l_1, \ldots, l_n) \;\; x \trianglerighteq t \wedge \bigwedge_{i=1}^{i=n} \; x \not\trianglerighteq l_i.$$

has no solution in the term algebra. In this formulation of the problem, the formula is built-up with finitely many unary predicates $\trianglerighteq t, \trianglerighteq l_1, \ldots, \trianglerighteq l_n$ instead of the binary predicate $\trianglerighteq$, whose first-order language is indeed undecidable as a consequence of the undecidability of the Σ_3 fragment of the theory of subterm [75].

It turns out that this will be enough a restriction for decidability (the first-order theory of $\trianglerighteq$ can easily be shown undecidable). The method for proving decidability is based on Buchi's technique: given a formula without universal quantifiers (they can of course be replaced by negations and existential quantifications), we start by constructing an automaton recognizing the language defined by each atomic subformula; we then use the closure of the class of automata under the Boolean operations for handling conjunctions, disjunctions and negation; we finally use the closure under projection for eliminating the existential quantifiers; the encoded formula has no solution iff the automaton recognizes an empty language.

So, the main question is to find a class of automata powerful enough so as to encode the atomic formulae, which in addition is closed under the above operations, and for which the emptyness problem is decidable. This approach is taken in [7], in which the authors use bottom-up tree automata that include (some bounded number of) *equality tests* along any path leading to a final state. The actual algorithm based on pumping properties of these automata is doubly exponential as already mentioned. A simply exponential algorithm based on more sophisticated pumping techniques has been given very recently by Comon and Jacquemart [8].

The following is an automaton recognizing the language of ground terms in normal form for the single rule $f(x, f(a, x)) \rightarrow a$. A transition of the form $l \longrightarrow_{p \neq q} r$ means that the rule $l \rightarrow r$ applies provided $l|_p$ and $l|_q$ are different terms. The use of more complicated tests in which positions appear as arguments of terms with the intended meaning that $s[p]_q$ is the term $s[l|p]_q$ is only a matter of convenience, although it permits reduce the size of the automaton by an exponential factor.

Ground reducibility being an expensive test, it is worth avoiding it. Fortunately, this test is not necessary for (most) specifications whose set of function symbols can be split into constructors and defined symbols in the sens of Huet [34]: defined symbols generate inductive reducibility tests that are automatically satisfied as soon as the test applies to a term containing such a defined symbol.

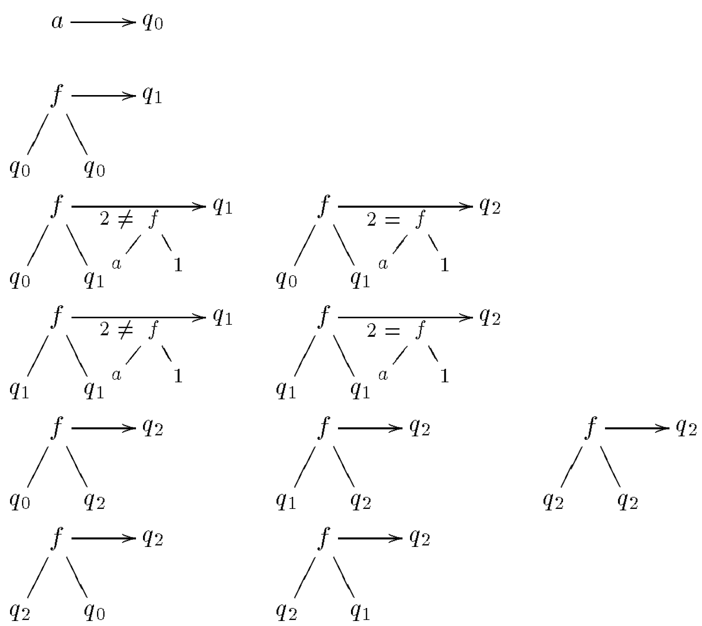


Fig. 19. Automaton with constraints

Plaisted was the first to prove the decidability of inductive reducibility [64], and Comon to consider the problem in terms of formal language theory (using grammars and reachability techniques instead of automata and pumping techniques).

10 Generalizations

The techniques developed so far require termination. We will briefly mention how to generalize them in order to accommodate more general situations, by using rewriting modulo or ordered rewriting.

Ordered rewriting uses directly the equations for rewriting, provided their instances are directed by means of an appropriate ordering.

Definition 28. A term s rewrites to a term t at position p with respect to a rewrite ordering $\succeq$ using an equation $l \simeq r \in E$, if there exists a substitution σ such that $s|_p = l\sigma, t = s[r\sigma]_p$, and $l\sigma \succ r\sigma$, or $s|_p = r\sigma, t = s[l\sigma]_p$, and $r\sigma \succ l\sigma$. We write $s \xrightarrow[l\simeq r]{p} t$ in the first case, and $s \xrightarrow[r\simeq l]{p} t$ in the second.

In the latter view of rewriting, an equation $l \simeq r$ is indeed a *rewrite rule* $l \rightarrow r$ if there are pairs s, t such that $s \xrightarrow[l\simeq r]{p} t$, but no pairs u, v such that $u \xrightarrow[r\simeq l]{p} v$.

Although commutativity can be used for ordered rewriting as defined above, for example $a+b \xrightarrow[x+y\simeq y+x]{} b+a$ if $a+b > b+a$ in the given ordering, it is in practice useful to separate the equations into two sets denoted by R and S. Equations in R will be used for rewriting. Equations in S will be used to find R-redexes by equational reasoning. This general principle known under the name of *rewriting modulo* has been proved very efficient in a number of cases by appropriately choosing R and S.

Definition 29 [63]. Let R be a set of rules and S be a set of equations. A term s rewrites to a term t at position p using a rule $l = r$ by using *extended rewriting* with S if there exist a substitution σ such that $s|_p \xleftrightarrow[S]{*} l\sigma$ and $t = s[r\sigma]_p$.

There are other definitions of rewriting modulo. In the above, a redex at position p is obtained by restricting the search to terms that differ from s by applying S-equalities under the position p only. This allows to obtain the substitution σ by S-pattern matching instead of equational search. The case where the whole term is searched is called *class-rewriting*, but its efficiency is dubious. The search can be restricted even more than in the above definition, for example to the positions of non-linear variables of the left hand side of the equation [23]. Although much more efficient (only standard pattern matching is used in conjunction with S-equality tests), the latter variant has not yet demonstrated its usefulness. Normalized rewriting is another variant of rewriting modulo in which terms must be normalized with respect to the S part before to be rewritten according to the R part. The S part is therefore itself a rewrite system, possibly modulo associativity and commutativity. An important example is given by symbolic computations over polynomials.

Termination for whatever variant of rewriting modulo S needs an *S-rewrite ordering*, that is a quasi ordering whose strict part is a rewrite ordering and whose equivalence contains the congruence generated by S. This is an area where research is still actively going on. On the other hand, most problems relating to confluence are basically solved [39; 1]. Orthogonal systems have received little attention, though. For ordered rewriting, joinability of an appropriate notion of critical pairs is a sufficient condition for confluence [31]. An interesting conjecture is the decidability of ground confluence for ordered rewriting when the ordering used is a total recursive path ordering with all statuses lexicographic.

The above definitions can be extended to more sophisticated notions of equations, such as conditional equations, order-sorted equations, and constrained equations. We refer to [21] for the definitions.

Completion has been extended to handle various situations, in particular to cope with associative-commutative axioms [63] which cannot be handled as rules. More generally, completion frameworks for rewriting modulo are described in [39; 3; 53], for ordered rewriting in [31; 60], and for constraint rewriting in [13]. Normalized completion embodies various completion methods, including the computation of a Gröbner basis of an polynomial ideal over a commutative ring [53]. Ordered completion, the variant based on ordered rewriting, has the important property that it always succeeds if it terminates. In addition, if the ordering used has the subterm property and a *canonical* set of rules exists for the considered theory, then ordered completion is guaranteed to find it, hence it is refutationally complete [1].

Acknowledgments: to Evelyne Contejean for her help with the drawings.

References

1. Leo Bachmair. *Canonical Equational Proofs*. Birkhäuser, Boston, 1991.
2. Leo Bachmair and Nachum Dershowitz. Critical pair criteria for completion. *Journal of Symbolic Computation*, 6(1):1–18, 1988.
3. Leo Bachmair, Nachum Dershowitz, and Jieh Hsiang. Orderings for equational proofs. In *Proc. 1st IEEE Symp. Logic in Computer Science, Cambridge, Mass.*, pages 346–357, June 1986.
4. Leo Bachmair, Harald Ganzinger, Christopher Lynch, and Wayne Snyder. Basic paramodulation and superposition. In Deepak Kapur, editor, *Proc. 11th Int. Conf. on Automated Deduction, Saratoga Springs, NY, LNCS 607*. Springer-Verlag, June 1992.
5. F. Barbanera, M. Fernández, and H. Geuvers. Modularity of strong normalization in the λ-algebraic-cube. 1993.
6. G. Birkhoff. On the structure of abstract algebras. In *Proc. Cambridge Phil. Society, 31*, 1935.
7. A-C. Caron, J.-L. Coquidé, and M. Dauchet. Encompassment properties and automata with constraints. In Claude Kirchner, editor, *Proc. 5th. Int. Conf. on Rewriting Techniques and Applications*, Lecture Notes in Computer Science, vol. 690, Montreal, Canada, 1993. Springer-Verlag.
8. H. Comon and Florent Jacquemard. Ground reducibility and automata with disequality constraints. In P. Enjalbert, editor, *Proc. 11th Symp. on Theoretical Aspects of Computer Science*, Paris, 1994. Springer-Verlag.
9. Hubert Comon. Unification et disunification: Théorie et applications. Thèse de Doctorat, Institut National Polytechnique de Grenoble, France, 1988.
10. Hubert Comon. Inductive proofs by specifications transformation. In *Proc. 3rd Rewriting Techniques and Applications, Chapel Hill, LNCS 355*, pages 76–91. Springer-Verlag, April 1989.
11. Hubert Comon. Solving symbolic ordering constraints. *International Journal of Foundations of Computer Science*, 1(4):387–411, 1990.
12. Hubert Comon. Disunification: a survey. In Jean-Louis Lassez and Gordon Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*. MIT Press, 1991.
13. Hubert Comon. Completion of rewrite systems with membership constraints. In W. Kuich, editor, *Proc. 19th Int. Coll. on Automata, Languages and Programming, LNCS 623*, Vienna, 1992. Springer-Verlag. An extended version is available as LRI Research Report number 699, Sept. 1991.
14. Max Dauchet. Simulation of a turing machine by a left-linear rewrite rule. In *Proc. 3rd Rewriting Techniques and Applications, Chapel Hill, LNCS 355*, 1989.
15. Nachum Dershowitz. A note on simplification orderings. *Information Processing Letters*, 9(5):212–215, November 1979.
16. Nachum Dershowitz. Applications of the Knuth-Bendix completion procedure. In *Proceedings of the Seminaire d'Informatique Theorique*, pages 95–111, Paris, France, December 1982.
17. Nachum Dershowitz. Orderings for term rewriting systems. *Theoretical Computer Science*, 17(3):279–301, March 1982.
18. Nachum Dershowitz. A taste of rewrite systems. In *Proc. Functional Programming, Concurrency, Simulation and Automated Reasonning*, 1993.
19. Nachum Dershowitz. Trees, ordinals and termination. In *Proc. CAAP 93, LNCS 668*, 1993.
20. Nachum Dershowitz and Charles Hoot. Topics in termination. In *Proc. 5th Rewriting Techniques and Applications, Montréal, LNCS 690*, 1993.

21. Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–309. North-Holland, 1990.
22. Nachum Dershowitz and Mitsuhiro Okada. Proof-theoretic techniques for term rewriting. In *Proc. 3rd IEEE Symp. Logic in Computer Science, Edinburgh*, June 1988.
23. Hervé Devie. Une approche algébrique de la réécriture de preuves équationnelles et son application à la dérivation de procédures de complétion. Thèse de Doctorat, Université de Paris-Sud, France, Octobre 1991.
24. K. Drosten. *Termerssetzungssysteme.* PhD thesis, University of Passau, Germany, 1989. In german.
25. M. Fay. First-order unification in an equational theory. In *Proc. 4th Workshop on Automated Deduction, Austin, Texas*, pages 161–167, 1979.
26. M. Fernandez and J.-P. Jouannaud. Modularity properties of term rewriting systems revisited. submitted, 1993.
27. Maribel Fernández. Narrowing based procedures for equational disunification. *Applicable Algebra in Engineering Communication and Computing*, 3:1–26, 1992.
28. Laurent Fribourg. A strong restriction of the inductive completion procedure. *Journal of Symbolic Computation*, 8:253–276, 1989.
29. J. A. Goguen, T. Winkler, José Meseguer, K. Futatsugi, and Jean-Pierre Jouannaud. *Applications of Algebraic Specifications Using OBJ*, chapter Introducing OBJ*. Cambridge University Press, 1993. D. Coleman, R. Gallimore and J. Goguen eds.
30. Jacques Herbrand. Recherches sur la théorie de la démonstration. Thèse d'Etat, Univ. Paris, 1930. Also in: Ecrits logiques de Jacques Herbrand, PUF, Paris, 1968.
31. Jieh Hsiang and Michaël Rusinowitch. On word problems in equational theories. In *Proc. in 14th ICALP Karlsruhe*, July 1987.
32. Gérard Huet. Confluent reductions: abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, October 1980.
33. Gérard Huet. A complete proof of correctness of the Knuth-Bendix completion algorithm. *Journal of Computer and System Sciences*, 23:11–21, 1981.
34. Gérard Huet and J.-M. Hullot. Proofs by induction in equational theories with constructors. *Journal of Computer and System Sciences*, 25(2), 1982.
35. Gérard Huet and Dallas S. Lankford. On the uniform halting problem for term rewriting systems. Research Report 283, INRIA, March 1978.
36. J.-M. Hullot. Canonical forms and unification. In *Proc. 5th Conf. on Automated Deduction, Les Arcs, France, LNCS 87*. Springer-Verlag, July 1980.
37. Jean-Pierre Jouannaud and Claude Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In Jean-Louis Lassez and Gordon Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*. MIT-Press, 1991.
38. Jean-Pierre Jouannaud and Hélène Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal on Computing*, 15(4), November 1986.
39. Jean-Pierre Jouannaud and Hélène Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal on Computing*, 15(4):1155–1194, 1986.
40. Jean-Pierre Jouannaud and Emmanuel Kounalis. Automatic proofs by induction in theories without constructors. *Information and Computation*, 82(1), July 1989.
41. Jean-Pierre Jouannaud and Mitsuhiro Okada. Executable higher-order algebraic specification languages. In *Proc. 6th IEEE Symp. Logic in Computer Science, Amsterdam*, pages 350–361, 1991.
42. Jean-Pierre Jouannaud and Mitsuhiro Okada. Satisfiability of systems of ordinal notations with the subterm property is decidable. In *Proc. 18th Int. Coll. on Automata, Languages and Programming, Madrid, LNCS 510*, 1991.

43. Deepak Kapur, Paliath Narendran, and Hantao Zhang. On sufficient completeness and related properties of term rewriting systems. *Acta Informatica*, 24(4):395–415, 1987.
44. Hélène Kirchner and M. Hermann. Meta-rule inference from crossed rewrite systems. Draft version, presented at CTRS, Montreal, 1990.
45. Jan Willem Klop. Term rewriting systems: a tutorial. *EATCS Bulletin*, 32:143–182, June 1987.
46. Jan Willem Klop. Term Rewriting Systems. In S. Abramsky, Dov.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 1–116. Clarendon Press, 1992.
47. Donald E. Knuth and Peter B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.
48. G. Kucherov and M. Tajine. Decidability of regularity and related properties of ground normal form languages. Research Report 92-R-062, CRIN, Nancy, France, 1992. To appear in Information and Computation.
49. Mahahito Kurihara and Azuma Ohuchi. Non-copying term rewriting and modularity of termination. Hokkaido University.
50. Mahahito Kurihara and Azuma Ohuchi. Modularity of simple termination of term rewriting systems with shared constructors. *Theoretical Computer Science*, 103:273–282, 1992.
51. Dallas S. Lankford. Canonical inference. Memo ATP-32, University of Texas at Austin, March 1975.
52. Dallas S. Lankford. On proving term rewriting systems are noetherian. Technical Report MTP-3, Mathematics Department, Louisiana Tech. Univ., 1979.
53. Claude Marché. Réécriture modulo une théorie présentée par un système convergent et décidabilité des problèmes du mot dans certains classes de théories équationnelles. Thèse de Doctorat, Université de Paris-Sud, France, 1993.
54. José Meseguer and Joseph Goguen. Initiality, induction and computability. In M. Nivat and J. Reynolds, editors, *Algebraic Methods in Semantics*, chapter 14. Cambridge Univ. Press, 1985.
55. Y. Métivier. About the rewriting systems produced by the knuth-bendix completion algorithm. *Information Processing Letters*, 16(1):31–34, 1983.
56. A. Middeldorp. Modular aspects of properties of term rewriting systems related to normal forms. In *Proc. 3rd Rewriting Techniques and Applications, Chapel Hill, LNCS 355*, pages 263–277. Springer-Verlag, 1989.
57. Aart Middeldorp. *Modular Properties of Term Rewriting Systems.* PhD thesis, Free University of Amsterdam, Netherland, 1990.
58. D. Musser. Proving inductive properties of abstract data types. In *Proc. 7th ACM Symp. Principles of Programming Languages, Las Vegas*, 1980.
59. M. H. A. Newman. On theories with a combinatorial definition of 'equivalence'. *Ann. Math.*, 43(2):223–243, 1942.
60. Robert Nieuwenhuis and Albert Rubio. Completion of first-order clauses by basic superposition with ordering constraints. Tech. report, Dept. L.S.I., Univ. Polit. Catalunya, 1991. To appear in Proc. 11th Conf. on Automated Deduction, Saratoga Springs, 1992.
61. Robert Nieuwenhuis and Albert Rubio. Basic superposition is complete. In B. Krieg-Bruckner, editor, *Proc. European Symp. on Programming, LNCS 582*, pages 371–389, Rennes, 1992. Springer-Verlag.
62. W. Nutt, P. Réty, and Gert Smolka. Basic narrowing revisited. *Journal of Symbolic Computation*, 7(3/4):295–318, 1989.

63. Gerald E. Peterson and Mark E. Stickel. Complete sets of reductions for some equational theories. *Journal of the ACM*, 28(2):233–264, April 1981.
64. David Plaisted. Semantic confluence tests and completion methods. *Information and Control*, 65:182–215, 1985.
65. David A. Plaisted. Semantic confluence tests and completion methods. *Information and Control*, 65(2-3):182–215, May/June 1985.
66. Detlef Plump. Collapsed tree rewriting: Completeness, confluence and modularity. In *Proc. Conditional Term Rewriting Systems, LNCS 656*, 1993.
67. J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.
68. Barry K. Rosen. Tree-manipulating systems and church-rosser theorems. *J. of the Association for Computing Machinery*, 20(1):160–187, January 1973.
69. Michaël Rusinowitch. On termination of the direct sum of term rewriting systems. *Information Processing Letters*, 26:65–70, 1987.
70. C. C. Squier. Word problems and a homological finiteness condition for monoids. *Journal of Pure and Applied Algebra*, 49:201–217, 1987.
71. M. Stone. The theory of representations for Boolean algebra. *Transactions of the American Mathematical Society*, 40:37–111, 1936.
72. W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 134–191. Elsevier, 1990.
73. Y. Toyama. Counterexamples to termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25:141–143, April 1987.
74. Y. Toyama. On the Church-Rosser property for the direct sum of term rewriting systems. *Journal of the ACM*, 34(1):128–143, April 1987.
75. Ralf Treinen. A new method for undecidability proofs of first order theories. Tech. Report A-09/90, Universität des Saarladandes, Saarbrücken, May 1990.
76. Benjamin Werner. *Méta-théorie du Calcul des Constructions Inductives*. Thèse Univ. Paris VII, France, 1994.
77. H. Zhang. Automated proof of ring commutativity problems by algebraic methods. *Journal of Symbolic Computation*, 9:423–427, 1990.
78. I. I. Zhegalkin. On a technique of evaluation of propositions in symbolic logic. *Matematicheskii Sbornik*, 34(1):9–27, 1927.