Tools for Proving Inductive Equalities Relative Completeness and ω-Completeness

Azzeddine LAZREK, Pierre LESCANNE, Jean-Jacques THIEL
Centre de Recherche en Informatique de Nancy
Campus Scientifique BP 239
54506 Vandœuvre-les-Nancy Cedex FRANCE

March, 4, 1989

Abstract

Inductive theorems are properties valid in the initial algebra. A now popular tool for proving them in equational theories or abstract data types is based on proof by consistency. This method uses a completion procedure and requires two essential properties of the specification, namely relative completeness and ω-completeness. This paper investigates ways of proving them. For the first one, the complement algorithm is presented. It is based on unification and computation of coverings and complements. For the second one, a technique based on discrimination of pairs of normal forms is explained and illustrated through examples.

KEYWORDS: specification, abstract data types, initial algebra, term rewriting, critical pairs, completion procedure, theorem proving, induction.

1 INTRODUCTION

Rewriting systems have many applications to computer science. They are used as a base for algebraic specifications of abstract data types [5, 13, 28]. They also provide a nice mechanism for functional programming languages. Their basic feature is a *call by matching* and they are included in actual languages like HOPE [1], ML [4, 29, 6], MIRANDA [40], OBJ [7, 9]. In this paper, our main concern is proofs of properties of operators or functions, expressed as equalities and usually based on some kind of induction [30, 10, 17, 25, 36, 32, 35, 21, 23, 18]. For example, consider the type *List*, where the constructors are $[\,]$, $\lambda x.[x], a, b$ and @. $[\,]$ is the empty list, $[x]$ is a list with one element x, a and b are two constants and @ is the append operator. These constructors satisfy the relations,

$$\begin{array}{rcl} [\,]@x & \to & x \\ x@[\,] & \to & x \\ (x@y)@z & \to & x@(y@z) \end{array}$$

defined by rewrite rules. Let us define a function $flatten$ by:

$$flatten([\,]) \to [\,]$$

$$\begin{array}{rcl}
flatten([x]) & \rightarrow & flatten(x) \\
flatten(a) & \rightarrow & a \\
flatten(b) & \rightarrow & b \\
flatten(a@x) & \rightarrow & a@flatten(x) \\
flatten(b@x) & \rightarrow & b@flatten(x) \\
flatten([x]@y) & \rightarrow & flatten(x)@flatten(y).
\end{array}$$

We may want to prove that $flatten$ is an involution, in other words,

$$flatten(flatten(x)) = flatten(x)$$

or $flatten$ is a morphism for @, that is

$$flatten(x@y) = flatten(x)@flatten(y).$$

The method, we are interested in, does not use an inference rule explicitly, but a proof by absence of contradiction or a proof by consistency [19]. Basically this requires two kinds of properties on the specification. The *relative* or *sufficient completeness* checks that the operations are completely defined. The *ω-completeness* asserts that all inductive theorems can be proved equationally. The first property has to be proved for all parts of the specification and we provide a new presentation of an algorithm based on a calculus of complements [38]. The second one is usually satisfied for one part called relations among constructors, and we investigate a method for proving it in some specific cases.

The paper is divided into three parts. In Section 2, we describe the completion-based induction principle or proof by consistency, which relies on the ω-completeness of the relations among constructors and the relative completeness of the specifications. A procedure for checking the relative completeness is described and proved in Section 3. In Section 4, ω-completeness of term rewriting systems is investigated through examples.

2 PROOF BY CONSISTENCY

2.1 Notations

Let F be a set of operators or functions, X a set of variables. Suppose that F is divided into a set C of constructors, and a set D of defined operators, therefore $F = D \oplus C$. We assume that C is not empty and that X, C and D are disjoint. *Constructors* are supposed to describe each object of the type. *Defined operators* are functions defined on the abstract data type and are supposed to disappear in the computation of the values of the function on objects of the type. In addition, we suppose that there exists only one sort. Actually, this is not a restriction, and the result of this paper can be extended to many-sorted data types.

Notations used in this paper are summarized in Figure 1.

2.2 Definitions

Let u and v be two terms of $T(F, X)$ and let us define two congruences over $T(F, X)$ as follows [15].

$\oplus$	$=$	disjoint union of sets.
$\ominus$	$=$	set difference.
$T(F,X)$	$=$	set of terms with operators in F and variables in X.
$T(F)$	$=$	set of ground terms on F, i.e., without variables.
$\Sigma(F,X)$	$=$	set of substitutions on $T(F,X)$.
$\Sigma(F)$	$=$	set of ground substitutions on $T(F)$.
$T(C,X)$	$=$	set of terms built with only constructors in C.
$T(C)$	$=$	set of ground terms built on C.
A	$=$	set of axioms i.e., pair of equivalent terms.
(F,A)	$=$	a specification with operators F and axioms A.
$M(F,A)$	$=$	class of models of (F,A) or of F-algebras that satisfy A.
$T(F,X,A)$	$=$	free algebra of $M(F,A)$ on X.
$T(F,A)$	$=$	initial algebra of $M(F,A)$.
$Var(t)$	$=$	set of variables which have at least one occurrence in t.
$G(t,C)$	$=$	set of ground instances of t by $\sigma \in \Sigma(C)$, i.e., $\{\sigma(t) \mid \sigma \in \Sigma(C)\}$.
$G(T,C)$	$=$	$\bigcup_{t\in T} G(t,C)$.
R	$=$	a term rewrite system.
$L(f,R)$	$=$	set of left-hand sides of rules of R with root f.

Figure 1: Notations used in this paper

Definition 1 (Equational equality) *$u =_A v$ or $u = v$ is an* equational theorem *in the theory generated by A if and only if $u = v$ is valid in $M(F,A)$ i.e., valid in every algebra that satisfies A. From the completeness theorem for equational deduction, this is equivalent to $u = v$ is a consequence of A, written $A \vdash u = v$. It is also equivalent to $u = v$ being valid in the free algebra $T(F,X,A)$ written $T(F,X,A) \models u = v$, where $X \supseteq Var(u) \cup Var(v)$.*

Definition 2 (Inductive equality) *$u =_{ind(A)} v$ or $u = v$ is an* inductive theorem *if and only if $u = v$ is valid in $T(F,A)$, written $T(F,A) \models u = v$. This is equivalent to $\sigma(u) =_A \sigma(v)$ for all ground substitution $\sigma \in \Sigma(F)$.*

Generally, to prove the validity of an equation in $T(F,A)$, we need inductive reasoning, whereas equational reasoning is sufficient in $T(F,X,A)$.

Example 1 Peano natural numbers. *Let $C = \{0, S\}$, $D = \{+\}$ and suppose the set NAT of axioms is,*

$$\begin{aligned} 0 + x &= x \\ S(x) + y &= S(x+y) \end{aligned}$$

The equation

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

is an equational theorem of NAT, and the following equations are inductive theorems that are not equational:

$$x + 0 = x$$

$$\begin{array}{rcl} x + S(y) & = & S(x+y) \\ x + y & = & y + x \\ x + (y+z) & = & (x+y)+z \end{array}$$

In a simple view of a structured specification of an abstract data type, the set A of axioms is divided into two subsets, A_D and A_C, as follows:

$$\begin{array}{rcl} A_D & = & \text{specification of defined operators.} \\ A_C & = & \text{set of constructor relations, possibly empty.} \end{array}$$

From a functional programming point of view, A_C defines the data structure one works on and A_D is the definition of a set of functions i.e., a functional program. The following relations are satisfied $A_C \subseteq T(C,X) \times T(C,X)$ and $A = A_C \oplus A_D$. In what follows we are interested by axioms such that

$$A_{D,f} \subseteq \bigcup_{\sigma \in \Phi} \sigma(f(x_1, \ldots, x_n)) \times T(F,X)$$

where Φ is a finite subset of $\Sigma(C,X)$.

$$A_D = \bigcup_{f \in D} A_{D,f}$$

In other words, the left-hand sides of the axioms have root f for $f \in D$ and otherwise contain only constructors. This strong restriction on the specifications are necessary for the complement method to work. Usually two important concepts are attached to structured algebraic specifications defined as follows.

Definition 3 (Relative completeness) *The specification (F,A) is complete w.r.t. (C,A_C) iff for every $u \in T(F)$ there is a $v \in T(C)$ s.t. $u =_A v$.*

Example 2 *The equational specification of $flatten$*

$$\begin{array}{rcl} flatten([\,]) & = & [\,] \\ flatten(a) & = & a \\ flatten(b) & = & b \\ flatten(a@x) & = & a@flatten(x) \\ flatten(b@x) & = & b@flatten(x) \\ flatten([x]@y) & = & flatten(x)@flatten(y) \end{array}$$

is relatively complete w.r.t. List. Indeed all the expressions containing $flatten$ can be simplified, except those of the form $flatten([x])$. However,

$$\begin{array}{rcl} flatten([x]) & = & flatten([x]@[\,]) = flatten([x])@flatten([\,]) \\ & = & flatten(x)@flatten([\,]) = flatten(x)@[\,] = flatten(x) \end{array}$$

Definition 4 (Relative consistency) *The specification (F,A) is consistent w.r.t. (C,A_C) iff for every u and v in $T(C)$ $u =_A v$ implies $u =_{A_C} v$.*

If (F, A) is a specification relatively complete and consistent w.r.t. (C, A_C), then $T(C)/=_{A_C}$ is isomorphic to $T(F)/=_A$, i.e., the algebra on the classes of ground terms modulo A_C is the algebra on the classes of ground terms modulo A. Relative completeness is called *sufficient completeness* by J. V. Guttag and J. J. Horning [12] and the *principle of definition* by G. Huet and J. M. Hullot [17].

Often one would like to be able to "decide" i.e., to prove or disprove, inductive theorems. As it will be seen later on, this is especially useful in the theory (C, A_C) of the relations among the constructors. This is possible if the inductive theory coincides with the equational theory and the latter is decidable because, for instance, it is associated with a convergent term rewriting system. This is called the *inductive completeness* by E. Paul [32]. This concept was studied first by A. Tarski and presented among others by J. Heering under the name of *ω-completeness* [14], by L. Henkin [15] from a logician's viewpoint, by W. Taylor [37] from a universal algebra viewpoint and by G. D. Plotkin [34] who shows that the $\lambda\kappa\beta\eta$-calculus is not ω-complete.

Definition 5 (ω-completeness) *The specification (F, A) is ω- complete iff every inductive theorem is an equational theorem.*

Example 3 *The following specifications (F, A) are ω-complete:*

$$F = \{0, succ, pred\} \quad A = \{succ(pred(x)) = x, pred(succ(x)) = x\} \tag{1}$$
$$F = \{0, 1, +\} \quad A = \{0 + x = x, x + y = y + x, x + (y + z) = (x + y) + z\} \tag{2}$$
$$F = \{0, opp, succ\} \quad A = \{opp(0) = 0, opp(opp(x)) = x, succ(opp(succ(x))) = opp(x)\} \tag{3}$$
$$F = \{0, a, b, .\} \quad A = \{0.x = x, x.0 = x, x.(y.z) = (x.y).z\} \tag{4}$$
$$F = \{0, [\_], @\} \quad A = \{0@x = x, x@0 = x, x@(y@z) = (x@y)@z\} \tag{5}$$
$$\begin{aligned} F = \{0, 1, +, opp\} \quad & A = \{0 + x = x, x + y = y + x, x + (y + z) = (x + y) + z, \\ & opp(0) = 0, opp(opp(x)) = x, opp(x + y) = opp(x) + opp(y)\} \end{aligned} \tag{6}$$
$$F = \{0, s, eq\} \quad A = \{eq(0, s(x)) = 0, eq(x, x) = s(0), eq(s(x), s(y)) = eq(x, y), eq(x, y) = eq(y, x)\} \tag{7}$$

proof of (1) is easy using the method shown in Section 4, Proof of (2) is from [32], Proof of (3), (4), (6) and, (7) are from Section 4 and (5) is a generalization of (4). The following specification (A, F) is not ω-complete:

$$F = \{0, 1, +\} \quad A = \{0 + x = x, x + 0 = x, x + (y + z) = (x + y) + z\}$$

because $T(F, A) \models x + y = y + x$.

2.3 The intuitive idea behind proof by consistency

Relative completeness induces a very nice method for proving inductive theorems. Indeed E is a "consequence" of a relatively complete set of axioms A if and only if $E \oplus A$ is relatively consistent w.r.t. A. This kind of theory is sometimes called *maximally consistent* [8] or Hilbert-Post complete. In this case "consequence" is w.r.t. replacement of equals by equals and induction. So inductive proofs are decomposed into two parts, a proof of relative completeness which is usually made once for all, and a proof of relative consistency. Methods for checking relative completeness are presented in Section 3. The proof of consistency is based on the existence of a ground confluent set of rules. Most of the time, ground confluence is obtained by a completion procedure, which generates rules from equations and checks their confluence. If these equations contain defined functions, this is like proving lemmas by induction. Otherwise they contain only constructors and they are proved equationally by the means of the inductive completeness. Recently, methods based on or related to inductive reducibility [18], were proposed to prove equations valid in $T(F, A)$.

3 RELATIVE COMPLETENESS

In this section, we suppose that the specification is associated with a confluent and noetherian term rewriting system R [16]. This system is usually produced from A by a completion procedure. We use the notation (F, R) or $(C \oplus D, R_C \oplus R_D)$ instead of (F, A) or $(C \oplus D, A_C \oplus A_D)$ and we use this system R to check the completeness of A. Obviously (F, R) is relatively complete w.r.t. (C, R_C) if the R-normal form of any ground term of $T(F)$ belongs to $T(C)$. This will be checked using the following concepts:

Definition 6 (Convertibility) *(F, R) is convertible to C if and only if for all ground term t in $T(F)$, there exists a term u in $T(C)$ such that $t \overset{*}{\to}_R u$. If $D = \{f\}$ we say that f is convertible.*

This is a particular case of a more general concept.

Definition 7 (Inductive reducibility) *A term t is inductively reducible or ground reducible w.r.t. a specification (F, R) if $\sigma(t)$ is R-reducible for all ground substitution $\sigma \in \Sigma(F)$.*

The convertibility is equivalent to the inductive reducibility of the terms $f(x_1, \ldots, x_n)$ for each $f \in D$. Convertibility implies relative completeness.

Lemma 1 *Given a specification (F, R) associated with a confluent and noetherian rewrite system, it is convertible to C if and only if it is relatively complete w.r.t. (C, R_C).*

> **Proof:** See [23]. □

However if the rewrite system is not confluent and noetherian, relative completeness does not imply convertibility as shown by the following example.

Example 4 *The specification*

$$\begin{aligned}
flatten([\,]) &\to [\,] \\
flatten(a) &\to a \\
flatten(b) &\to b \\
flatten(a@x) &\to a@flatten(x) \\
flatten(b@x) &\to b@flatten(x) \\
flatten([x]@y) &\to flatten(x)@flatten(y)
\end{aligned}$$

is relatively complete as mentioned in Example 2, but since, for instance, $flatten([a])$ is not reducible, it is not convertible. Indeed, the system is not confluent, but the rule

$$flatten([x]) \to flatten(x)$$

is obtained by completion.

Lemma 2 *f is convertible to C by R_D if and only if all terms of $G(f(x_1, \ldots, x_n), C) \ominus G(L(f, R_D), C)$ are R_C-reducible.*

> **Proof:** $G(f(x_1, \ldots, x_n), C)$ is the set of all ground terms with f at the root and constructors elsewhere. If every term with root f, which is not reducible by a rule in R_D, is R_C-reducible, then all ground terms with root f are R-reducible and f is convertible. □

Example 5 *Let $C = \{0, opp, succ\}$, $D = \{+\}$ and suppose the set R_C of axioms is*

$$\begin{array}{rcl} opp(0) & \rightarrow & 0 \\ opp(opp(x)) & \rightarrow & x \\ succ(opp(succ(x))) & \rightarrow & opp(x) \end{array}$$

and R_D is

$$\begin{array}{rcl} 0+x & \rightarrow & x \\ succ(x)+y & \rightarrow & succ(x+y) \\ opp(succ(x))+y & \rightarrow & opp(succ(x+opp(y))). \end{array}$$

$+$ is convertible, indeed

$$G(x_1+x_2, C) \ominus G(L(+, R_D), C) = G(opp(opp(x)) + y) \oplus G(opp(0) + y, C)$$

contains only R_C-reducible terms.

In order to use the previous lemma in practice, we introduce the following concepts.

Definition 8 (Covering) *Let M and N be two subsets of $T(F, X)$. We say that M* covers *N iff $G(N, C)$ is a subset of $G(M, C)$, i.e., each instance of a term in N is an instance of a term in M.*

Example 6 *: Let $C = \{0, S, P\}$. $\{S(x) + y, P(x) + y\}$ does not cover $\{x + y\}$ but $\{0 + y, S(x) + y, P(x) + y\}$ covers $\{x + y\}$.*

The goal of the *cover* is to find the part K of $G(f(x_1, \dots, x_n))$ which is not covered by $G(L(f, R_D), C)$ and to check that the terms of K are R_C-reducible. In practice, we cannot use this technique directly because it would induce manipulations of infinite sets of terms, so we introduce the following concept.

Definition 9 (Complement of a term) *Let t be a term of $T(C, X)$. We call any finite set K of terms s.t. $T(C) = G(t, C) \oplus G(K, C)$ a* complement *of t.*

If $t \in X$, K is empty, since $G(t, C) = T(C)$. The following proposition gives a constructive definition of a complement of a *linear* term in $T(C, X)$ which is a term with at most one occurrence of each variable.

Proposition 1 *If t is a linear term of $T(C, X)$ s.t. $t = c_j(t_1, \dots, t_{n_j})$, $C = \{c_1, \dots, c_m\}$ where n_h is the arity of c_h and*

$$\begin{array}{rl} C(t) = & \{c_i(x_1, \dots, x_{n_i}) | x_1, \dots, x_{n_i} \in X \;\&\; 1 \le i \le m \;\&\; i \ne j\} \\ & \oplus \{c_j(t_1, \dots, t_{k-1}, v, x_{k+1}, \dots, x_{n_j}) | 1 \le k \le n_j \& x_{k+1}, \dots, x_{n_j} \in X \& v \in C(t_k)\}. \end{array}$$

then $C(t)$ is a complement of t, which means $T(C) = G(t, C) \oplus G(C(t), C)$.
N.B. The variables in $t_1, \dots, t_{k-1}, v, x_{k+1}, \dots, x_{n_j}$ are supposed different.

Proof: $G(t,C) \cap G(C(t),C) = \emptyset$ is obvious. Let us prove $G(t,C) \cup G(C(t),C) = T(C)$. Suppose u is a term of $T(C)$ and $t = c_j(t_1,\ldots,t_{n_j})$. If the root of u is c_i with $i \neq j$, then $u \in G(C(t),C)$, because u is an instance of $c_i(x_1,\ldots,x_{n_i})$. Otherwise u is of the form $c_j(u_1,\ldots,u_{n_j})$. If there exists σ, s.t. $\sigma(t) = u$, then $u \in G(t,C)$. If no such σ exists, we build a substitution θ in the following way, let $k \in [1..n_j]$ s.t. $(\forall h < k)(\exists \theta_h)\theta_h(t_h) = u_h$ and there is no substitution that matches t_k with u_k, therefore there exists θ_k and $v \in C(t_k)$ with $\theta_k(v) = u_k$. Since the term t is linear, it is possible to define the substitution θ as follows:

- if x occurs in t_h with $h < k$, then $\theta(x) = \theta_h(x)$
- if x occurs in v, then $\theta(x) = \theta_k(x)$
- if $x = x_h$ with $h > k$, then $\theta(x) = u_h$.

It is now obvious that $\theta(c_j(t_1,\ldots,t_{k-1},v,x_{k+1},\ldots,x_{n_j})) = u$ and $u \in G(C(t),C)$.
□

Note that the definition of $C(t)$ given by Proposition 1 is a finite set of terms $\{t_1,\ldots,t_n\}$ such that $G(t,C), G(t_1,C),\ldots,G(t_n,C)$ is a partition of $T(C)$, which means that the $G(t_i,C)$'s and $G(t,C)$ are disjoint. Other definitions of complements are possible for other purposes [27].

Example 7 *With the conventions of the previous example.*

$$C(0) = \{S(x), P(x)\},$$

$$C(S(0)) = \{0, P(x), S(S(x)), S(P(x))\}.$$

Definition 10 (Linear Substitution) *A substitution σ with domain $Dom(\sigma)$ is linear if it satisfies the following property*

$$(\forall x \in Dom(\sigma))\sigma(x) \textit{ is linear and } (\forall y \in Dom(\sigma))\ x \neq y \Rightarrow Var(\sigma(x)) \cap Var(\sigma(y)) = \emptyset.$$

Lemma 3 *A linear substitution transforms any linear term into a linear term. If there is at least one operator with an arity larger than 2 this property characterizes linear substitutions.*

Definition 11 (Complement of a Substitution) *Let σ be a linear substitution in $T(C,X)$ and $Dom(\sigma)$ be its domain. The complement of σ is the set $C(\sigma)$ of all linear substitutions ρ such that:*

(i) $\rho \neq \sigma$.

(ii) $Dom(\rho) = Dom(\sigma)$.

(iii) $(\forall x \in Dom(\rho))\rho(x) \in C(\sigma(x))$ *or* $\rho(x) = \sigma(x)$.

Example 8 *Let* $\sigma = \{x \leftarrow 0, y \leftarrow S(y'), z \leftarrow z'\}$

$$\begin{aligned} C(\sigma) = \ & \{\{x \leftarrow 0, y \leftarrow 0, z \leftarrow z'\}, \{x \leftarrow 0, y \leftarrow P(y'), z \leftarrow z'\}, \\ & \{x \leftarrow S(x'), y \leftarrow S(y'), z \leftarrow z'\}, \{x \leftarrow S(x'), y \leftarrow 0, z \leftarrow z'\}, \\ & \{x \leftarrow S(x'), y \leftarrow P(y'), z \leftarrow z'\}, \{x \leftarrow P(x'), y \leftarrow S(y'), z \leftarrow z'\}, \\ & \{x \leftarrow P(x'), y \leftarrow 0, z \leftarrow z'\}, \{x \leftarrow P(x'), y \leftarrow P(y'), z \leftarrow z'\}\}. \end{aligned}$$

When $t \in T(C,X)$ the complement of a substitution provides a nice way to compute a basis for the set $G(t,C) \ominus G(\sigma(t),C)$. The next proposition gives an easy method for computing the complement of a term $t \in T(C,X)$.

Proposition 2 *Let t be a term and σ be a linear substitution*

$$G(t,C) = G(\sigma(t),C) \cup \bigcup_{\rho \in C(\sigma)} G(\rho(t),C)$$

and

$$(\forall \rho \in C(\sigma))[G(\rho(t),C) \cap G(\sigma(t),C) = \emptyset \vee \rho(t) = \sigma(t)].$$

Proof: By definition of the complement of a linear substitution. □

From Proposition 2 we get

$$G(t,C) = G(\sigma(t),C) \oplus \bigcup_{\rho \in C(\sigma) \& \rho(t) \neq \sigma(t)} G(\rho(t),C).$$

Usually $Dom(\sigma) \subset Var(t)$, but if $Dom(\sigma) = Var(t)$ then one gets a simpler equality

$$G(t,C) = G(\sigma(t),C) \oplus \bigcup_{\rho \in C(\sigma)} G(\rho(t),C).$$

Notice that the restriction is about the linearity of the substitution not about the linearity of t or $\sigma(t)$. For example, if $C = \{0, S\}$ and $\sigma = \{y \leftarrow S(x)\}$, then

$$\begin{aligned} G(f(y,y),C) \ominus G(\sigma(f(y,y)),C) &= G(f(y,y),C) \ominus G(f(S(x),S(x)),C) \\ &= G(f(0,0),C). \end{aligned}$$

This can be computed although $f(y,y)$ and $\sigma(f(y,y))$ are not linear. The following theorem gives a sufficient condition for testing covering.

Theorem 1 *Let M and N be two finite subsets of $\{\sigma(f(x_1,\ldots,x_n)) | \sigma \in \Sigma(C,X)\}$, M covers N if one of the following conditions are satisfied:*

(i) *N is empty.*

(ii) *There exists two terms $m \in M$ and $n \in N$ s.t. m and n are unified by a linear substitution σ and $M - \{m\} \cup \{\rho(m) | \rho \in C(\sigma) \& \rho(m) \neq \sigma(m)\}$ covers $N - \{n\} \cup \{\rho(n) | \rho \in C(\sigma) \& \rho(n) \neq \sigma(n)\}$.*

Proof: Let $M' = M - \{m\} \cup \{\rho(m) | \rho \in C(\sigma) \& \rho(m) \neq \sigma(m)\}$ and $N' = N - \{n\} \cup \{\rho(n) | \rho \in C(\sigma) \& \rho(n) \neq \sigma(n)\}$. By the previous proposition:

$$\begin{aligned} G(M,C) &= G(M - \{m\},C) \cup G(m,C) \\ &= G(M - \{m\},C) \cup (G(\sigma(m),C) \oplus \textstyle\bigcup_{\rho \in C(\sigma) \& \rho(m) \neq \sigma(m)} G(\rho(m),C)) \\ &= G(M',C) \cup G(\sigma(m),C) \end{aligned}$$

similarly,

$$\begin{aligned} G(N,C) &= G(N - \{n\},C) \cup G(n,C) \\ &= G(N - \{n\},C) \cup (G(\sigma(n),C) \oplus \textstyle\bigcup_{\rho \in C(\sigma) \& \rho(n) \neq \sigma(n)} G(\rho(n),C)) \\ &= G(N',C) \cup G(\sigma(n),C) \end{aligned}$$

Thus, since $\sigma(m) = \sigma(n)$, $G(M,C)$ contains $G(N,C)$ if $G(M',C)$ contains $G(N',C)$. □

Now we may state the theorem for testing the convertibility of an operator to a set of constructors [38]. One starts with $M_0 = \{L(f, R_D)\}$ and N_0 which is such that $G(N_0, C) = G(f(x_1, \ldots, x_n), C)$, and repeat the operations described in part (ii) of the theorem until N is empty or M is empty or no $m \in M$ and no $n \in N$ can be unified by a linear substitution. This will eventually happen since all the terms are of the form $f(t_1, \ldots, t_n)$. They are produced by unification and computation of complements and therefore are always different and are never deeper than the terms in M_0 and N_0. Let us call M_{last} and N_{last} the final results. The third statement of the following theorem assumes that N_0 has no superposition.

Theorem 2 (Complement Algorithm) *If the algorithm described above starts with an M_0 which is the set of left-hand sides of the definition of f in $A_{D,f}$ and an N_0 which is such that f is convertible to C by N_0, then*

- *if M_{last} and N_{last} are empty, f is convertible to C without ambiguity.*
- *if M_{last} is not empty and N_{last} is empty, f is convertible to C but all terms in M_{last} are defined more than once.*
- *if N_{last} is not empty and N_0 without superposition, f is not defined on the patterns that are not inductively (C, R_C)-reducible. Especially, f is convertible to C if and only if all the terms in N_{last} are inductively (C, R_C)-reducible.*

Proof: The first two statements are easy. Let us proof only the third one and assume that N_{last} is not empty and N_0 has no superposition. Therefore M_{last} is empty. Since, when computing complements, one creates terms without superposition, each N_i has no superposition for $0 \leq i \leq last$. If a term t is an instance of a term $n \in N_{last}$, it cannot be an instance of any $\sigma_i(n_i)$, where σ_i is the linear substitution and n_i the chosen term in N_i computed at the i^{th} step, with $0 \leq i \leq last$. Therefore, t cannot be an instance of an $m \in M_j$ for $0 \leq j \leq last$. Thus $t = \sigma(n)$ for $n \in N_{last}$ and $\sigma \in \Sigma(C)$ if and only if, for no $\tau \in \Sigma(C)$ and no $m \in M_0$, $t = \tau(m)$. Therefore f is C-convertible if and only if all the instances by a ground substitution into $\Sigma(C)$ of terms in N_{last} are (C, R_C)-reducible. $\square$

The content of N_{last} is interesting, since it gives the patterns where the function has to be defined. This feature makes our algorithm really handy in a environment for functional programming or abstract data types, since these patterns are obtained by unification they are in some sense the most general ones and operationally they seem to be better than methods based on unfolding a tree as proposed by E. Kounalis [22], D. Plaisted [33] and D. Kapur, P. Narendran and H. Zhang [20]. The concept of covering appeared first in [38], but was presented independently and more recently by Comon [2] and Kucherov [24] in the context of sufficient completeness, by Lassez and Marriott in the context of programming logic and automated learning [26] and by Laville in pattern matching algorithms [27].
Remark: Note that if M contains only linear terms, this procedure is a decision procedure for convertibility [31], but it is worthwhile to emphasize that this method can handle nonlinear term-rewriting systems. In general, one takes $N_0 = \{f(x_1, \ldots, x_n)\}$, but the algorithm may fail because all the substitutions that unify the terms are not linear, usually another complete and non ambiguous set N_0 could lead to a success. This is the case in Example 12 due to E. Kounalis [22].

Example 9 *Consider the axiomatization of Example 5. Here*

$$M_0 = \{0 + x, succ(x) + y, opp(succ(x)) + y\}$$

and let us take $N_0 = \{z + w\}$*. If one takes the terms in* M_0 *in their order, one gets*

$$\begin{aligned}
M_1 &= \{succ(x) + y, opp(succ(x) + y)\}\\
N_1 &= \{succ(z) + w, opp(z) + w\}\\
M_2 &= \{opp(succ(x) + y)\}\\
N_2 &= \{opp(z) + w\}\\
M_3 &= \emptyset\\
N_3 &= \{opp(0) + w, opp(opp(z)) + w\}.
\end{aligned}$$

In the first two steps, the unifier is trivially the identity and in the third one, the substitution is $\{z \leftarrow succ(x)\}$*.* N_3 *contains only* C*-reducible terms then the definition of* $+$ *is relatively complete.*

Example 10 *Consider the example* $flatten$ *proposed in the introduction and suppose that the rules*

$$\begin{aligned}
flatten(b) &\to b\\
flatten(a@x) &\to a@flatten(x)\\
flatten(b@x) &\to b@flatten(x)
\end{aligned}$$

are not given. Let us take

$$M_0 = \{flatten[\ ], flatten([x]), flatten(a), flatten([x]@y)\}$$

and $N_0 = \{flatten(z)\}$*. One gets*

$$\begin{aligned}
M_1 &= \{flatten([x]), flatten(a), flatten([x]@y)\}\\
N_1 &= \{flatten([x]), flatten(a), flatten(b), flatten(x@y)\}\\
M_2 &= \{flatten(a), flatten([x]@y)\}\\
N_2 &= \{flatten(a), flatten(b), flatten(x@y)\}\\
M_3 &= \{flatten([x]@y)\}\\
N_3 &= \{flatten(b), flatten(x@y)\}\\
M_4 &= \emptyset\\
N_4 &= \{flatten(b), flatten([\ ]@y), flatten(a@y), flatten(b@y), flatten((x@y)@z)\}
\end{aligned}$$

Since $flatten([\ ]@y)$ *and* $flatten((x@y)@z)$ *are* C*-reducible, the algorithm says that* $flatten$ *is not defined on* $flatten(b)$*,* $flatten(a@y)$ *and* $flatten(b@x)$*, therefore the definition, given in the introduction, that contains these lacking patterns is relatively complete.*

Example 11 *If the left-hand sides of the definition of eq are*

$$\{eq(x, x), eq(0, s(x)), eq(s(x), 0), eq(s(x), s(y))\}$$

the procedure can prove it is convertible to $\{0, s\}$ *although* $eq(x, x)$ *is not linear.*

Example 12 *This function computes the digit to carry in a binary adder. It is also the function majority in a fault tolerant system, see also [39]. Let* $C = \{1, 0\}$, $D = \{f\}$ *and the set of axioms:*

$$\begin{array}{rcl} f(x,x,y) & \rightarrow & x \\ f(x,y,x) & \rightarrow & x \\ f(y,x,x) & \rightarrow & x \end{array}$$

To test the completeness of this specification we take

$$M_0 = \{f(x,x,y), f(x,y,x), f(y,x,x)\}$$

and we can take

$$N_0 = \{f(z,w,1), f(z,w,0)\}$$

which is without superposition and covers $\{f(x_1,x_2,x_3)\}$*. This can be shown by running the algorithm on* $N_0 = \{f(x_1,x_2,x_3)\}$*. Let us prove that* M_0 *covers* N_0*. If one starts by unifying* $f(x,x,y)$ *and* $f(z,w,1)$ *one finds the most general unifier* $\sigma = \{z \leftarrow x, w \leftarrow x, y \leftarrow 1\}$ *which is not linear. However* $f(x,y,x)$ *and* $f(z,w,1)$ *are unified by* $\theta = \{x \leftarrow 1, y \leftarrow w, z \leftarrow 1\}$ *which is linear and*

$$C(\theta) = \{\{x \leftarrow 0, y \leftarrow w, z \leftarrow 1\}, \{x \leftarrow 1, y \leftarrow w, z \leftarrow 0\}, \{x \leftarrow 0, y \leftarrow w, z \leftarrow 0\}\}$$

In M_0, $f(x,y,x)$ *is replaced by* $f(0,w,0)$ *and in* N_0, $f(z,w,1)$ *is replaced by* $f(0,w,1)$.

$$M_1 = \{f(x,x,y), f(0,w,0), f(y,x,x)\}$$

has to cover

$$N_1 = \{f(0,w,1), f(z,w,0)\}$$

$f(y,x,x)$ *and* $f(z,w,0)$ *are unified by* $\zeta = \{y \leftarrow z, x \leftarrow 0, w \leftarrow 0\}$

$$C(\zeta) = \{\{y \leftarrow z, x \leftarrow 0, w \leftarrow 1\}, \{y \leftarrow z, x \leftarrow 1, w \leftarrow 0\}, \{y \leftarrow z, x \leftarrow 1, w \leftarrow 1\}\}.$$

Then in M_1, $f(y,x,x)$ *is replaced by* $f(z,1,1)$, *in* N_1, $f(z,w,0)$ *is replaced by* $f(z,1,0)$ *and*

$$M_2 = \{f(x,x,y), f(0,w,0), f(z,1,1)\}$$

has to cover

$$N_2 = \{f(0,w,1), f(z,1,0)\}$$

$f(x,x,y)$ *and* $f(0,w,1)$ *are unified by* $\eta = \{x \leftarrow 0, w \leftarrow 0, y \leftarrow 1\}$.

$$\begin{array}{rc} C(\eta) = & \{\{x \leftarrow 1, w \leftarrow 0, y \leftarrow 1\}, \{x \leftarrow 0, w \leftarrow 1, y \leftarrow 0\}, \\ & \{x \leftarrow 1, w \leftarrow 1, y \leftarrow 1\}, \{x \leftarrow 1, w \leftarrow 0, y \leftarrow 0\}, \\ & \{x \leftarrow 1, w \leftarrow 1, y \leftarrow 0\}, \{x \leftarrow 0, w \leftarrow 1, y \leftarrow 1\}, \{x \leftarrow 0, w \leftarrow 0, y \leftarrow 0\}\}. \end{array}$$

In M_2, $f(x,x,y)$ *is replaced by* $f(1,1,1)$, $f(1,1,0)$, $f(0,0,0)$, *in* N_2, $f(0,w,1)$ *is replaced by* $f(0,1,1)$ *and*

$$M_3 = \{f(1,1,1), f(1,1,0), f(0,0,0), f(0,w,0), f(z,1,1)\}$$

has to cover

$$N_3 = \{f(0,1,1), f(z,1,0)\}$$

$f(0,w,0)$ and $f(z,1,0)$ are unified by $\rho = \{w \leftarrow 1, z \leftarrow 0\}$.

$$C(\rho) = \{\{w \leftarrow 0, z \leftarrow 0\}, \{w \leftarrow 0, z \leftarrow 1\}, \{w \leftarrow 1, z \leftarrow 1\}\}.$$

In M_3, $f(0,w,0)$ is replaced $f(0,0,0)$, in N_3, $f(z,1,0)$ is replaced by $f(1,1,0)$ and

$$M_4 = \{f(1,1,1), f(1,1,0), f(0,0,0), f(0,0,0), f(z,1,1)\}$$

has to cover

$$N_4 = \{f(0,1,1), f(1,1,0)\}$$

$f(z,1,1)$ and $f(0,1,1)$ are unified by $\tau = \{z \leftarrow 0\}$ and

$$C(\tau) = \{z \leftarrow 1\}.$$

In M_4, $f(z,1,1)$ is replaced by $f(1,1,1)$, in N_4, $f(0,1,1)$ disappears and

$$M_5 = \{f(1,1,1), f(1,1,0), f(0,0,0), f(0,0,0), f(1,1,1)\}$$

has to cover

$$N_5 = \{f(1,1,0)\}$$

$f(1,1,0)$ and $f(1,1,0)$ are unified by the identity substitution and both disappear from M_5 and N_5.

$$M_6 = \{f(1,1,1), f(0,0,0), f(0,0,0), f(1,1,1)\}$$

covers trivially

$$N_6 = \emptyset$$

hence f is convertible to C and the two terms $f(1,1,1)$ and $f(0,0,0)$ are defined more than once, exactly three times because of the two occurrences in M_6.

The previous example suggests that the method does not fail often. Actually Kounalis has shown that if the set of relations among constructors is empty then the method is complete. Actually the failure of Example 12 is due to the finiteness of $T(C)$. In the case of an infinite $T(C)$ a non linear term always exists.

Theorem 3 (Kounalis [23]) *If C is empty, the complement algorithm is complete, which means that it does not fail.*

Sketch of the proof: Two cases have to be considered. If $T(C)$ is finite one takes $N_0 = \{f(t_1, ..., t_n) | t_i \in T(C)\}$ and the method cannot fail.

If $T(C)$ is infinite, the completeness relies on a result due to Lassez and Marriott [26] that says that for a linear term t, $G(t, C)$ cannot be covered by disjoint sets of instances of non linear terms. Thus at each step, one can always choose a non linear term in N_i.

Relative completeness and equational rewriting

The algorithm we have presented does not require use of the most general unifier when computing the complements. Any unifier may work, provided it is linear, but with non most general unifiers the termination is not guaranteed. For instance, in Example 11, if in the definition of eq we unify, $eq(x,x)$ and $eq(y,z)$ with $\{x \leftarrow 0, y \leftarrow 0, z \leftarrow 0\}$, the complement of $eq(0,0)$ in $eq(y,z)$ is $\{eq(s(y),z), eq(0,s(z))\}$. Then $eq(s(x),s(x))$ and $eq(s(y),z)$ can be unified by $\{x \leftarrow 0, y \leftarrow 0, z \leftarrow s(0)\}$, this may continue forever with $\{x \leftarrow 0, y \leftarrow 0, z \leftarrow s^2(0)\}$, $\{x \leftarrow 0, y \leftarrow 0, z \leftarrow s^3(0)\}$, ... $\{x \leftarrow 0, y \leftarrow 0, z \leftarrow s^n(0)\}$, ...

In many specific situations, one works modulo a set of equations, for instance modulo commutativity or modulo commutativity and associativity that are equalities that cannot be oriented into rewrite rules. Examples of this kind are given in the next section. In this case, the system returns a complete set of unifiers which may contains more than one substitution and usually this set is not minimal. The complement algorithm still works and uses all the substitutions yielded by the equational unification algorithm. We have to give up on results on the emptyness of N_{last}, that are based on the minimality (or the generality) of the unifiers. Moreover the argument for the termination of the algorithm cannot be used because most of time the unifier increases the depth of the term, especially by adding variables. Therefore a specific method has to be adapted to each specific equational theory.

4 ω-COMPLETENESS

In this section, we give a general method for proving ω-completeness in the case of specifications described by term rewriting systems and we illustrate its application on examples. This method is based on discrimination by ground terms of pairs of non equal non ground terms.

By definition, a specification (F,A) is ω-complete iff for every terms u and v in $T(F,X)$:

$$u =_{ind(A)} v \;\Rightarrow\; u =_A v. \tag{1}$$

Let $u\!\downarrow$ denote the R-normal form of u, where R is a convergent rewriting system associated with A. Obviously,

$$u =_{ind(A)} v \;\rightarrow\; u\!\downarrow =_{ind(A)} v\!\downarrow,$$

and

$$u =_A v \rightarrow u\!\downarrow = v\!\downarrow$$

i.e., $u\!\downarrow$ is syntactically the same term as $v\!\downarrow$. Thus (1) is equivalent to

$$u\!\downarrow =_{ind(A)} v\!\downarrow \;\Rightarrow\; u\!\downarrow = v\!\downarrow\,. \tag{2}$$

In other words,

$$(\forall \sigma \in \Sigma(F))\; \sigma(u\!\downarrow)\!\downarrow = \sigma(v\!\downarrow)\!\downarrow \Rightarrow u\!\downarrow = v\!\downarrow\,.$$

By contraposition, the inductive completeness can be expressed as

$$u\!\downarrow \neq v\!\downarrow \Rightarrow (\exists \sigma \in \Sigma(F))\; \sigma(u\!\downarrow)\!\downarrow \neq \sigma(v\!\downarrow)\!\downarrow\,.$$

This means that, if $u\!\downarrow$ and $v\!\downarrow$ are two different normal forms, their exists a ground substitution σ that produces two different normal forms after instanciation and normalization. We say that σ discriminates u and v.

Example 13 *Let $F = \{0, opp, succ\}$ and suppose the set of axioms is:*

$$\begin{array}{rcl} opp(0) & \rightarrow & 0 \\ opp(opp(x)) & \rightarrow & x \\ succ(opp(succ(x))) & \rightarrow & opp(x) \end{array}$$

The normal forms of the ground terms are one of the following:

$$\begin{array}{rl} 0 & \\ succ^n(0) & \text{for } n > 0 \\ opp(succ^n(0)) & \text{for } n > 0 \end{array}$$

The normal forms of the terms of the free algebra $T(F, \{x\}, A)$ are one of the following:

$$\begin{array}{l} succ^n(x) \text{ for } n \geq 0 \\ succ^n(opp(x)) \text{ for } n \geq 0 \\ opp(succ^n(x)) \text{ for } n > 0 \\ opp(succ^n(opp(x))) \text{ for } n > 0 \end{array}$$

where $succ^0(x)$ is x and $succ^n(x) = suc(\ldots(succ(x))\ldots)$, n times.

If two normal forms have two different variables they can be discriminated. Let $u\downarrow$ and $v\downarrow$ be two terms of $T(F, \{x\}, A)$ s.t. $u\downarrow \neq v\downarrow$. In what follows, some cases are skipped by symmetry.

- *if $u\downarrow = succ^n(x)$ then*
  - *if $v\downarrow = succ^m(opp(x))$ then*
    - *if $n = m$ then let $\sigma = \{x \leftarrow succ(0)\}, \sigma(u\downarrow)\downarrow = succ^{n+1}(0)$ and $\sigma(v\downarrow)\downarrow = succ^{n-1}(0)$ thus $\sigma(u\downarrow)\downarrow \neq \sigma(v\downarrow)\downarrow$*
    - *if $n \neq m$ then let $\sigma = \{x \leftarrow 0\}, \sigma(u\downarrow)\downarrow = succ^n(0)$ and $\sigma(v\downarrow)\downarrow = succ^m(0)$ thus $\sigma(u\downarrow)\downarrow \neq \sigma(v\downarrow)\downarrow$*
  - *if $v\downarrow = opp(succ^m(x))$, let $\sigma = \{x \leftarrow 0\}$ then $\sigma(u\downarrow)\downarrow = succ^n(0)$ and $\sigma(v\downarrow)\downarrow = opp(succ^m(0))$ thus $\sigma(u\downarrow)\downarrow \neq \sigma(v\downarrow)\downarrow$*
  - *if $v\downarrow = opp(succ^m(opp(x)))$, let $\sigma = \{x \leftarrow 0\}$ then $\sigma(u\downarrow)\downarrow = succ^n(0)$ and $\sigma(v\downarrow)\downarrow = opp(succ^m(0))$ thus $\sigma(u\downarrow)\downarrow \neq \sigma(v\downarrow)\downarrow$*
- *$u\downarrow = succ^n(opp(x))$ then*
  - *if $v\downarrow = opp(succ^m(x))$ let $\sigma = \{x \leftarrow 0\}$ then $\sigma(u\downarrow)\downarrow = succ^n(0)$ and $\sigma(v\downarrow)\downarrow = opp(succ^m(0))$ thus $\sigma(u\downarrow)\downarrow \neq \sigma(v\downarrow)\downarrow$*
  - *if $v\downarrow = opp(succ^m(opp(x)))$ then $\sigma = \{x \leftarrow 0\}$, $\sigma(u\downarrow)\downarrow = succ^n(0)$ and $\sigma(v\downarrow)\downarrow = opp(succ^m(0))$ thus $\sigma(u\downarrow)\downarrow \neq \sigma(v\downarrow)\downarrow$*
- *$u\downarrow = opp(succ^n(x))$ then $v\downarrow = opp(succ^m(opp(x)))$*
  - *if $n = m$ let $\sigma = \{x \leftarrow succ(0)\}$ then $\sigma(u\downarrow)\downarrow = opp(succ^{n+1}(0))$ and $\sigma(v\downarrow)\downarrow = opp(succ^{n-1}(0))$ thus $\sigma(u\downarrow)\downarrow \neq \sigma(v\downarrow)\downarrow$*

– *if $n \neq m$ let $\sigma = \{x \leftarrow 0\}$,*
then $\sigma(u\downarrow)\downarrow = opp(succ^n(0))$ and $\sigma(v\downarrow)\downarrow = opp(succ^m(0))$ thus $\sigma(u\downarrow)\downarrow \neq \sigma(v\downarrow)\downarrow$

Example 14 *Let $F = \{0, a, b, .\}$ and suppose the set of axioms is:*

$$\begin{aligned} x.0 &\rightarrow x \\ 0.x &\rightarrow x \\ (x.y).z &\rightarrow x.(y.z) \end{aligned}$$

the initial algebra $T(F, A)$ is the free monoid $(\{a, b\})^*$ and the free algebra $T(F, X, A)$ is equivalent to $(\{a, b\} \cup X)^*$. Let $u\downarrow$ and $v\downarrow$ two terms of $T(F, X, A)$ s.t. $u\downarrow \neq v\downarrow$ therefore there exist $c, d \in \{a, b\} \cup X$ s.t. $c \neq d$, $u\downarrow = t.c.t'$ and $v\downarrow = t.d.t''$.

- *If $c \in \{a, b\}$ and $d \in \{a, b\}$ then for all σ, $\sigma(u\downarrow)\downarrow \neq \sigma(v\downarrow)\downarrow$*
- *if $c \in \{a, b\}$ and $d = x$ then if $c = a$ (resp. $c = b$) then let $\sigma = \{x \leftarrow b\}$(resp.$\{x \leftarrow a\}$) thus $\sigma(u\downarrow)\downarrow \neq \sigma(v\downarrow)\downarrow$*
- *if $c = x$ and $d = y$ then let $\sigma = \{x \leftarrow a, y \leftarrow b\}$ thus $\sigma(u\downarrow)\downarrow \neq \sigma(v\downarrow)\downarrow$*

Example 15 *Let $F = \{0, 1, opp, +\}$ and suppose the set of axioms is*

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

The normal forms are $u = p_1.x_1 + \ldots + p_m.x_m + q_1.opp(y_1) + \ldots + q_n.opp(y_n)$. Note that $x_i \neq y_j$ for all i and j. Let $u \neq u'$, without loss of generality, one may suppose that a variable z occurs in the first part of u with coefficient p and either in the first part of u' with coefficient p' and $p' \neq p$ or in the second part of u' with coefficient q'. In both case, define the substitution σ, s.t. $\sigma(z) = 1$ and $\sigma(v) = 0$ for $v \neq z$. In the first case, we get $\sigma(u) = p.1$ and $\sigma(u') = p'.1$ and, in the second case, we get $\sigma(u) = p.1$ and $\sigma(u') = q'.opp(1)$.

Example 16 *Let $F = \{0, s, eq\}$ and consider the following set of axioms:*

$$\begin{aligned} eq(0, s(x)) &= 0 \\ eq(x, x) &= s(0) \\ eq(s(x), s(y)) &= eq(x, y) \\ eq(x, y) &= eq(y, x) \end{aligned}$$

The normal forms are equivalence classes of terms modulo the commutativity of eq, namely the class that contains just 0, the classes that contain just $s^n(0)$ for each n, the classes $\{eq(s^n(x), y), eq(y, s^n(x))\}$ for each n, the classes $\{eq(s^n(0), y), eq(y, s^n(0))\}$ for each n and the classes $\{eq(s^n(x), 0), eq(0, s^n(x))\}$ for each n. Theses classes are easily discriminated.

5 Conclusion

This method for checking relative completeness based on calculus of complements is practical since it can handle most of the specification we know and return useful information on where the functions have to be defined. It was implemented in the rewrite rule laboratory REVE. Namely this lead to study a new kind of equational problems called *disunification* [3]. The problem of the ω-completeness is harder and one may expect to find methods by generalizing the one which is given in Section 4.

The authors had many discussions with many people on these issues and among them they remember the fruitful ones they had with Irina Bercovici and Alain Laville. They would like to thank everyone.

References

[1] R.M. Burstall, D.B. MacQueen, and D.T. Sannella. Hope: an experimental applicative language. In *Conference Record of the 1980 LISP Conference*, pages 136–143, 1980.

[2] H. Comon. Sufficient completeness, term rewriting system and anti-unification. In J. Siekmann, editor, *Proceedings 8th Conference on Automated Deduction*, pages 128–140, Springer Verlag, Lecture Notes in Computer Science 230, 1986.

[3] H. Comon and P. Lescanne. *Equational problems and disunification.* Technical Report 88-R-026, Centre de Recherche en Informatique de Nancy, 1988. To appear in Journal of Symbolic Computation.

[4] G. Cousineau, L. Paulson, G. Huet, R. Milner, M. Gordon, and C. Wadsworth. *The ML Handbook.* INRIA, Rocquencourt, May 1985.

[5] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1. Equations and initial semantics.* Volume 6 of *EATCS Monographs on Theorical Computer Science*, Springer-Verlag, 1985.

[6] Projet FORMEL. *CAML: the reference Manuel.* Technical Report, INRIA-ENS, March 1987.

[7] K. Futatsugi, J. Goguen, J-P. Jouannaud, and J. Meseguer. Principles of OBJ-2. In B. Reid, editor, *Proceedings 12th ACM Symp. on Principles of Programming Languages*, pages 52–66, Association for Computing Machinery, 1985.

[8] J. H. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving.* Volume 5 of *Computer Science and Technology Series*, Harper & Row, New-York, 1986.

[9] J. Goguen, C. Kirchner, H. Kirchner, A. Megrelis, J. Meseguer, and T. Winkler. An introduction to OBJ-3. In J-P. Jouannaud and S. Kaplan, editors, *Proceedings 1st International Workshop on Conditional Term Rewriting Systems*, Springer-Verlag, June 1988. Also as internal report CRIN: 88-R-001.

[10] J.A. Goguen. How to prove algebraic inductive hypotheses without induction, with applications to the correctness of data type implementation. In W. Bibel and Kowalski, editors, *Proceedings 5th Conference on Automated Deduction*, pages 356–373, Springer-Verlag, Les Arcs (France), 1980.

[11] G. Grätzer. *Universal Algebra.* Springer-Verlag, 1979. Second Edition.

[12] J. V. Guttag and J. J. Horning. The algebraic specification of abstract data types. *Acta Informatica*, 10:27–52, 1978.

[13] J.V. Guttag. Abstract data types and software validation. *Communications of the Association for Computing Machinery*, 21:1048–1064, 1978.

[14] J. Heering. Partial evaluation and ω-completeness of algebraic specifications. *Theoretical Computer Science*, 43:149–167, 1986.

[15] L. Henkin. The logic of equality. *The American Mathemtical Monthly*, 84:597–612, 1977.

[16] G. Huet. Confluent reductions: abstract properties and applications to term rewriting systems. *Journal of the Association for Computing Machinery*, 27(4):797–821, 1980. Preliminary version in 18th Symposium on Foundations Of Computer Science, IEEE, 1977.

[17] G. Huet and J-M. Hullot. Proofs by induction in equational theories with constructors. *Journal of Computer and System Sciences*, 25(2):239–266, October 1982. Preliminary version in Proceedings 21st Symposium on Foundations of Computer Science, IEEE, 1980.

[18] J.P. Jouannaud and E. Kounalis. Proof by induction in equational theories without constructors. In *Proceedings 1st Symp. on Logic In Computer Science*, pages 358–366, Boston (USA), 1986.

[19] D. Kapur and D. R. Musser. Proof by consistency. *Artificial Intelligence*, 13(2):125–157, 1987.

[20] D. Kapur, P. Narendran, and H. Zhang. On sufficient completeness and related properties of term rewriting systems. *Acta Informatica*, 24:395–415, 1987.

[21] H. Kirchner. A general inductive completion algorithm and application to abstract data types. In R. Shostak, editor, *Proceedings 7th international Conference on Automated Deduction*, pages 282–302, Springer-Verlag, Lecture Notes in Computer Science, 1984.

[22] E. Kounalis. Completeness in data type specifications. In B. Buchberger, editor, *Proceedings EUROCAL Conference*, Springer-Verlag, Linz (Austria), 1985.

[23] E. Kounalis. Validation de spécifications algébriques par complétion inductive. Thèse de Doctorat, Université de Nancy 1, CRIN, Nancy, 1985.

[24] G. A. Kucherov. A new quasi-reducibility testing algorithm and its application to proofs by induction. In *Proceedings workshop on Algebraic and Logic Programming*, pages 204–213, Akademie Verlag, 1988. To appear in Lecture Notes in Computer Science.

[25] D. S. Lankford. *A simple explanation of inductionless induction.* Technical Report, Lousiana Tech, Ruston (Louisiana), 1981.

[26] J-L. Lassez and K. Marriott. Explicit representation of terms defined by counter examples. *Journal of Automated Reasoning*, 3(3):301–318, 1986.

[27] A. Laville. Lazy pattern matching in the ML language. In *Proceedings 7th Conf. on Foundations of Software Technology and Theoretical Computer Science*, pages 400–419, Springer-Verlag, Lecture Notes in Computer Science, December 1987.

[28] J. Meseguer and J.A. Goguen. Initiality, induction and computability. In M. Nivat and J. Reynolds, editors, *Algebraic Methods in Semantics*, Cambridge University Press, 1985.

[29] R. Milner. A proposal for standard ML. In *Proceedings ACM Conference on LISP and Functional Programming*, 1984.

[30] D.L. Musser. On proving inductive properties of abstract data types. In *Proceedings 7th ACM Symp. on Principles of Programming Languages*, pages 154–162, Association for Computing Machinery, 1980.

[31] T. Nipkow and G. Weikum. A decidability result about sufficient completeness of axiomatically specified abstract data types. In *6th GI Conference*, pages 257–268, Springer-Verlag, Lecture Notes in Computer Science, 1983.

[32] E. Paul. Proof by induction in equational theories with relations between constructors. In B. Courcelle, editor, *Proceedings 9th Colloquium on Trees in Algebra and Programming*, pages 210–225, Cambridge University Press, 1984.

[33] D. Plaisted. Semantic confluence and completion method. *Information and Control*, 65:182–215, 1985.

[34] G. D. Plotkin. The λ-calculus is ω-incomplete. *Journal of Symbolic Logic*, 39:313–317, 1974.

[35] L. Puel. Proof in the final algebra. In B. Courcelle, editor, *Proceedings 9th Colloquium on Trees in Algebra and Programming*, pages 227–242, Cambridge University Press, 1984.

[36] J-L. Rémy. Etude des systèmes de réécriture conditionnels et applications aux types abstraits algébriques. Thèse d'Etat de l'Institut National Polytechnique de Lorraine, 1982.

[37] W. Taylor. Equational logic. *Houston Journal of Mathematics*, 1979. Appears also in [11], Appendix 4.

[38] J-J. Thiel. Stop loosing sleep over incomplete data type specifications. In *Proceeding 11th ACM Symp. on Principles of Programming Languages*, pages 76–82, Association for Computing Machinery, 1984.

[39] Y. Toyama. Counterexamples to terminating for the direct sum of term rewriting systems. *Information Processing Letters*, 25(3):141–143, May 1986.

[40] D. A. Turner. MIRANDA: a non-strict functional language with polymorphic types. In J-P. Jouannaud, editor, *Proceedings 2nd Conf. on Functional Programming Languages and Computer Architecture*, pages 1–16, Springer-Verlag, Lecture Notes in Computer Science, 1985.