Simultaneous Checking of Completeness and Ground Confluence for Algebraic Specifications

ADEL BOUHOULA
Higher School of Communication of Tunis (Sup'Com)
University of November 7th at Carthage - Tunisia

Algebraic specifications provide a powerful method for the specification of abstract data types in programming languages and software systems. Completeness and ground confluence are fundamental notions for building algebraic specifications in a correct and modular way. Related works for checking ground confluence are based on the completion techniques or on the test that all critical pairs between axioms are valid w.r.t. a sufficient criterion for ground confluence. It is generally accepted that such techniques may be very inefficient even for very small specifications. Indeed, the completion procedure often diverges and there often exist many critical pairs of the axioms. In this paper, we present a procedure for simultaneously checking completeness and ground confluence for specifications with free/non-free constructors and parameterized specifications. If the specification is not complete or not ground confluent, then our procedure will output the set of patterns on whose ground instances a function is not defined and it can easily identify the rules that break ground confluence. In contrast to previous work, our method does not rely on completion techniques and does not require the computation of critical pairs of the axioms. The method is entirely implemented and allowed us to prove the completeness and the ground confluence of many specifications in a completely automatic way where related techniques diverge or generate very complex proofs. Our system offers two main components: (i) a completeness and ground confluence analyser that compute pattern trees of defined functions and may generate some proof obligations; and (ii) a procedure to prove (joinable) inductive conjectures which is used to discharge those proof obligations.

Categories and Subject Descriptors: F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Mechanical theorem proving*; F.4.2 [**Mathematical Logic and Formal Languages**]: Grammars and Other Rewriting Systems; F.4.3 [**Mathematical Logic and Formal Languages**]: Formal languages—*Algebraic language theory.*

General Terms: Theory, verification, languages.

Additional Key Words and Phrases: Automated deduction, term rewriting systems, algebraic apecifications, parameterization, completeness, ground confluence.

Author's address: Adel Bouhoula. Sup'Com, City of Communication Technologies, 2083 Ariana, Tunisia. Fax : +216 71 856 829.
E-mail : adel.bouhoula@supcom.rnu.tn

A Preliminary version of the results of this paper was presented at *the Fifteenth IEEE International Conference on Automated Software Engineering*. IEEE Computer Society Press, Grenoble, France, 143–151 [Bouhoula 2000].

1. INTRODUCTION

One of the important aspects in the process of program construction is the choice and the treatment of the basic data structures. Algebraic specification methods provide techniques for data abstraction and the validation and analysis of data structures. The basic idea of the algebraic approach [Wirsing 1990] is to describe data structures by just giving the names of the different sets of data and the names of the basic functions and their characteristic properties. Therefore, algebraic specifications provide a powerful method for the specification of software systems in an abstract way and independently of its effective implementation. For the description of large data structures and complex systems, we can use parameterized specifications [Padawitz 1988] which allow one to compose specifications in a modular way and to build larger specifications from smaller ones. Often algebraic specifications are built with conditional equations. Semantically, the motivation for this is the existence of initial models; operationally, the motivation is the ability to use term rewriting techniques for computing and automatic prototyping.

The completeness and ground confluence properties are very important for building algebraic specifications in a correct and modular way. Ground confluence is particularly useful to guarantee the refutational completeness of inductive theorem proving, which implies that every conjecture that is not valid in the initial model will be detected in finite time. This property is very important since practice shows that code is usually buggy, and therefore many properties which are expected to hold, in fact do not.

Completeness means that any ground (i.e., variable-free) term should return a result built upon constructor symbols. Many techniques have been developed for checking this property for non-conditional specifications [Guttag and Horning 1978; Huet and Hullot 1982; Dershowitz 1983; Thiel 1984; Kounalis 1985; Comon 1986; Jouannaud and Kounalis 1989; Lazrek et al. 1990; Kapur et al. 1991; Kapur 1994] and conditional specifications [Bouhoula 1996; Bouhoula and Jouannaud 2001]. Ground confluence guarantees the property of uniqueness in computation with ground terms. Several works have proposed sufficiency criteria for confluence of conditional systems [Küchlin 1985; Dershowitz et al. 1987; Kaplan 1987; Gramlich and Wirth 1996]. However, little work has been carried out on checking ground confluence. This is mostly due to the fact that the problem is much harder. Indeed, ground confluence is undecidable [Kapur et al. 1990] even for equational theories with only unary function symbols. Plaisted has proposed a semantic confluence test [Plaisted 1985], but he has not shown how his test can be automatized. Completion techniques are used in [Ganzinger 1987; Fribourg 1989]. It is generally accepted that such techniques may be very inefficient since the completion procedure often diverges even for very small specifications. Other methods were developed in [Göbel 1987; Kounalis and Rusinowitch 1991; Becker 1993; 1996] which do not rely on the completion framework. The key idea of these methods is to compute all critical pairs between axioms, and then to check each critical pair w.r.t. a sufficient criterion for ground confluence. The main drawback of these methods is that they generate a lot of critical pairs and that the ground confluence criteria are very hard to automate.

The key idea of our method comes from the observation that completeness and

ground confluence are interdependent. Indeed, to prove completeness for a conditional specification $\mathcal{SP}$, we need $\mathcal{SP}$ to be ground confluent [Bouhoula 1996], and to prove that $\mathcal{SP}$ is ground confluent using semantic techniques as in [Plaisted 1985], we need $\mathcal{SP}$ to be complete. This observation motivates us to develop a new procedure for simultaneously checking completeness and ground confluence for specifications with free/non-free constructors and parameterized specifications. Our procedure computes a pattern tree for every defined symbol, and identifies a set of rules whose inductive validity has to be checked. The leaves of the tree give a partition of the possible arguments for defined functions. If all the leaves are ground reducible and non-ambiguous, and if the identified rules are inductively valid, then we conclude that the given specification is complete and ground confluent. If the specification is not complete neither ground confluent, then our procedure will output the set of patterns on whose ground instances a function is not defined and it can easily detect the rules that break ground confluence.

As opposed to previous works, our procedure does not rely on completion techniques, and does not require the computation of critical pairs of the axioms. The method has been implemented in the prover SPIKE [Bouhoula et al. 1995; Bouhoula and Rusinowitch 1995; Bouhoula 1997]. We have tested this system on several examples which have highlighted the simplicity and the efficiency of our approach compared to related techniques.

The organization of this paper is as follows: In Section 2, we briefly introduce basic concepts about term rewriting and order-sorted algebras. In Section 3, we give the formal definitions of completeness and ground confluence. We present in Section 4 the ingredients needed to compute an induction schema for a given specification which are necessary for the computation of pattern trees. In Sections 5 and 6, we describe our inference system for simultaneously checking completeness and ground confluence for specifications with free constructors. Then, we show the soundness and the completeness of our procedure. In Section 7 and Section 8, we extend our inference system in order to check the completeness and ground confluence of specifications with non-free constructors and parameterized specifications, respectively. We present in Section 9 some computer experiments.

2. BASIC CONCEPTS

We assume that the reader is familiar with the basic concepts of term rewriting [Dershowitz and Jouannaud 1990], order-sorted algebras [Goguen and Meseguer 1988; Smolka et al. 1987] and mathematical logic. Notions and notations not defined here are standard.

2.1 Order-Sorted Signature

An *order-sorted* signature Σ consists of a set $\mathcal{S}$ of sort symbols, a partial ordering $\leq$ on $\mathcal{S}$, a countable set $\mathcal{X}_s$ of variables for each sort symbol $s \in \mathcal{S}$, a set $\mathcal{F}$ of function symbols disjoint from $\mathcal{X} = \uplus_{s\in\mathcal{S}}\mathcal{X}_s$, such that each function symbol is equipped with an arity $n \in \mathbb{N}$, n input sorts $s_1, \ldots, s_n$, and an output sort s. Let $\mathcal{F}_{s_1\times\ldots\times s_n\to s}$ denote the set of function symbols of output sort s, whose n successive inputs belong to the sorts $s_1, \ldots, s_n$ (a function symbol can have more than one

function declaration):

$$\mathcal{F} = \bigcup_{n} \bigcup_{s_1,\ldots,s_n,s\in\mathcal{S}} \mathcal{F}_{s_1\times\ldots\times s_n\to s}$$

An *order-sorted* term T of sort $s \in \mathcal{S}$ is either a variable of sort t where $t \leq s$ or $f(U_1,\ldots,U_n)$ where $f \in \mathcal{F}_{t_1\times\ldots\times t_n\to t}$, $t \leq s$ and $\forall i \in [1..n]$, U_i is a term of sort t_i. We use small letters s, t for sorts, and capitals L, R, S, T, U, V, W for terms. The set of *order-sorted terms* will be denoted by $\mathcal{T}(\mathcal{F},\mathcal{X})$.

Terms are identified with finite labelled trees as usual. *Positions* are strings of positive integers. Λ is the empty string (root position) and $|p|$ stands for the length of the string p. We use $\mathcal{P}os(U)$ for the set of positions in U, $\mathcal{FP}os(U)$ for its set of non-variable positions, and $\mathcal{VP}os(U)$ for its set of variable positions. The *depth* (resp. *non-variable depth*) of a term T is the maximum length of a position $p \in \mathcal{P}os(T)$ (resp. $p \in \mathcal{FP}os(T)$).

The *subterm* of M at position p is denoted by $M|_p$, and we write $M \rhd M|_p$ if $p \neq \Lambda$. The result of replacing $M|_p$ with N at position p in M is denoted by $M[N]_p$. This is also used to indicate that N is a subterm of M, in which case p may be omitted. We use $\mathcal{V}ar(M)$ for the set of variables of M. Variable-free terms are called *ground*. By $\mathcal{T}(\mathcal{F})$ we denote the set of ground terms. A term M is *linear* if every variable in $\mathcal{V}ar(M)$ occurs exactly once in M. We assume that each sort contains a ground term. Given an order-sorted term T such that $\mathcal{V}ar(T) \subseteq \overline{x}$, an *environment* $\Gamma = \{\overline{x} : \overline{s}\}$ assigns a unique sort to every variable of $\overline{x}$.

Order-sorted substitutions are written as $\{x_1 \mapsto M_1,\ldots,x_n \mapsto M_n\}$ where M_i and x_i are different terms of the same sort. If $\forall i \in [1..n]$ M_i is a variable and $\forall i,j \in [1..n]$ $M_i \neq M_j$, the substitution is a *renaming*. We use small Greek letters for substitutions and postfix notation for their application.

A term T is *subsumed* by a term S if $T = S\sigma$ for some substitution σ. We also say that S is *more general* than T. Subsumption is a quasi-ordering on terms denoted by $\trianglerighteq$, whose strict part is well-founded, and whose equivalence, $\stackrel{\bullet}{=}$, called *conversion*, is given by the renaming of variables. This ordering is extended to substitutions by letting $\sigma \trianglerighteq \tau$ if $\sigma = \tau\theta$ for some θ. We say that two terms S and T *unify* if there exists a substitution σ such that $S\sigma = T\sigma$. The set of unifiers of two given terms S, T possesses a unique (up to conversion) minimal unifier with respect to subsumption, called the *most general unifier* of S and T, and denoted by $mgu(S,T)$. We say that a term S matches a term T with a substitution σ if $S\sigma = T$.

A sort s is a *least sort*, if $s \in \mathcal{S}$ and all sorts $t \in \mathcal{S}$, $t \not< s$. We say that a signature Σ is *regular* if each term has a unique least sort. Regularity guarantees that syntactic unification does not yield infinitely many most general unifiers [Schmidt-Schauß 1989]. Throughout this paper, we assume that Σ is regular.

We denote by $\overline{U}$ the list (or vector) $(U_1,\ldots,U_n)$. Given two vectors $\overline{U}$ and $\overline{U'}$ of equal length over the respective sets E and E', and a binary relation $*$ over $E \times E'$, we use the notation $\overline{U} * \overline{U'}$ as an abbreviation for the vector $(U_1 * U'_1,\ldots,U_n * U'_n)$. When E and E' are sets of formulae, $\overline{U} * \overline{U'}$ will instead denote the formula $(U_1 * U'_1 \wedge \ldots \wedge U_n * U'_n)$.

2.2 Axioms

We assume that the signature comes in two parts, a set of constructors $\mathcal{C}$, and a set of defined symbols $\mathcal{D}$ along with a rewrite ordering $\succ$, that is, an ordering on terms which is well-founded, and monotonic with respect to contexts and substitutions [Dershowitz 1987]. We use $\mathcal{T}(\mathcal{C},\mathcal{X})$ and $\mathcal{T}(\mathcal{D},\mathcal{X})$ for the respective sets of terms. $\overline{s}$, $\overline{U}$ will denote, respectively, lists or products of sorts and terms.

2.2.1 *Constructors.* The axioms for constructors are *rewrite rules* of the form:

$$\overline{x} : \overline{s} \Rightarrow c(\overline{S}) \rightarrow T$$

where $c \in \mathcal{C}_{\overline{s}\rightarrow s}$, $c(\overline{S})$ is a linear constructor term of sort s, T is a constructor term of sort s, $\overline{x} = \mathcal{V}ar(\overline{S})$, and $c(\overline{S}) \succ T$. We denote by $\mathcal{R}_{\mathcal{C}}$ this set of rewrite rules.

2.2.2 *Defined Symbols.* Given $f \in \mathcal{D}_{\overline{s}\rightarrow s}$, the axioms defining f are *order-sorted conditional rewrite rules* of the form: $\overline{x} : \overline{s} \wedge \overline{V} = \overline{W} \Rightarrow f(\overline{L}) \rightarrow R$ satisfying a *reductivity* condition [Dershowitz and Okada 1990]:
(i) $\overline{x} = \mathcal{V}ar(\overline{L})$ and $f(\overline{L})$ and R are of the same sort,
(ii) $f(\overline{L}) \succ R$
(iii) $\forall V \in \overline{V}, \forall W \in \overline{W} : f(\overline{L})\ (\succ \cup \rhd)\ V, W$.
We call *unconditional* a rewrite rule of the form $\overline{x} : \overline{s} \Rightarrow L \rightarrow R$, and denote by $\mathcal{R}_{\mathcal{D}}$ the set of rules for defined symbols.

To each non-left linear rule $P \Rightarrow L \rightarrow R$, we associate its *linearized* [1] version $P' \wedge P'' \Rightarrow L' \rightarrow R'$, such that L' is linear, $L = L'\sigma$ for some renaming σ, $R = R'\sigma$, $P = P'\sigma$, and $x = y \in P''\ \forall x,\ y \in \mathcal{V}ar(L')$ such that $x\sigma = y\sigma$.

Given a rule $P \Rightarrow L \rightarrow R$, L is called the left-hand side of the rule.

2.3 Order-Sorted Rewriting

We now proceed with the operational semantics, that is, the definition of rewriting order-sorted terms with rules in $\mathcal{R} = \mathcal{R}_{\mathcal{C}} \cup \mathcal{R}_{\mathcal{D}}$. First, we need to define how substitutions operate on order-sorted terms:

Definition 2.1. Given an order-sorted term $(S, \{\overline{x} : \overline{s}\})$ where S is a term over the environment $(S, \{\overline{x} : \overline{s}\})$, the order-sorted substitution $\sigma = \{x_1 \mapsto (t_1, \Gamma_1), \ldots, x_n \mapsto (t_n, \Gamma_n)\}$, is *admissible* if (t_i, Γ_i) have the sort s_i for all $i \in [1..n]$, and $\Gamma = \bigcup_{i\in[1..n]} \Gamma_i$ is an environment. The instantiation of $(S, \{\overline{x} : \overline{s}\})$ by σ is defined as the order-sorted term $(S\{\overline{x} \mapsto \overline{t}\}, \Gamma)$. ◇

Rewriting with respect to $\mathcal{R}$ is order-sorted rewriting as used in OBJ [Futatsugi et al. 1985]:

Definition 2.2. $(S,\Gamma) \longrightarrow^{p}_{L\rightarrow R \texttt{ if } \overline{U:s}\wedge\overline{V}=\overline{W}} (T,\Gamma)$ if
(i) L matches $(S|_p, \Gamma)$ with substitution σ, for all $i \in [1..n] : U_i\sigma$ have the sort s_i, and $\overline{V}\sigma \downarrow_{\mathcal{R}} \overline{W}\sigma$,
(ii) $T = S[R\sigma]_p$.
We will often abuse the notations by writing $S \longrightarrow_{\mathcal{R}} T$, therefore assuming the environment in which a term is rewritten. A term S is *irreducible* (i.e., *not reducible*) by $\mathcal{R}$ if there is no T such that $S \longrightarrow_{\mathcal{R}} T$. A substitution σ is *irreducible* by $\mathcal{R}$ if

[1] This shows that left-linearity is not a real restriction when dealing with conditional rules.

$x\sigma$ is *irreducible* by $\mathcal{R}$ for every variable x of its domain. We also use $\longrightarrow^*_{\mathcal{R}}$, $\longrightarrow^+_{\mathcal{R}}$ and $\longleftrightarrow^*_{\mathcal{R}}$ for, respectively, the reflexive, transitive, the transitive, and the reflexive, symmetric, transitive closures of a rewrite relation $\longrightarrow_{\mathcal{R}}$, $V \downarrow_{\mathcal{R}} W$ for $V \longrightarrow^*_{\mathcal{R}} U \ ^*_{\mathcal{R}}\!\longleftarrow W$ for some U, and $S\downarrow_{\mathcal{R}}$ for a normal form of S via the rewrite system $\mathcal{R}$. ◇

Since the rules in $\mathcal{R}$ are reductive, $\longrightarrow_{\mathcal{R}}$ is terminating and therefore rewriting with $\mathcal{R}$ is decidable.

3. COMPLETENESS AND GROUND CONFLUENCE

When for all possible arguments the result of a defined operator can be expressed with constructors only, we say that this operator is completely defined w.r.t. the constructors.

Definition 3.1. Let $\mathcal{R}$ be a rewrite system. The operator $f \in \mathcal{D}$ is completely defined w.r.t. $\mathcal{C}$ iff for all $T_1, \ldots, T_n$ in $\mathcal{T}(\mathcal{C})$, there exists T in $\mathcal{T}(\mathcal{C})$ such that $f(T_1, \ldots, T_n) \rightarrow^+_{\mathcal{R}} T$. We say that $\mathcal{R}$ is complete iff each defined operator $f \in \mathcal{D}$ is completely defined. ◇

Ground confluence guarantees the property of uniqueness in computation with ground terms.

Definition 3.2. Let $\mathcal{R}$ be a rewrite system. We say that $\mathcal{R}$ is ground confluent iff for any ground terms $U,\ V,\ W \in \mathcal{T}(\mathcal{F})$, if $V \ ^*_{\mathcal{R}}\!\longleftarrow U \longrightarrow^*_{\mathcal{R}} W$, then $V \downarrow_{\mathcal{R}} W$. ◇

The specifications given in Fig. 1 and Fig. 2 are complete and ground confluent as proved in Examples 9.1 and 7.4. Now, if we remove the rule $x < 0 \rightarrow False$ from Fig. 1, then the specification NAT will be incomplete, since, for example, the term $0 < 0$ can not be reduced to a constructor term. If we reformulate the rule $even(x + (y + y)) \rightarrow even(x)$ as $even(x + (y + y)) \rightarrow even(x + y)$, then the specification NAT will not be ground confluent, since, for example, $False \ ^*_{\mathcal{R}}\!\leftarrow even(0 + (s(0) + s(0))) \rightarrow^*_{\mathcal{R}} True$, and $False$ and $True$ are in normal form.

4. COMPUTATION OF INDUCTION SCHEMAS

Now, we will give the ingredients needed to compute an induction schema for a given specification.

Definition 4.1. A term (T, Γ) is *ground reducible* (resp. *ground irreducible*) if $T\gamma$ is reducible (resp. irreducible) for every irreducible admissible ground substitution γ. ◇

Definition 4.2. A sort s is *free* if every ground constructor term of sort s is irreducible. A Cartesian product of sorts is free if its components are. ◇

In the example of Fig. 1, $(s(x), \{x : Nat\})$ is ground irreducible, $(x < y, \{x : Nat, y : Nat\})$ is ground reducible, and Nat is a free sort.

Note that a sort s is free iff every term inhabiting s is ground irreducible. Freeness makes induction easy on free sorts. On the other hand, non-free sorts can be decomposed into free ones.

specification: NAT
sorts Nat, Bool;
constructors:
0 : → Nat;
s _ : Nat → Nat;
True : → Bool;
False : → Bool;
defined functions:
_ + _ : Nat Nat → Nat;
_ ∗ _ : Nat Nat → Nat;
_ < _ : Nat Nat → Bool;
even _ : Nat → Bool;
odd _ : Nat → Bool;
axioms:
(1) 0+x → x;
(2) x+0 → x;
(3) s(x)+y → s(x+y);
(4) x+s(y) → s(x+y);
(5) (x+y)+z → x+(y+z);
(6) 0 ∗ x → 0;
(7) s(x) ∗ y → (x ∗ y)+y;
(8) x ∗ 0 → 0;
(9) x<0 → False;
(10) 0<s(x) → True;
(11) s(x)<s(y) → x<y;
(12) even(0) → True ;
(13) even(s(0)) → False ;
(14) even(s(s(x))) → even(x) ;
(15) even(x)=True ⇒ odd(x) → False ;
(16) even(s(x))=True ⇒ odd(x) → True ;
(17) even(x+x) → True ;
(18) even(s(x+x)) → False ;
(19) even(x+(y+y)) → even(x) ;
(20) odd(s(s(x))) → odd(x) ;
(21) odd(x+x) → False ;
(22) odd(s(x+x)) → True ;
(23) odd(x)=True ∧ odd(y)=True ⇒ odd(x+y) → False ;
(24) even(x)=True ∧ even(y)=True ⇒ even(x+y) → True ;
(25) odd(x)=True ∧ odd(y)=False ⇒ odd(x+y) → True ;
(26) even(x)=True ∧ even(y)=False ⇒ even(x+y) → False ;
(27) even(x ∗ s(s(y))) → even(x ∗ y) ;
(28) even(x)=True ⇒ even(x ∗ y) → True ;
(29) odd(x)=True ∧ odd(y)=True ⇒ even(x ∗ y) → False ;

Fig. 1. A specification of natural numbers

Definition 4.3. Given a sort s, a set S of free subsorts of s is a *cover sort* of s if every irreducible ground constructor term T inhabiting s inhabits a unique sort in S. ◇

Our uniqueness assumption is motivated by efficiency reasons. In the example of Fig. 2, Int has no cover sort, while in the example of Fig. 1, Nat has itself as a trivial cover sort. In Fig. 6, $\{Zero, Neg, Pos\}$ is a cover sort of Int. Cover sorts

specification: INTEGERS
sorts Int, Bool;
constructors:
0 : → Int;
s _ : Int → Int;
p _ : Int → Int;
True : → Bool;
False: → Bool;
defined functions:
− _ : Int → Int;
_ + _: Int Int → Int;
_ ∗ _: Int Int → Int;
_ < _: Int Int → Bool;
axioms:
(a) p(s(x)) → x;
(b) s(p(x)) → x;
(c) - 0 → 0;
(d) - s(x) → p(- x);
(e) - p(x) → s(- x);
(f) 0+x → x;
(g) x+0 → x;
(h) s(x)+y → s(x+y);
(i) p(x)+y → p(x+y);
(j) (x+y)+z → x+(y+z);
(k) 0 ∗ x → 0;
(l) s(x) ∗ y → (x ∗ y)+y;
(m) p(x) ∗ y → (x ∗ y)+ (-y);
(n) (x ∗ y) ∗ z → x ∗ (y ∗ z);
(o) (x+y) ∗ z → (x ∗ z) + (y ∗ z);
(p) x + (y ∗ z) → (x+y) ∗ (x+z);
(q) 0<0 → False;
(r) 0<x=True ⇒ 0<s(x) → True;
(s) 0<x=False ⇒ 0<p(x) → False;
(t) s(x)<y → x<p(y);
(u) p(x)<y → x<s(y);

Fig. 2. A specification of integers

allow us to have constructor symbols which are free in some subset of a cover sort, and defined in the complement. For example, s is free on $Zero \cup Pos$ and defined on Neg.

Definition 4.4. We call *structural scheme* of free sort s, denoted by $\mathcal{SC}(s)$, the set of terms $c(x_1, \ldots, x_n)$ such that c is a constructor function with codomain s, and arity n, and $x_1, \ldots, x_n$ are distinct variables. Note that $\{x_1, \ldots, x_n\}$ is empty if c is a constant symbol. ◇

For the example of Fig. 1, $\mathcal{SC}(Nat) = \{0, s(x)\}$.

Definition 4.5. The set $\mathcal{IndPos}(f, \mathcal{R})$ of *induction positions* of $f \in \mathcal{D}$ is the set of non-root positions p such that there exists in $\mathcal{R}$ a rewrite rule of left-hand side $f(\overline{L})$, such that for each $L_i \in \overline{L}$: $L_i \in \mathcal{T}(\mathcal{C}, \mathcal{X})$, and p is a position in $f(\overline{L})$ of a non-variable subterm.

Given $\mathcal{R}$, the set of *induction variables* of a term T, written $\mathcal{I}nd\mathcal{V}ar(T,\mathcal{R})$, is the subset of $\mathcal{V}ar(T)$ whose elements inhabit a free sort s and occur in a subterm of T of the form $f(\overline{S})$, such that for each $S_i \in \overline{S}\ :\ S_i \in \mathcal{T}(\mathcal{C},\mathcal{X})$, at an induction position of f. ◇

In the example of Fig. 1, $+, *, <, even, odd$ have the respective sets of induction positions $\{1,2\},\{1,2\},\{1,2\},\{1,1.1\},\{1,1.1\}$. In the Example of Fig. 6, x is an induction variable of $(p(x),\{x : Pos\})$.

5. HOW TO CHECK COMPLETENESS AND GROUND CONFLUENCE

The main idea behind our test for completeness and ground confluence is to compute a *pattern tree* for every defined operator $f : s_1 \times \ldots \times s_n \to s$ in $\mathcal{D}$. A pattern tree for f is a tree whose nodes are labelled by patterns, whose root is labelled by the initial pattern $(f(x_1,\ldots,x_n),\{x_1 : s_1,\ldots,x_n : s_n\})$ where n is the arity of f and $x_1,\ldots,x_n$ are distinct variables, and such that the successors of any internal node labelled by the pattern $(f(\overline{T}),\Gamma)$ are obtained by either covering the sort or the set of values (i.e., ground constructor terms) of an induction variable in $f(\overline{T})$. The restriction of induction variables allows us to build a pattern tree which captures the structure of the axioms. To compute pattern trees, we use the following notions.

Definition 5.1. A *pattern* [2] is an order-sorted term $(f(\overline{T}),\Gamma)$ such that $f \in \mathcal{D}$ and $T_i \in \mathcal{T}(\mathcal{C},\mathcal{X})$ for every $T_i \in \overline{T}$. ◇

A formula φ is a *deductive theorem* of $\mathcal{R}$ if it is valid in any model of $\mathcal{R}$. This will be denoted by $\mathcal{R} \models \varphi$. A formula φ is an *inductive theorem* (or an *inductive conjecture*) of $\mathcal{R}$ if it is valid in the initial model of $\mathcal{R}$. This will be denoted by $\mathcal{R} \models_{\mathcal{I}nd} \varphi$.

Definition 5.2. A term (T,Γ) is *strongly reducible* if
(i) T is reducible, or
(ii) the formula $P_1\sigma_1 \vee \ldots \vee P_n\sigma_n$ is an inductive theorem of $\mathcal{R}$, where $\{P_i \Rightarrow L_i \to R_i\}_{i\in[1..n]}$ is the set of linearized rules in $\mathcal{R}$ such that each L_i matches T with the substitution σ_i. ◇

If $\mathcal{R}$ is ground convergent (i.e. is ground confluent and terminates), then every *strongly reducible* term is ground reducible.

In Fig. 1, the pattern $(odd(x),\{x : Nat\})$ is strongly reducible since the left-hand sides of the axioms (15) and (16) match $odd(x)$ and $even(x) = True \vee even(s(x)) = True$ is an inductive theorem. However, the pattern $(even(x),\{x : Nat\})$ is not strongly reducible since there is no axiom whose left-hand side matches $even(x)$. In Fig. 2, the pattern $(0 < s(x),\{x : Int\})$ is not strongly reducible since $x : Int \wedge 0 < x = True$ is not an inductive theorem.

Definition 5.3. A formula $\varphi = \overline{U} = \overline{V} \Rightarrow L_1 = R_1 \vee \ldots \vee L_n = R_n$ is a *joinable-inductive theorem* of $\mathcal{R}$ (or a *joinable-inductive conjecture* or *inductively joinable* w.r.t. $\mathcal{R}$) iff for each ground substitution σ, if $\overline{U}\sigma \downarrow_{\mathcal{R}} \overline{V}\sigma$, then there exists $i \in [1..n]$ such that $L_i\sigma \downarrow_{\mathcal{R}} R_i\sigma$. ◇

[2] The environment Γ will be omitted if the specification is many sorted.

Of course, the two notions of joinable-inductive and inductive conjectures coincide if $\mathcal{R}$ is ground confluent. However, they can differ otherwise. Consider for instance $\mathcal{R} = \{c \to a,\ c \to b\}$. The conjecture $a = b$ is an inductive theorem (since $a \longleftrightarrow^*_{\mathcal{R}} b$) but it is not joinable inductive theorem (as a and b are in normal form w.r.t. $\mathcal{R}$). On the other hand, the conjecture $a = b \Rightarrow f(x) = c$ is a joinable inductive theorem but not an inductive theorem (for the same reasons).

In [Bouhoula and Jacquemard 2007], we have developed a new method for checking inductive joinability for non-confluent equational specifications made of conditional and constrained rewrite rules. The constraints are interpreted over constructor terms, and may express syntactic equality, disequality, ordering and also membership to a fixed tree language. Constrained equational axioms between constructor terms are supported and can be used in order to specify complex data structures like sets, sorted lists, trees and powerlists. Membership constraints permit in particular to express problems of verification of trace properties for systems and communication protocols.

We have shown (see Theorem 1 and Theorem 2 in [Bouhoula and Jacquemard 2007]) that our inference system is sound and allows to refute false conjectures, even if the axioms are not complete and not ground confluent.

A restricted version of this method has been implemented in the prover SPIKE and allows to check inductive joinability for non-confluent and unconstrained conditional specifications with free constructors.

Definition 5.4. A term (T, Γ) is *non-ambiguous* if for all rules $P \Rightarrow L \to R$ and $P' \Rightarrow L' \to R'$ in $\mathcal{R}$ such that $T = L\sigma = L'\sigma'$, the formula $P\sigma \wedge P'\sigma' \Rightarrow R\sigma = R'\sigma'$ is a joinable-inductive theorem of $\mathcal{R}$ ◇

Let $\mathcal{R}$ be the following rewriting system:

$$\begin{aligned} f(x) &\to 0 \\ f(x) &\to g(x) \\ g(0) &\to 0 \\ g(x) = 0 \Rightarrow g(s(x)) &\to 0 \end{aligned}$$

$f(x)$ is non-ambiguous since there are only two rules in $\mathcal{R}$ whose left-hand sides match $f(x)$ and $g(x) = 0$ is a joinable-inductive theorem.

In Fig. 1, the pattern $(odd(x), \{x : Nat\})$ is non-ambiguous, since

$$even(x) = True \wedge even(s(x)) = True \Rightarrow True = False$$

is a joinable-inductive theorem. If we reformulate the rule $x + 0 \to x$ as $x + 0 \to 0$, then the pattern $(s(0) + 0, \emptyset)$ will be ambiguous since $s(0 + 0) = 0$ is not a joinable-inductive theorem.

Definition 5.5. If the left-hand side L of a rule $P \Rightarrow L \to R$ unifies, via a most general unifier σ, with a non-variable subterm S at position u in a left-hand side L' of a rule $P' \Rightarrow L' \to R'$, then the conditional equation

$$P\sigma \wedge P'\sigma \Rightarrow L'\sigma[R\sigma]_u = R'\sigma$$

is called a *critical pair* of the two rules, where $L'\sigma[R\sigma]_u$ is obtained by replacing S in L' by R and applying σ.

Let $\mathcal{R}$ and $\mathcal{R}'$ be two rewrite systems, we denote by $\mathcal{CP}(\mathcal{R}, \mathcal{R}')$ the set of critical pairs between a rule in $\mathcal{R}$ and a rule in $\mathcal{R}'$.

A critical pair $P \Rightarrow L = R$ is *trivial* if L is identical to R. ◇

The conditional equation $even(x + x) = True \Rightarrow False = False$ is a critical pair of the rules (15) and (21). This critical pair is trivial.

6. SPECIFICATIONS WITH FREE CONSTRUCTORS

Our procedure for checking the completeness and ground confluence of $\mathcal{R}$ is presented in Fig. 3 as a set of inference rules operating on ($\mathcal{P}$, $\mathcal{MP}$, $\mathcal{AP}$, $\mathcal{UR}$, $\mathcal{NP}$), where $\mathcal{P}$ is a set of patterns labelling the leaves of the tree constructed so far, $\mathcal{MP}$ is its set of strongly irreducible (i.e., not strongly reducible) leaves, $\mathcal{AP}$ is its set of ambiguous leaves, $\mathcal{UR}$ is the set of rules in $\mathcal{R}$ whose left-hand sides match the leaves of the tree, and $\mathcal{NP}$ is the set of patterns for which we still must compute their pattern trees.

Success applies when the sets $\mathcal{P}$, $\mathcal{MP}$, $\mathcal{AP}$ and $\mathcal{NP}$ are empty and all the rules in $\mathcal{R}-\mathcal{UR}$ are inductive theorems of $\mathcal{UR}$. We can then conclude that $\mathcal{R}$ is complete and ground confluent (see Theorem 6.2). **Ground Confluence** applies when the sets $\mathcal{P}$, $\mathcal{AP}$ and $\mathcal{NP}$ are empty and all the rules in $\mathcal{R}-\mathcal{UR}$ are inductive theorems of $\mathcal{UR}$. We can then conclude that $\mathcal{R}$ is ground confluent (see Theorem 6.2). **Missing Patterns** applies when the sets $\mathcal{P}$ and $\mathcal{NP}$ are empty, but $\mathcal{MP}$ is not. In this case, $\mathcal{R}$ is not complete (see Theorem 6.2), the user is prompted to complete the specification at the patterns in $\mathcal{MP}$. **Ambiguous Patterns** applies when the sets $\mathcal{P}$ and $\mathcal{NP}$ are empty, but $\mathcal{AP}$ is not. Then, $\mathcal{R}$ is not ground confluent (see Theorem 6.2), the user is prompted to correct the specification to make the patterns in $\mathcal{AP}$ non-ambiguous. **Non-Valid Rules** applies when the sets $\mathcal{P}$, $\mathcal{MP}$, $\mathcal{AP}$ and $\mathcal{NP}$ are empty, but some rules in $\mathcal{R}-\mathcal{UR}$ are not an inductive theorems of $\mathcal{UR}$. In this case, $\mathcal{R}$ is not ground confluent (see Theorem 6.2), the user is prompted to correct the specification to make the rules in $\mathcal{R}-\mathcal{UR}$ inductively valid. **Decompose Variable** applies when a pattern $(T[x], \{x : s\} \cup \Gamma)$ has an induction variable x. Then, it instantiates x by terms in a structural scheme of s. **Checking Leaf** applies when a leaf (T, Γ) is found such that **Decompose Variable** can not be applied. A strategy of efficient use of the inference system $\mathcal{CGC}$ is given in Fig. 4. Of course, the following theorems are not depending on this strategy. In the following, we say $\mathcal{CGC}$ succeeds if the rule **Success** applies.

The height of the pattern tree is bounded. This result is shown by the following lemma:

LEMMA 6.1. *The pattern trees computed by the inference system $\mathcal{CGC}$ are finite.*

PROOF. Let $f \in \mathcal{D}$, the set of rules in $\mathcal{R}$ which have the function symbol f at the top is finite. This means that the set $\mathcal{I}nd\mathcal{P}os(f)$ is finite too. As a consequence the set $\mathcal{I}nd\mathcal{V}ar(T, \mathcal{R})$ decreases during the construction of the tree since consecutive grafts in the same branch of the tree are made at deeper and deeper positions. Consequently, the height of the pattern tree is bounded. □

We can now address the soundness of our inference system:

THEOREM 6.2 (SOUNDNESS). *Let $\mathcal{R}$ be a reductive conditional rewriting system. If $\mathcal{CGC}$ succeeds then $\mathcal{R}$ is complete and ground confluent. If the rule* **Ground**

Initialization:
$$\frac{}{(\emptyset,\ \emptyset,\ \emptyset,\ \emptyset,\ \bigcup_{f\in\mathcal{D}}\ \{f(\overline{x})\})}$$

Checking New Function:
$$\frac{(\emptyset,\ \mathcal{MP},\ \mathcal{AP},\ \mathcal{UR},\ \mathcal{NP}\cup\{T\})}{(\{T\},\ \mathcal{MP},\ \mathcal{AP},\ \mathcal{UR},\ \mathcal{NP})}$$

Decompose Variable:
$$\frac{(\mathcal{P}\cup\{(T[x]_p,\{x:s\}\cup\Gamma)\},\ \mathcal{MP},\ \mathcal{AP},\ \mathcal{UR},\ \mathcal{NP})}{(\mathcal{P}\cup\bigcup_{i\in I}\{T[S_i]_p\},\ \mathcal{MP},\ \mathcal{AP},\ \mathcal{UR},\ \mathcal{NP})}$$
`if` $x\in\mathcal{I}nd\mathcal{V}ar(T,\ \mathcal{R})$

`where` $\{S_i\}_{i\in I}$ is a structural scheme of s

Checking Leaf:
$$\frac{(\mathcal{P}\cup\{T\},\ \mathcal{MP},\ \mathcal{AP},\ \mathcal{UR},\ \mathcal{NP})}{(\mathcal{P},\ \mathcal{MP}\cup\mathcal{MP}',\ \mathcal{AP}\cup\mathcal{AP}',\ \mathcal{UR}\cup\mathcal{UR}',\ \mathcal{NP})}$$
`if` $\mathcal{I}nd\mathcal{V}ar(T,\ \mathcal{R})=\emptyset$

`where` $\begin{cases}\mathcal{MP}' = \text{ if } T \text{ is strongly reducible then } \emptyset \text{ else } \{T\}\\ \mathcal{AP}' = \text{ if } T \text{ is ambiguous then } \{T\} \text{ else } \emptyset\\ \mathcal{UR}' \text{ is the set of rules in } \mathcal{R}\\ \qquad \text{whose left-hand sides matche } T\end{cases}$

Success:
$$\frac{(\emptyset,\ \emptyset,\ \emptyset,\ \mathcal{UR},\ \emptyset)}{\mathcal{R} \text{ is complete and ground confluent}}$$
`if` $\mathcal{UR}\models_{\mathcal{I}nd}(\mathcal{R}-\mathcal{UR})$

Ground Confluence:
$$\frac{(\emptyset,\ \mathcal{MP},\ \emptyset,\ \mathcal{UR},\ \emptyset)}{\mathcal{R} \text{ is ground confluent}}$$
`if` $\mathcal{UR}\models_{\mathcal{I}nd}(\mathcal{R}-\mathcal{UR})$

Missing Patterns:
$$\frac{(\emptyset,\ \mathcal{MP},\ \mathcal{AP},\ \mathcal{UR},\ \emptyset)}{\mathcal{MP}}$$
% $\mathcal{R}$ is not complete
`if` $\mathcal{MP}\neq\emptyset$

Ambiguous Patterns:
$$\frac{(\emptyset,\ \mathcal{MP},\ \mathcal{AP},\ \mathcal{UR},\ \emptyset)}{\mathcal{AP}}$$
% $\mathcal{R}$ is not ground confluent
`if` $\mathcal{AP}\neq\emptyset$

Non-Valid Rules:
$$\frac{(\emptyset,\ \emptyset,\ \emptyset,\ \mathcal{UR},\ \emptyset)}{\mathcal{NVR}}$$
% $\mathcal{R}$ is not ground confluent
`if` $\mathcal{NVR}\subseteq(\mathcal{R}-\mathcal{UR})$ and $\mathcal{UR}\not\models_{\mathcal{I}nd}\mathcal{NVR}$

Fig. 3. $\mathcal{CGC}$: Rules for completeness and ground confluence of $\mathcal{R}$

```
CGC(P, MP, AP, UR, NP) =
  begin
    Initialisation;
    while NP ≠ ∅ do
        Checking New Function;
        while ∃T ∈ P such that IndVar(T, R) ≠ ∅
          do Decompose Variable enddo;
        while P ≠ ∅ do Checking Leaf enddo
      enddo;
    if MP = ∅ and AP = ∅ and UR ⊨_Ind (R − UR)
      then R is complete and ground confluent
      else
        if AP = ∅ and UR ⊨_Ind (R − UR)
          then R is ground confluent
          else
            begin
              if MP ≠ ∅ then R is not complete;
              if (AP ≠ ∅) or (MP = ∅ and AP = ∅ and UR ⊭_Ind (R − UR))
                then R is not ground confluent
            end
  end.
```

Fig. 4. A strategy of use of the inference system $\mathcal{CGC}$

Confluence *applies, then* $\mathcal{R}$ *is ground confluent. If the rule* **Missing Patterns** *applies, then* $\mathcal{R}$ *is not complete. If one of the rules* **Ambiguous Patterns** *and* **Non-Valid Rules** *applies, then* $\mathcal{R}$ *is not ground confluent.*

PROOF. The proof is given in the Appendix. □

If the rewrite rules in $\mathcal{R}$ are unconditional then we have the following result:

THEOREM 6.3. *Let* $\mathcal{R}$ *be a reductive unconditional rewriting system. If all the leaves of the pattern trees are strongly reducible, then* $\mathcal{R}$ *is complete.*

PROOF. Let us show that $\mathcal{R}$ is complete. Let $T \in \mathcal{T}(\mathcal{F})$ and T' be the normal form of T with respect to $\mathcal{R}$. If T' is a constructor term, we are done. Otherwise, T' must contain a subterm T'' of the form $g(\overline{T})$ where $g \in \mathcal{D}$ and for all $i \in [1..n]$, $T_i \in \mathcal{T}(\mathcal{C})$. Since the leaves of the pattern trees exhaust all cases by construction, this subterm must be an instance of a leaf S: $S\sigma = T''$, where σ is a ground substitution over $\mathcal{T}(\mathcal{C})$. Since the rewrite rules in $\mathcal{R}$ are unconditional, then S must be strongly reducible by case (i) of Definition 5.2. It follows that S is reducible and therefore T'' is also reducible. This contradicts the fact that T' is in normal form. □

Our inference system is also complete.

THEOREM 6.4 (COMPLETENESS). *Let* $\mathcal{R}$ *be a reductive conditional rewriting system. Assume an oracle for deciding (joinable) inductive conjectures. If* $\mathcal{R}$ *is complete and ground confluent, then* $\mathcal{CGC}$ *will succeed.*

PROOF. If $\mathcal{R}$ is complete and ground confluent, then after the application of the rule **Initialization** and the saturation of the application of the rules **Check-**

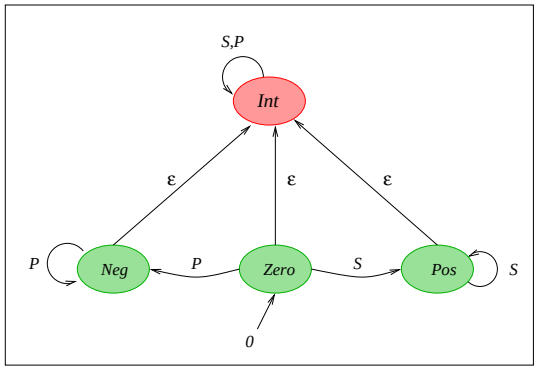

Fig. 5. A normal form automaton

ing New Function, **Decompose Variable** and **Checking Leaf** we must obtain $\mathcal{MP} = \emptyset$ (otherwise, by Theorem 6.2, we conclude that $\mathcal{R}$ is not complete), $\mathcal{AP} = \emptyset$ (otherwise, by Theorem 6.2, we conclude that $\mathcal{R}$ is not ground confluent) and $\mathcal{UR} \models_{\mathcal{I}nd} (\mathcal{R} - \mathcal{UR})$ (otherwise, by Theorem 6.2, we conclude that $\mathcal{R}$ is not ground confluent). Hence **Success** applies. □

7. SPECIFICATIONS WITH NON-FREE CONSTRUCTORS

Our procedure for checking completeness and ground confluence uses a notion of completeness of the specification of constructors.

7.1 Complete Specifications of Constructors

Constructor symbols may be free for some sorts, and completely defined in all other cases. For example, s is free on $Zero \cup Pos$ and defined on Neg.

Definition 7.1. A constructor c is *free* at sort $\overline{s}$ if $c(\overline{x})$ is ground irreducible in the environment $\{\overline{x} : \overline{s}\}$.

A constructor c is *defined* at sort $\overline{s}$ if $c(\overline{x})$ is ground reducible in the environment $\{\overline{x} : \overline{s}\}$.

A constructor c is *complete* at sort $\overline{s}$ if there exists a cover sort S of $\overline{s}$ such that c is free at all sorts in some subset of S and defined at all sorts in its complement.

A specification of constructors is *complete* if each constructor is complete. ◇

We can easily transform a given specification of constructors into a complete one provided that the rewrite rules for constructors are unconditional and left-linear (see [Bouhoula and Jouannaud 2001]). We start first by computing the tree automaton recognizing the set of irreducible ground terms (see for example the tree automaton given in Fig. 5 which recognizes the set of irreducible ground terms of the specification given in Fig. 2). Then, we transform each ε-transition into a subsort relation, and we transform each transition between sorts into a signature declaration (see Fig. 6).

specification: INTEGERS
sorts: Zero, Pos, Neg, Int, Bool;
subsorts: Zero, Neg, Pos < Int;
constructors:

0	:	→ Int;
0	:	→ Zero;
s _	: Int	→ Int;
s _	: Zero	→ Pos;
s _	: Pos	→ Pos;
p _	: Int	→ Int;
p _	: Zero	→ Neg;
p _	: Neg	→ Neg;
True	:	→ Bool;
False	:	→ Bool;

axioms:
x: Int ⇒ p(s(x)) → x;
x: Int ⇒ s(p(x)) → x;

Fig. 6. A complete specification of constructors

7.2 Inference Rules

Now, we will define a new inference system $\mathcal{CGC}'$ from $\mathcal{CGC}$ by adding the rules **Non-Valid Critical Pairs** and **Decompose Sort**, and reformulating the rules **Checking Leaf**, **Success**, **Ground Confluence** and **Non-Valid Rules** (see Fig. 7).

Decompose Sort applies when a pattern $(T, \{x : s\} \cup \Gamma)$ is not strongly reducible and has a variable ranging over a non-free sort. **Non-Valid Critical Pairs** applies when the sets $\mathcal{P}$, $\mathcal{AP}$ and $\mathcal{NP}$ are empty, but some critical pairs between a rule in $\mathcal{UR}$ and a rule in $\mathcal{R}_\mathcal{C}$ are not joinable-inductive theorems of $\mathcal{UR} \cup \mathcal{R}_\mathcal{C}$. In this case, the user is prompted to correct the specification to make these critical pairs valid. A strategy of use of the inference system $\mathcal{CGC}'$ is given in Fig. 8.

We can now address the soundness of our inference system $\mathcal{CGC}'$:

THEOREM 7.2 (SOUNDNESS). *Let $(\mathcal{C}, \mathcal{R}_\mathcal{C})$ be a complete specification of constructors and $\mathcal{R}$ be a reductive conditional rewriting system. Assume that $\mathcal{R}_\mathcal{C}$ is ground confluent* [3]*. If the rule* **Success** *applies, then $\mathcal{R}$ is complete and ground confluent. If the rule* **Ground Confluence** *applies, then $\mathcal{R}$ is ground confluent. If the rule* **Missing Patterns** *applies, then $\mathcal{R}$ is not complete. If one of the rules* **Ambiguous Patterns**, **Non-Valid Rules** *or* **Non-Valid Critical Pairs** *applies, then $\mathcal{R}$ is not ground confluent.*

PROOF. To show that $\mathcal{R}$ is ground confluent, it is sufficient to show that $\mathcal{UR} \cup \mathcal{R}_\mathcal{C}$ is ground confluent since the rules in $\mathcal{R}_\mathcal{D} - \mathcal{UR}$ are inductive theorems of $\mathcal{UR} \cup \mathcal{R}_\mathcal{C}$. Since $\mathcal{AP}$ is empty (i.e., all the leaves of the pattern trees are non-ambiguous and do not contain any induction variable and the sorts of the variables in the leaves are free) and all the rules in $\mathcal{UR}$ are of the form $P \Rightarrow f(\overline{T}) \to R$ where $f \in \mathcal{D}$ and $T_i \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ for each $T_i \in \overline{T}$, then all critical pairs between rules in $\mathcal{UR}$ are trivial

[3] We will show in Section 7.3 that we can avoid the test of the ground confluence of $\mathcal{R}_\mathcal{C}$.

Decompose Sort:

$$\frac{(\mathcal{P} \cup \{(T[x], \{x : s\} \cup \Gamma)\},\ \mathcal{MP},\ \mathcal{AP},\ \mathcal{UR},\ \mathcal{NP})}{(\mathcal{P} \cup \bigcup_{i \in I} \{(T[x], \{x : s_i\} \cup \Gamma)\},\ \mathcal{MP},\ \mathcal{AP},\ \mathcal{UR},\ \mathcal{NP})}$$

if s is non-free

where $\{s_i\}_{i \in I}$ is a cover sort of s

Checking Leaf:

$$\frac{(\mathcal{P} \cup \{\overbrace{(T, \{x_1 : s_1, \ldots, x_n : s_n\})}^{T'}\},\ \mathcal{MP},\ \mathcal{AP},\ \mathcal{UR},\ \mathcal{NP})}{(\mathcal{P},\ \mathcal{MP} \cup \mathcal{MP}',\ \mathcal{AP} \cup \mathcal{AP}',\ \mathcal{UR} \cup \mathcal{UR}',\ \mathcal{NP})}$$

if $\begin{cases} \mathcal{IndVar}(T',\ \mathcal{R}) = \emptyset \\ s_1, \ldots, s_n \text{ are free} \end{cases}$

where $\begin{cases} \mathcal{MP}' = \text{ if } T' \text{ is strongly reducible then } \emptyset \text{ else } \{T'\} \\ \mathcal{AP}' = \text{ if } T' \text{ is ambiguous then } \{T'\} \text{ else } \emptyset \\ \mathcal{UR}' \text{ is the set of rules in } \mathcal{R} \\ \qquad \text{whose left-hand sides matche } T' \end{cases}$

Success:

$$\frac{(\emptyset,\ \emptyset,\ \emptyset,\ \mathcal{UR},\ \emptyset)}{\mathcal{R} \textit{ is complete and ground confluent}}$$

if $\begin{cases} \mathcal{UR} \cup \mathcal{R}_{\mathcal{C}} \models_{\mathcal{I}nd} (\mathcal{R}_{\mathcal{D}} - \mathcal{UR}) \\ \mathcal{CP}(\mathcal{UR}, \mathcal{R}_{\mathcal{C}}) \text{ are inductively joinable w.r.t. } \mathcal{UR} \cup \mathcal{R}_{\mathcal{C}} \end{cases}$

Ground Confluence:

$$\frac{(\emptyset,\ \mathcal{MP},\ \emptyset,\ \mathcal{UR},\ \emptyset)}{\mathcal{R} \textit{ is ground confluent}}$$

if $\begin{cases} \mathcal{UR} \cup \mathcal{R}_{\mathcal{C}} \models_{\mathcal{I}nd} (\mathcal{R}_{\mathcal{D}} - \mathcal{UR}) \\ \mathcal{CP}(\mathcal{UR}, \mathcal{R}_{\mathcal{C}}) \text{ are inductively joinable w.r.t. } \mathcal{UR} \cup \mathcal{R}_{\mathcal{C}} \end{cases}$

Non-Valid Rules:

$$\frac{(\emptyset,\ \emptyset,\ \emptyset,\ \mathcal{UR},\ \emptyset)}{\mathcal{NVR}}$$

if $\mathcal{NVR} \subseteq (\mathcal{R}_{\mathcal{D}} - \mathcal{UR})$ and $\mathcal{UR} \cup \mathcal{R}_{\mathcal{C}} \not\models_{\mathcal{I}nd} \mathcal{NVR}$

Non-Valid Critical Pairs:

$$\frac{(\emptyset,\ \mathcal{MP},\ \emptyset,\ \mathcal{UR},\ \emptyset)}{\mathcal{CP}(\mathcal{UR}, \mathcal{R}_{\mathcal{C}})}$$

if $\mathcal{CP}(\mathcal{UR}, \mathcal{R}_{\mathcal{C}})$ are not inductively joinable w.r.t. $\mathcal{UR} \cup \mathcal{R}_{\mathcal{C}}$

Fig. 7. $\mathcal{CGC}'$: Rules for completeness and ground confluence of $\mathcal{R}$

or joinable-inductive theorems. On the other hand, all critical pairs between a rule in $\mathcal{UR}$ and a rule in $\mathcal{R}_{\mathcal{C}}$ are joinable-inductive theorems of $\mathcal{UR} \cup \mathcal{R}_{\mathcal{C}}$. Therefore, we conclude that $\mathcal{UR}$ is ground confluent since $\mathcal{R}_{\mathcal{C}}$ is ground confluent by assumption.

The remainder of the proof is similar to the proof of Theorem 6.2. □

The inference system $\mathcal{CGC}'$ is also complete.

THEOREM 7.3 (COMPLETENESS). *Let $\mathcal{R}$ be a reductive conditional rewriting system. Assume an oracle for deciding (joinable) inductive conjectures. If $\mathcal{R}$ is complete and ground confluent, then $\mathcal{CGC}'$ will succeed.*

PROOF. The proof is similar to the proof of Theorem 6.4. □

```
CGC'(P, MP, AP, UR, NP) =
  begin
    Initialisation;
    while NP ≠ ∅ do
      Checking New Function;
      while ∃(T, {x : s} ∪ Γ) ∈ P such that (x is an induction variable) or (s is non-free)
        do
          if x is an induction variable then Decompose Variable;
          if s is non-free then Decompose Sort
        enddo;
      while P ≠ ∅ do Checking Leaf enddo
    enddo;
    % We assume that R_C is ground confluent
    CP ← CP(UR, R_C);
    if MP = ∅ and AP = ∅ and UR ∪ R_C |=_Ind (R_D − UR)
        and CP are inductively-joinable w.r.t. UR ∪ R_C
      then R is complete and ground confluent
      else
        if AP = ∅ and UR ∪ R_C |=_Ind (R_D − UR)
            and CP are inductively-joinable w.r.t. UR ∪ R_C
          then R is ground confluent
          else
            begin
              if MP ≠ ∅ then R is not complete;
              if (AP ≠ ∅) or (CP are not inductively joinable w.r.t. UR ∪ R_C) or
                    (MP = ∅ and AP = ∅ and CP are inductively-joinable w.r.t. UR ∪ R_C
                        and UR ∪ R_C |≠_Ind (R_D − UR))
                then R is not ground confluent
            end
  end.
```

Fig. 8. A strategy of use of the inference system $\mathcal{CGC}'$

Example 7.4. Let $\mathcal{SP}$ be the specification obtained by adding to the specification given in Fig. 2, the complete specification of constructors given in Fig. 6. Let us prove that $\mathcal{SP}$ is complete and ground confluent. We start by the computation of pattern trees of defined functions $-$, $+$, $*$, $<$ (see for example the pattern tree of $<$ given in Fig. 9). All the leaves of the trees are strongly reducible and non-ambiguous. On the other hand, rules j, n, o and p are inductive theorems w.r.t. $\mathcal{UR} = \mathcal{R} - \{\mathsf{j}, \mathsf{n}, \mathsf{o}, \mathsf{p}\}$, and we can easily check that all critical pairs between a rule in $\mathcal{UR}$ and a rule in $\mathcal{R}_\mathcal{C}$ are joinable-inductive theorems of $\mathcal{UR} \cup \mathcal{R}_\mathcal{C}$. Then, by Theorem 7.2, we conclude that $\mathcal{R}$ is complete and ground confluent. Note that the patterns $(0 < p(z), \{z : Neg\})$ and $(0 < s(z), \{z : Pos\})$ are strongly reducible since we have $\mathcal{R} \models_{\mathcal{I}nd} z : Neg \Rightarrow 0 < z = False$ and $\mathcal{R} \models_{\mathcal{I}nd} z : Pos \Rightarrow 0 < z = True$.

7.3 How to Avoid the Computation of Critical Pairs

In order to check ground confluence with the inference system $\mathcal{CGC}'$, we must compute all critical pairs between a rule in $\mathcal{UR}$ and a rule in $\mathcal{R}_\mathcal{C}$. The number of generated critical pairs is very small in practice. However, we can avoid the com-

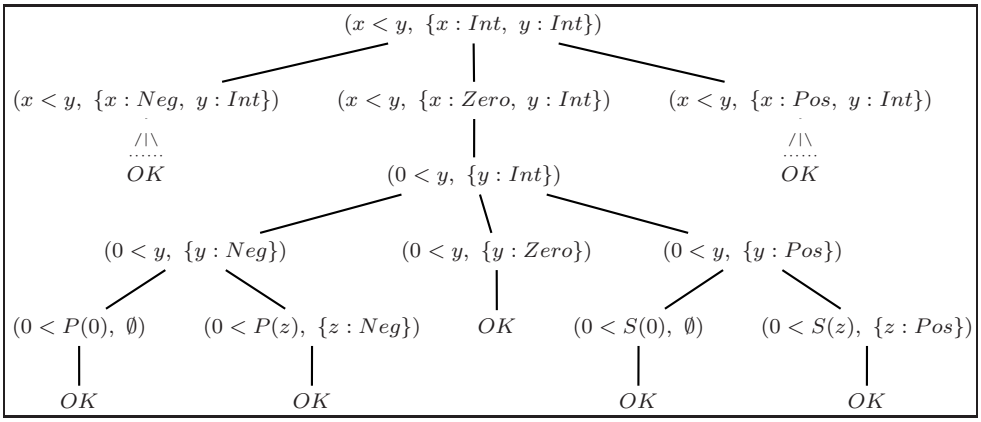

Fig. 9. The pattern tree of $<$

putation of critical pairs as well as the test of the ground confluence of $\mathcal{R}_{\mathcal{C}}$, by transforming the specification with non-free constructors into an order-sorted specification where every function symbol is either a free constructor or a completely defined function. Indeed, in Section 7.1, we have transformed the automaton which describes the set of ground terms in normal form into new subsort relations and signature declarations, but we have not modified the axioms.

Now, we will also transform the axioms, as in [Comon 1989], by considering the new subsorts. See for example the specification [4] given in Fig. 10 which is obtained from the specification given in Fig. 2 (we will simply specify the functions $-$, $+$, $*$). Now, we can apply the inference system $\mathcal{CGC}$ to the obtained specification, and we conclude that the new specification is complete and ground confluent.

8. PARAMETERIZED SPECIFICATIONS

Parameterization is very important for building up larger data types and software systems from generic specifications in a highly reusable way.

Definition 8.1. A *parameterized specification* is a pair $\mathcal{PS} = (\mathcal{P}, \mathcal{B})$ with $\mathcal{P} \subseteq \mathcal{B}$. We call $\mathcal{P} = (\mathcal{F}_{\mathcal{P}}, \mathcal{E}_{\mathcal{P}})$ the *parameter* specification, and $\mathcal{B} = (\mathcal{F}_{\mathcal{B}}, \mathcal{E}_{\mathcal{B}})$ the *body* specification, where $\mathcal{E}_{\mathcal{P}}$ is the set of parameter *constraints* consisting of equational clauses over $\mathcal{F}_{\mathcal{P}}$, and $\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}$ is the set of axioms of the parameterized specification. We assume that the axioms in $\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}$ are reductive rewrite rules over $\mathcal{F}_{\mathcal{B}}$. $\diamond$

An actualization of the parameter theory $\mathcal{E}_{\mathcal{P}}$ is a model $\mathcal{A}$ of $\mathcal{E}_{\mathcal{P}}$. In order to be able to integrate an actualization $\mathcal{A}$ of the parameter theory into the rewrite process, we describe $\mathcal{A}$ by its so-called *diagram* [Ehrig and Mahr 1985]. For this reason we enrich the signatures by adding new constants $\underline{a}$ for each element a of the carrier set A of $\mathcal{A}$. Let $\mathcal{N}(\mathcal{A})$ be the set of new constants and let $\mathcal{F}(\mathcal{A}) = \mathcal{F} \cup \mathcal{N}(\mathcal{A})$. The diagram $\mathcal{D}(\mathcal{A})$ of $\mathcal{A}$ is the set of rules $f(\underline{a}_1, \ldots, \underline{a}_n) \rightarrow \underline{a}$ such that $f \in \mathcal{F}_{\mathcal{P}}$; $a_i,\ a \in A$ and $f^{\mathcal{A}}(a_1, \ldots, a_n) = a$. We denote by $\mathcal{E}_{\mathcal{B}}(\mathcal{A})$ the set $\mathcal{E}_{\mathcal{B}} \cup \mathcal{D}(A)$.

[4] When the sort of a variable is not mentioned, it must be understood that it has the greatest sort of its connected component.

specification: INTEGERS
sorts: Zero, Pos, Neg, Int;
subsorts: Zero, Neg, Pos < Int;
constructors:
$0'$: → Zero;
s' _ : Zero → Pos;
s' _ : Pos → Pos;
p' _ : Zero → Neg;
p' _ : Neg → Neg;
defined functions:
0 : → Int;
s _ : Int → Int;
p _ : Int → Int;
− _ : Int → Int;
_ + _ : Int Int → Int;
_ ∗ _ : Int Int → Int;
axioms:
$0 \to 0'$;
x:Zero ⇒ p(x) → p'(x);
x:Neg ⇒ p(x) → p'(x);
x:Zero ⇒ p(s'(x)) → x;
x:Pos ⇒ p(s'(x)) → x;
x:Zero ⇒ s(x) → s'(x);
x:Pos ⇒ s(x) → s'(x);
x:Zero ⇒ s(p'(x)) → x;
x:Neg ⇒ s(p'(x)) → x;
- $0' \to 0'$;
x:Zero ⇒ - s'(x) → p(- x);
x:Pos ⇒ - s'(x) → p(- x);
x:Zero ⇒ - p'(x) → s(- x);
x:Neg ⇒ - p'(x) → s(- x);
$0'$+x → x;
x+$0'$ → x;
x:Zero ⇒ s'(x)+y → s(x+y);
x:Pos ⇒ s'(x)+y → s(x+y);
x:Zero ⇒ p'(x)+y → p(x+y);
x:Neg ⇒ p'(x)+y → p(x+y);
$0' * x \to 0$;
x:Zero ⇒ s'(x) ∗ y → (x ∗ y) + y;
x:Pos ⇒ s'(x) ∗ y → (x ∗ y) + y;
x:Zero ⇒ p'(x) ∗ y → (x ∗ y) + (-y);
x:Neg ⇒ p'(x) ∗ y → (x ∗ y) + (-y);
s(p(x)) → x;
p(s(x)) → x;
s(x)+y → s(x+y);
p(x)+y → p(x+y);
(x+y)+z → x+(y+z);

Fig. 10. A new specification of Integers

```
parameter specification: TOSET
sorts Elem, Bool;
functions:
True :                  → Bool;
False :                 → Bool;
_<_  : Elem Elem        → Bool;
constraints:
True = False ⇒;
x < x = False;
x < y = True ∨ x < y = False;
x < y = False ∨ y < x = False;
x < y = False ∨ y < z = False ∨ x < z = True;
end
```

Fig. 11. The parameter specification TOSET

```
specification: SORTING[T::TOSET]
sorts List;
constructors:
Nil      :              → List;
cons_    : Elem List    → List;
defined functions:
insert_  : Elem List    → List;
isort_   : List         → List;
sorted_  : List         → Bool;
axioms:
insert(x,Nil) → Cons(x,Nil);
y < x=False ⇒ insert(x,Cons(y,z)) → Cons(x,Cons(y,z));
y < x=True ⇒ insert(x,Cons(y,z)) → Cons(y,insert(x,z));
isort(Nil) → Nil ;
isort(Cons(x,l)) → insert(x,isort(l));
sorted(Nil) → True;
sorted(Cons(x,Nil)) → True;
y < x=True ⇒ sorted(Cons(x,Cons(y,z))) → False;
y < x=False ⇒ sorted(Cons(x,Cons(y,z))) → sorted(Cons(y,z));
sorted(insert(x,y)) → sorted(y);
sorted(isort(x)) → True;
end
```

Fig. 12. The parameterized specification SORTING

Example 8.2. Fig. 11 and Fig. 12 give an example of a parameterized specification. Note that the constraint 'True = False ⇒' is semantically equivalent to the inequality 'True ≠ False'. To prove the termination of $\mathcal{E}_{\mathcal{B}} - \mathcal{E}_{\mathcal{P}}$, we can use the *lexicographic path ordering* $\prec$ (see for instance [Dershowitz 1987]) with the following precedence on functions:

$$False \prec True \prec < \prec Nil \prec Cons \prec sorted \prec insert \prec isort$$

We can easily check that the specification NAT (resp. INTEGERS) given in Fig. 1 (resp. Fig. 2) is an instance of the parameter specification TOSET, since all the parameter constraints given in Fig. 11 are inductive theorems of the specification

NAT (resp. INTEGERS).

Definition 8.3. Let $\mathcal{PS} = (\mathcal{P}, \mathcal{B})$ be a *parameterized specification*. Assume that $\mathcal{F}_\mathcal{B} - \mathcal{F}_\mathcal{P}$ is partitioned into two disjoint sets $\mathcal{F}_\mathcal{D} \cup \mathcal{F}_\mathcal{C}$, with $\mathcal{F}_\mathcal{C}$ the set of constructor symbols, and $\mathcal{F}_\mathcal{D}$ the set of defined symbols. We say that $\mathcal{PS}$ is *complete* iff each defined symbol $f \in \mathcal{F}_\mathcal{B} - \mathcal{F}_\mathcal{C}$ is completely defined.

A function symbol $f \in \mathcal{F}_\mathcal{B} - \mathcal{F}_\mathcal{C}$ is *completely defined* iff for each model $\mathcal{A}$ of $\mathcal{E}_\mathcal{P}$ and each term T of the form $f(T_1, \ldots, T_n)$ where for all $1 \leq i \leq n$, $T_i \in \mathcal{T}(\mathcal{F}_\mathcal{C}(\mathcal{A}))$, there exists $T' \in \mathcal{T}(\mathcal{F}_\mathcal{C}(\mathcal{A}))$ such that $f(T_1, \ldots, T_n) \rightarrow^+_{(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}) \cup \mathcal{D}(\mathcal{A})} T'$. ◇

Definition 8.4. Let $\mathcal{PS} = (\mathcal{P}, \mathcal{B})$ be a *parameterized specification*. We say that $\mathcal{PS}$ is *ground confluent* iff for each model $\mathcal{A}$ of $\mathcal{E}_\mathcal{P}$ and for each ground terms $U,\ V,\ W \in \mathcal{T}(\mathcal{F}(\mathcal{A}))$:
if $V \ {}^*_{(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}) \cup \mathcal{D}(\mathcal{A})}\!\!\longleftarrow\ U \longrightarrow^*_{(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}) \cup \mathcal{D}(\mathcal{A})} W$, then $V \downarrow_{(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}) \cup \mathcal{D}(\mathcal{A})} W$. ◇

The inference system $\mathcal{CGC}'$ can be easily adapted, as expected, to allow us to check the completeness and the ground confluence of $(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}) \cup \mathcal{D}(\mathcal{A})$ for every model $\mathcal{A}$ of $\mathcal{E}_\mathcal{P}$.

First of all, we need sufficient conditions to guarantee the well-foundedness of $\rightarrow_{(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}) \cup \mathcal{D}(\mathcal{A})}$. Let us first introduce the following notions. To distinguish between the rewrite steps that result from the system $\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}$ and those resulting from $\mathcal{D}(\mathcal{A})$ we write $\xrightarrow{\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}}_{(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}) \cup \mathcal{D}(\mathcal{A})}$ (resp. $\xrightarrow{\mathcal{D}(\mathcal{A})}_{(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}) \cup \mathcal{D}(\mathcal{A})}$) to indicate that the rule inducing the rewrite step is an element from $\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}$ (resp. $\mathcal{D}(\mathcal{A})$).

The following lemma which can be easily proved is fundamental in order to prove that the rewrite relation $\rightarrow_{(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}) \cup \mathcal{D}(\mathcal{A})}$ is terminating for every model $\mathcal{A}$ of $\mathcal{E}_\mathcal{P}$.

LEMMA 8.5. *Assume that $\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}$ is left-linear* [5]*, reductive and no symbol from $\mathcal{F}_\mathcal{P}$ occurs on the left-hand side of any rewrite rule from $\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}$. Let $T \in \mathcal{T}(\mathcal{F}(A))$, if there exists T' and T'' such that*

$$T \xrightarrow{\mathcal{D}(\mathcal{A})}_{(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}) \cup \mathcal{D}(\mathcal{A})} T' \xrightarrow{\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}}_{(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}) \cup \mathcal{D}(\mathcal{A})} T''$$

then there exists T''' such that

$$T \xrightarrow{\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}}_{(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}) \cup \mathcal{D}(\mathcal{A})} T''' \xrightarrow{\mathcal{D}(\mathcal{A})}{}^*_{(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}) \cup \mathcal{D}(\mathcal{A})} T''$$

As a consequence of the interchangeability lemma we get the well-foundedness of $\rightarrow_{(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}) \cup \mathcal{D}(\mathcal{A})}$.

THEOREM 8.6. *If $\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}$ is left-linear, reductive and no symbol from $\mathcal{F}_\mathcal{P}$ occurs on the left-hand side of any rewrite rule from $\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}$, then the rewrite relation $\rightarrow_{(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}) \cup \mathcal{D}(\mathcal{A})}$ is terminating for every algebra $\mathcal{A}$ in $Alg_{\Omega_\mathcal{P}, \mathcal{E}_\mathcal{P}}$.*

We can now address the soundness of our inference system:

THEOREM 8.7 (SOUNDNESS). *Let $\mathcal{PS} = (\mathcal{P}, \mathcal{B})$ be a* parameterized specification*. Assume that $\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}$ is left-linear, reductive and no symbol from $\mathcal{F}_\mathcal{P}$ occurs on the left-hand side of any rewrite rule from $\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}$. If $\mathcal{CGC}'$ succeeds then $\mathcal{PS}$ is complete and ground confluent. If the rule* **Ground Confluence** *applies, then $\mathcal{PS}$ is ground confluent. If the rule* **Missing Patterns** *applies, then $\mathcal{PS}$ is not complete. If one*

[5] If $\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}$ is not left-linear, then we can consider the linearized version of $\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}$.

of the rules **Ambiguous Patterns**, **Non-Valid Rules** *or* **Non-Valid Critical Pairs** *applies, then* $\mathcal{PS}$ *is not ground confluent.*

PROOF. Let $\mathcal{A}$ be a model of $\mathcal{E}_\mathcal{P}$. Since no symbol from $\mathcal{F}_\mathcal{P}$ occurs on the left-hand side of any rule from $\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}$, then no critical pair between a rewrite rule from $\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}$ and a rewrite rule from $\mathcal{D}(\mathcal{A})$ can exist for any model $\mathcal{A}$ of $\mathcal{E}_\mathcal{P}$. Now, by Theorem 8.6, we deduce that $\rightarrow_{(\mathcal{E}_\mathcal{B}-\mathcal{E}_\mathcal{P})\cup\mathcal{D}(\mathcal{A})}$ is terminating and therefore by following the same reasoning used in the proof of Theorem 6.2, we conclude that $(\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P})\cup\mathcal{D}(\mathcal{A})$ is ground confluent and complete.
The remainder of the proof is similar to the proof of Theorem 6.2. □

The inference system $\mathcal{CGC}'$ is also complete for parameterized specifications.

THEOREM 8.8 (COMPLETENESS). *Let* $\mathcal{PS} = (\mathcal{P}, \mathcal{B})$ *be a* parameterized specification. *Assume that* $\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}$ *is left-linear, reductive and no symbol from* $\mathcal{F}_\mathcal{P}$ *occurs on the left-hand side of any rewrite rule from* $\mathcal{E}_\mathcal{B} - \mathcal{E}_\mathcal{P}$. *Assume further an oracle for deciding (joinable) inductive conjectures. If* $\mathcal{PS}$ *is complete and ground confluent, then* $\mathcal{CGC}'$ *will succeed.*

PROOF. The proof is similar to the proof of Theorem 6.4. □

```
Pattern tree of +:

 x1 +  x2
   0 + x2
     0 + 0 -Ok-
     0 + s(x1) -Ok-
   s(x3) + x2
     s(x3) + 0 -Ok-
     s(x3) + s(x1) -Ok-

--Press Enter to go on.--

Pattern tree of *:

 x1 *  x2
   0 * x2
     0 * 0 -Ok-
     0 * s(x1) -Ok-
   s(x3) * x2
     s(x3) * 0 -Ok-
     s(x3) * s(x1) -Ok-

--Press Enter to go on.--
```

Fig. 13. Proving the Completeness and the Ground Confluence of Nat

```
Pattern tree of <:
 x1 < x2
   0 < x2
     0 < 0 -Ok-
     0 < s(x1) -Ok-
   s(x3) < x2
     s(x3) < 0 -Ok-
     s(x3) < s(x1) -Ok-

--Press Enter to go on.--

Pattern tree of even:
 even(x1)
   even(0) -Ok-
   even(s(x2))
     even(s(0)) -Ok-
     even(s(s(x1))) -Ok-

--Press Enter to go on.--

Pattern tree of odd:
 odd(x1)
   odd(0) -Ok-
   odd(s(0)) -Ok-
   odd(s(s(x1))) -Ok-
      False if even(s(s(x1)))=True
      True if even(s(s(s(x1))))=True

--Press Enter to go on.--

(1) All the leaves of the trees are strongly reducible and non-ambiguous.

(2) The following rules are proved
       to be inductively valid w.r.t the remainder of the axioms:

     (x1 + x2) + x3 = x1 + (x2 + x3) ;
     even(x1 + x1) = True ;
     even(s(x1 + x1)) = False ;
     even(x1 + (x2 + x2)) = even(x1) ;
     odd(s(s(x1))) = odd(x1) ;
     odd(x1 + x1) = False ;
     odd(s(x1 + x1)) = True ;
     odd(x1) = True, odd(x2) = True => odd(x1 + x2) = False ;
     even(x1) = True, even(x2) = True => even(x1 + x2) = True ;
     odd(x1) = True, odd(x2) = False => odd(x1 + x2) = True ;
     even(x1) = True, even(x2) = False => even(x1 + x2) = False ;
     even(x1 * s(s(x2))) = even(x1 * x2) ;
     even(x1) = True => even(x1 * x2) = True ;
     odd(x1) = True, odd(x2) = True => even(x1 * x2) = False

From (1) and (2) it follows that the axioms are complete and ground confluent.
```

Fig. 14. Proving the Completeness and the Ground Confluence of Nat (Cont.)

$$even(x_1 + (x_2 + (x_1 + x_2))) = True$$
$$even(x_1 + (x_2 + (x_3 + (x_1 + (x_2 + x_3))))) = True$$
$$even(x_1 + (x_2 + (x_3 + (x_4 + (x_2 + (x_3 + (x_4 + x_1))))))) = True$$
$$\vdots$$
$$even(x_1 + (x_2 + (x_3 + (x_2 + x_3)))) = even(x_1)$$
$$even(x_1 + (x_2 + (x_3 + (x_4 + (x_2 + (x_3 + x_4)))))) = even(x_1)$$
$$even(x_1 + (x_2 + (x_3 + (x_4 + (x_5 + (x_2 + (x_3 + (x_4 + x_5)))))))) = even(x_1)$$
$$\vdots$$
$$even(s(x_1)) = True \Rightarrow even(s((x_1 * s(s(x_2))) + x_2)) = False$$
$$even(s(x_1)) = True \Rightarrow even(x_1 * s(x_2 + (x_3 + (x_2 + x_3)))) = False$$
$$even(s(x_1)) = True \Rightarrow even((x_1 * s(x_2 + (x_3 + x_3))) + x_2) = False$$
$$\vdots$$
$$even(s(x_1)) = True \Rightarrow odd(x_1 + (x_2 + (x_3 + (x_2 + x_3)))) = True$$
$$even(s(x_1)) = True \Rightarrow odd(x_1 + (x_2 + (x_3 + (x_4 + (x_3 + (x_4 + x_2)))))) = True$$
$$even(s(x_1)) = True \Rightarrow odd(x_1 + (x_2 + (x_3 + (x_4 + (x_5 + (x_2 + (x_3 + (x_4 + x_5)))))))) = True$$
$$\vdots$$

Fig. 15. Divergence of the Completion procedure

9. IMPLEMENTATION AND COMPUTER EXPERIMENTS

We have implemented our new technique in the SPIKE system. The program is able to check both completeness and ground confluence for parameterized and non-parameterized conditional specifications with free constructors.

The program starts by computing a pattern tree for every defined symbol, and identifies a set of rules that we must check for validity. The leaves of the tree give a partition of the possible arguments for defined functions. If all the leaves are strongly reducible and non-ambiguous, and if the identified rules are inductively valid, then we conclude that the given specification is complete and ground confluent.

The root of a pattern tree is displayed first, and each level of the tree is indented to ease the reading. There are two kinds of leaves: leaves which are strongly reducible by case (i), and leaves which are strongly reducible by case (ii) (see Definition 5.2). For the last case, we display for each leaf l: $l[r_1\sigma_1]_{u_1}$ if $p_1\sigma_1, \ldots, l[r_n\sigma_n]_{u_n}$ if $p_n\sigma_n$ where $\forall i \in [1..n] : p_i \Rightarrow l_i \rightarrow r_i \in \mathcal{R}$ and $l|_{u_i} = l_i\sigma_i$. With each leaf comes a comment indicating whether it is strongly reducible and non-ambiguous.

Example 9.1. Fig. 13 and Fig. 14 show the transcript of a session with SPIKE for proving the completeness[6] and the ground confluence of the specification given in Fig. 1.

Using completion techniques, we obtain an infinite set of critical pairs (see Fig. 15).

This example cannot be checked by the methods of [Göbel 1987; Becker 1993], since they are designed for non-conditional specifications.

To prove the ground confluence of $\mathcal{R}$ using the methods of [Kounalis and Rusinowitch 1991; Becker 1996], we need to compute more than 120 critical pairs ! In

[6]Note that $odd(s(s(x)))$ is also strongly reducible by the rule (20).

```
Pattern tree of insert:
 insert(x1,x2)
   insert(x1,Nil) -Ok-
   insert(x1,Cons(x3,x4)) -Ok-
     Cons(x1,Cons(x3,x4)) if x3 < x1=False
     Cons(x3,insert(x1,x4)) if x3 < x1=True

--Press Enter to go on.--

Pattern tree of isort:
 isort(x1)
   isort(Nil) -Ok-
   isort(Cons(x2,x3)) -Ok-

--Press Enter to go on.--

Pattern tree of sorted:
 sorted(x1)
   sorted(Nil) -Ok-
   sorted(Cons(x2,x3))
     sorted(Cons(x2,Nil)) -Ok-
     sorted(Cons(x2,Cons(x1,x4))) -Ok-
       False if x1 < x2=True
       sorted(Cons(x1,x4)) if x1 < x2=False

--Press Enter to go on.--

(1) All the leaves of the trees are strongly reducible and non-ambiguous.

(2) The following rules are proved
       to be inductively valid w.r.t the remainder of the axioms:

     sorted(insert(x1,x2)) = sorted(x2) ;
     sorted(isort(x1)) = True

From (1) and (2) it follows that the axioms are complete and ground confluent.
```

Fig. 16. Proving the Completeness and the Ground Confluence of SORTING

addition, the validity test of most critical pairs fails or it is very hard to achieve.

For example, the critical pair [7] between rules (7) and (27):

$$(30)\ even(s(x_1) * x_2) = even((x_1 * s^2(x_2)) + s^2(x_2))$$

is not valid w.r.t. the sufficient criterion [8] for ground confluence given in [Kounalis and Rusinowitch 1991]:

Indeed, let us consider the instance of equation (30) by the test set substitution

[7] In order to simplify notations we write $s^n(x)$ instead of $\underbrace{s(\ldots s}_{n \text{ times}}(x)\ldots)$

[8] Let C be a critical pair between two rules in $\mathcal{R}$, then each instance of C by a test substitution must be joinable by $\mathcal{R} \cup \{C\}$.

```
Pattern tree of insert:
 insert(x1,x2)
   insert(x1,Nil) -strongly irreducible and non-ambiguous-
   insert(x1,Cons(x3,x4)) -strongly irreducible and non-ambiguous-
     Cons(x3,insert(x1,x4)) if x3 < x1=True

--Press Enter to go on.--


If you add the following right-hand sides, I will be able to prove completeness:
insert(x1,Nil) -> ...
x3 < x1 = False => insert(x1,Cons(x3,x4)) -> ...

Do you want to add new rules <y/n> ? : y

A new rule: insert(x1,Nil) -> Cons(x1,Nil)
Other rule <y/n> ? : y

A new rule: x3 < x1 = False => insert(x1,Cons(x3,x4)) -> Cons(x1,Cons(x3,x4))
Other rule <y/n> ? : n

Pattern tree of insert:
 insert(x1,x2)
   insert(x1,Nil) -Ok-
   insert(x1,Cons(x3,x4)) -Ok-
     Cons(x3,insert(x1,x4)) if x3 < x1=True
     Cons(x1,Cons(x3,x4)) if x3 < x1=False

--Press Enter to go on.--

...

(1) All the leaves of the trees are strongly reducible and non-ambiguous.

(2) The following rules are proved
       to be inductively valid w.r.t the remainder of the axioms:

     sorted(insert(x1,x2)) = sorted(x2) ;
     sorted(isort(x1)) = True ;

From (1) and (2) it follows that the axioms are complete and ground confluent.
```

Fig. 17. Interaction with the user in order to recover a Complete Specification

$\{x_1 \mapsto s^3(x'_1),\ x_2 \mapsto s^3(x'_2)\}$:

$$(31)\ even(s^4(x'_1) * s^3(x'_2)) = even((s^3(x'_1) * s^5(x'_2)) + s^5(x'_2))$$

The term $even(s^4(x'_1) * s^3(x'_2))$ is simplified by the axiom (7) into $even((s^3(x'_1) * s^3(x'_2)) + s^3(x'_2))$. We obtain:

$$(32)\ even((s^3(x'_1) * s^3(x'_2)) + s^3(x'_2)) = even((s^3(x'_1) * s^5(x'_2)) + s^5(x'_2))$$

(32) is not joinable by $\mathcal{R} \cup \{(30)\}$ and therefore we fail to check the validity of the

```
Pattern tree of insert:
 insert(x1,x2)
   insert(x1,Nil) -Ok-
   insert(x1,Cons(x3,x4)) -strongly reducible and ambiguous-
     Cons(x1,Cons(x3,x4)) if x3 < x1=False
     Cons(x3,Cons(x1,x4)) if x3 < x1=False
     Cons(x3,insert(x1,x4)) if x3 < x1=True

--Press Enter to go on.--

Pattern tree of isort:
 isort(x1)
   isort(Nil) -Ok-
   isort(Cons(x2,x3)) -Ok-

--Press Enter to go on.--

Pattern tree of sorted:
 sorted(x1)
   sorted(Nil) -Ok-
   sorted(Cons(x2,x3))
     sorted(Cons(x2,Nil)) -Ok-
     sorted(Cons(x2,Cons(x1,x4))) -strongly reducible and ambiguous-
       False if x1 < x2=True
       sorted(Cons(x2,x4)) if x1 < x2=True
       sorted(Cons(x1,x4)) if x1 < x2=False

--Press Enter to go on.--

(1) All the leaves of the trees are strongly reducible.

(2) The following patterns are ambiguous:

       insert(x1,Cons(x3,x4)) ;
       sorted(Cons(x2,Cons(x1,x4)))

Please correct the specification to make this patterns non-ambiguous.

From (2) it follows that the axioms are not ground confluent.
```

Fig. 18. Detection of Ambiguous Patterns

critical pair (30) with the method of [Kounalis and Rusinowitch 1991].

Example 9.2. Fig. 16 shows the transcript of a session with SPIKE for proving the completeness and the ground confluence of the parameterized specification given in Fig. 11 and Fig. 12. Note that this example cannot be checked by the methods of [Göbel 1987; Becker 1993; Kounalis and Rusinowitch 1991; Becker 1996], since they are designed for non-parameterized specifications.

If the specification is not complete, then our program will output the set of patterns on whose ground instances a function $f \in \mathcal{D}$ is not defined.

Example 9.3. Let us remove the two first axioms from Fig. 12. Then, *insert* is not completely defined. In Fig. 17, we describe a session with SPIKE to give an idea about the interaction with the user in order to recover a complete specification.

With SPIKE, it is easy to detect ambiguous patterns [9].

Example 9.4. Let us add the following rules:

```
y < x = False => insert(x,Cons(y,z)) = Cons(y,Cons(x,z))
y < x = True  => sorted(Cons(x,Cons(y,z))) = sorted(Cons(x,z))
```

to the specification given in Fig. 12. Then the patterns `insert(x1,Cons(x3,x4))` and `sorted(Cons(x2,Cons(x1,x4)))` are ambiguous. Therefore, the axioms are not ground confluent (see Fig. 18).

With our system, it is also easy to detect the rules that break ground confluence.

```
...

(1) All the leaves of the trees are strongly reducible and non-ambiguous.

(2) At least one of the following rules is proved
      to be not inductively valid w.r.t the remainder of the axioms:

     sorted(x1) = False => sorted(insert(x2,x1)) = True ;
     sorted(insert(x1,x2)) = sorted(x2) ;
     sorted(isort(x1)) = True

From (1) and (2) it follows that the axioms are not ground confluent.
```

Fig. 19. Detection of the rules that break Ground Confluence

Example 9.5. Let us add the rule:

```
(r) sorted(y) = False => sorted(insert(x,y)) = True
```

to the specification given in Fig. 12. Then, all the leaves of the trees are strongly reducible and non-ambiguous and the rule `(r)` is not an inductive conjecture [10] w.r.t. the remainder of the axioms. Therefore the axioms are not ground confluent (see Fig. 19).

[9] Our inference system for checking joinable-inductive conjectures allows to refute false conjectures, even if the axioms are not complete and not ground confluent (see Theorem 2 in [Bouhoula and Jacquemard 2007]).

[10] All the leaves of the trees are non-ambiguous. It follows from Theorem 6.2 that the axioms whose left-hand sides match the leaves of the trees are ground confluent. This property is necessary for the refutation of false inductive conjectures (see [Bouhoula and Rusinowitch 1995]).

10. CONCLUSION

We have presented a new procedure for simultaneously checking completeness and ground confluence for specifications with free/non-free constructors and parameterized specifications. As opposed to previous works for checking ground confluence, our procedure does not rely on completion techniques and does not require the computation of critical pairs of the axioms. Our technique has been implemented in the prover SPIKE. Our system offers two main components: (i) a completeness and ground confluence analyser that compute pattern trees of defined functions and may generate some proof obligations; and (ii) a procedure to prove (joinable) inductive conjectures which is used to discharge those proof obligations. Computer experiments show that our method is more practical than related approaches. Indeed, as shown in Example 9.1, our procedure allows us to check the ground confluence of specifications where the classical completion techniques generate an infinite set of critical pairs. Moreover, our proof of the ground confluence of the specification given in Fig. 1 is completely automatic and does not require the computation of critical pairs. However, the methods of [Göbel 1987; Kounalis and Rusinowitch 1991; Becker 1993; 1996] need to compute more than 120 critical pairs ! In addition, the validity test of most critical pairs fail or it is very hard to achieve.

We plan to extend our technique to membership equational theories [Bouhoula et al. 2000] as well as Associative/Commutative (AC) theories. The extension of related approaches for AC theories can be very inefficient since they are based on the computation of critical pairs, and therefore they need to compute the minimal complete set of AC-unifiers that has a double-exponential cardinality in the worst case [Kapur and Narendran 1992]. Unlike these approaches, our method should be also efficient for AC theories since it does not use unification, but matching only.

ACKNOWLEDGMENTS

I wish to thank the anonymous referees for their pertinent remarks and suggestions.

REFERENCES

Becker, K. 1993. Proving ground confluence and inductive validity in constructor based equational specifications. In *Theory and Practice of Software Development: 4th International Joint Conference Caap/Fase*, M.-C. Gaudel and J.-P. Jouannaud, Eds. Lecture Notes in Computer Science, vol. 668. Springer-Verlag, Orsay, France, 46–60.

Becker, K. 1996. How to Prove Ground Confluence. SEKI-report SR-96-02, Universität Kaiserslautern.

Bouhoula, A. 1996. Using induction and rewriting to verify and complete parameterized specifications. *Theoretical Computer Science 170,* 1-2, 245–276.

Bouhoula, A. 1997. Automated theorem proving by test set induction. *Journal of Symbolic Computation 23,* 1, 47–77.

Bouhoula, A. 2000. Simultaneous checking of completeness and ground confluence. In *the Fifteenth IEEE International Conference on Automated Software Engineering.* IEEE Computer Society Press, Grenoble, France, 143–151.

Bouhoula, A. and Jacquemard, F. 2007. Verifying regular trace properties of security protocols with explicit destructors and implicit induction. In *the Joint Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis (FCS-ARSPA-07).* Wroclaw, Poland, 27–44.

Bouhoula, A. and Jouannaud, J.-P. 2001. Automata-driven automated induction. *Information and Computation 169,* 1, 1–22.

Bouhoula, A., Jouannaud, J.-P., and Meseguer, J. 2000. Specification and proof in membership equational logic. *Theoretical Computer Science 170,* 1-2, 35–132.

Bouhoula, A., Kounalis, E., and Rusinowitch, M. 1995. Automated Mathematical Induction. *Journal of Logic and Computation 5,* 5, 631–668.

Bouhoula, A. and Rusinowitch, M. 1995. Implicit induction in conditional theories. *Journal of Automated Reasoning 14,* 2, 189–235.

Comon, H. 1986. Sufficient completeness, term rewriting system and anti-unification. In *Proceedings 8th International Conference on Automated Deduction*, J. Siekmann, Ed. Lecture Notes in Computer Science, vol. 230. Springer-Verlag, Oxford, UK, 128–140.

Comon, H. 1989. Inductive proofs by specifications transformations. In *Proceedings 3rd Conference on Rewriting Techniques and Applications*, N. Dershowitz, Ed. Lecture Notes in Computer Science, vol. 355. Springer-Verlag, Chapel Hill (N.C., USA), 76–91.

Dershowitz, N. 1983. Well-founded orderings. Tech. Rep. ATR-83-8478-3, Office of Information Sciences Research, The Aerospace Corporation, El Segundo, California USA. May.

Dershowitz, N. 1987. Corrigendum to termination of rewriting. *Journal of Symbolic Computation 4,* 3, 409–410.

Dershowitz, N. and Jouannaud, J.-P. 1990. *Handbook of Theoretical Computer Science*. Vol. B. Elsevier Science Publishers B. V., North-Holland, Chapter 6: Rewrite Systems, 244–320.

Dershowitz, N. and Okada, M. 1990. A rationale for conditional equational programming. *Theoretical Computer Science 75*, 111–138.

Dershowitz, N., Okada, M., and Sivakumar, G. 1987. Confluence of conditional rewrite systems. In *Proceedings 1st International Workshop on Conditional Term Rewriting Systems*, J.-P. Jouannaud and S. Kaplan, Eds. Lecture Notes in Computer Science, vol. 308. Springer-Verlag, Orsay, France, 31–44.

Ehrig, H. and Mahr, B. 1985. *Fundamentals of Algebraic Specification 1. Equations and initial semantics*. EATCS Monographs on Theoretical Computer Science, vol. 6. Springer-Verlag.

Fribourg, L. 1989. A strong restriction of the inductive completion procedure. *Journal of Symbolic Computation 8,* 3 (Sept.), 253–276.

Futatsugi, K., Goguen, J. A., Jouannaud, J.-P., and Meseguer, J. 1985. Principles of OBJ-2. In *Proceedings 12th ACM Symp. on Principles of Programming Languages*, B. Reid, Ed. ACM, 52–66.

Ganzinger, H. 1987. Ground term confluence in parametric conditional equational specifications. In *Proceedings STACS 87*, F. J. Brandenburg, G. Vidal-Naquet, and M. Wirsing, Eds. Lecture Notes in Computer Science, vol. 247. Springer-Verlag, 286–298.

Göbel, R. 1987. Ground confluence. In *Proceedings 2nd Conference on Rewriting Techniques and Applications*, P. Lescanne, Ed. Lecture Notes in Computer Science, vol. 256. Springer-Verlag, Bordeaux, France, 156–167.

Goguen, J. A. and Meseguer, J. 1988. Order-sorted algebra I: Partial and overloaded operations, errors and inheritance. Tech. rep., SRI International, Computer Science Lab. Given as lecture at a Seminar on Types, Carnegie-Mellon University, June 1983.

Gramlich, B. and Wirth, C.-P. 1996. Confluence of terminating conditional rewrite systems revisited. In *Proceedings 7th Conference on Rewriting Techniques and Applications*, H. Ganzinger, Ed. Lecture Notes in Computer Science, vol. 1103. Springer-Verlag, New Brunswick (New Jersey, USA), 245–259.

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

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

Jouannaud, J.-P. and Kounalis, E. 1989. Automatic proofs by induction in theories without constructors. *Information and Computation 82*, 1–33.

Kaplan, S. 1987. Simplifying conditional term rewriting systems: Unification, termination and confluence. *Journal of Symbolic Computation 4,* 3 (Dec.), 295–334.

Kapur, D. 1994. An automated tool for analyzing completeness of equational specifications. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA), volume Special Issue of* Software Engineering Notes, J. Siekmann, Ed. ACM Press, 28–43.

Kapur, D., Narendan, P., and Otto, F. 1990. On ground confluence of term rewriting systems. *Information and Computation 86,* 1, 14–31.

Kapur, D. and Narendran, P. 1992. Double-exponential complexity of computing a complete set of AC-unifiers. In *Proceedings 9th IEEE Symposium on Logic in Computer Science*. IEEE, Santa-Cruz (California, USA).

Kapur, D., Narendran, P., Rosenkrantz, D.-J., and Zhang, H. 1991. Sufficient-completeness, ground-reducibility and their complexity. *Acta Informatica 28,* 4, 311–350.

Kounalis, E. 1985. Completeness in data type specifications. In *Proceedings EUROCAL Conference*, B. Buchberger, Ed. Lecture Notes in Computer Science, vol. 204. Springer-Verlag, Linz, Austria, 348–362.

Kounalis, E. and Rusinowitch, M. 1991. Studies on the ground convergence property of conditional theories. In *Proceedings of AMAST*. Workshop in Computing. Springer-Verlag, Iowa City.

Küchlin, W. 1985. A confluence criterion based on the generalised Knuth-Bendix algorithm. In *Proceedings of the EUROCAL Conference*, B. Buchberger, Ed. Lecture Notes in Computer Science, vol. 204. Springer-Verlag, Linz, Austria, 390–399.

Lazrek, A., Lescanne, P., and Thiel, J.-J. 1990. Tools for proving inductive equalities, relative completeness and ω-completeness. *Information and Computation 84,* 1 (Jan.), 47–70.

Padawitz, P. 1988. The equational theory of parameterized specifications. *Information and Computation 76*, 121–137.

Plaisted, D. A. 1985. Semantic confluence tests and completion methods. *Information and Control 65,* 2/3 (May), 182–215.

Schmidt-Schauss, M. 1989. *Computational Aspects of an Order-Sorted Logic with Term Declarations*. Lecture Notes in Computer Science, vol. 395. Springer-Verlag.

Smolka, G., Nutt, W., Goguen, J. A., and Meseguer, J. 1987. Order sorted equational computation. In *Proceedings of the Colloquium on Resolution of Equations in Algebraic Structures*. Austin (Texas).

Thiel, J.-J. 1984. Stop losing sleep over incomplete data type specifications. In *Proceeding 11th ACM Symp. on Principles of Programming Languages*. ACM, Salt Lake City, Utah, 76–82.

Wirsing, M. 1990. Algebraic specification. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, J. van Leeuwen, A. Meyer, M. Nivat, M. Paterson, and D. Perrin, Eds. Elsevier and MIT Press, Chapter 13, 675–788.

APPENDIX

Proof of Theorem 6.2

(i) Let us prove that if $\mathcal{CGC}$ succeeds then $\mathcal{R}$ is complete and ground confluent. To show that $\mathcal{R}$ is ground confluent, it is sufficient to show that $\mathcal{UR}$ is ground confluent since the rules in $\mathcal{R} - \mathcal{UR}$ are inductive theorems of $\mathcal{UR}$. Since $\mathcal{AP}$ is empty (i.e., all the leaves of the pattern trees are non-ambiguous and do not contain any induction variable) and all the rules in $\mathcal{UR}$ are of the form $P \Rightarrow f(\overline{T}) \rightarrow R$ where $f \in \mathcal{D}$ and $T_i \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ for each $T_i \in \overline{T}$, then all critical pairs between rules in $\mathcal{UR}$ are trivial or joinable-inductive theorems. Therefore, we conclude that $\mathcal{UR}$ is ground confluent.
Now, let us show that $\mathcal{R}$ is complete. Let $T \in \mathcal{T}(\mathcal{F})$ and T' be the normal form of T with respect to $\mathcal{R}$. If T' is a constructor term, we are done.
Otherwise, T' must contain a subterm T'' of the form $g(\overline{T})$ where $g \in \mathcal{D}$ and for all $i \in [1..n]$, $T_i \in \mathcal{T}(\mathcal{C})$. Since the leaves of the pattern trees exhaust all cases by construction, this subterm must be an instance of a leaf S: $S\sigma = T''$, where σ is a ground substitution over $\mathcal{T}(\mathcal{C})$.
Since S must be strongly reducible, there exists a non-empty sequence of conditional rules in $\mathcal{R}$: $P_1 \Rightarrow L_1 \rightarrow R_1$, ..., $P_n \Rightarrow L_n \rightarrow R_n$ and a sequence of positions $u_1, u_2, \ldots, u_n$ in S such that $S|_{u_1} = T_1\sigma_1$, $S|_{u_2} = T_2\sigma_2$, $\ldots$, $S|_{u_n} = T_n\sigma_n$ and $P_1\sigma_1 \vee P_2\sigma_2 \vee \ldots \vee P_n\sigma_n$ is an inductive theorem of $\mathcal{R}$.
Then, there exists k such that $\mathcal{R} \models P_k\sigma_k$. Since $\mathcal{R}$ is ground confluent and the rewrite relation $\rightarrow_{\mathcal{R}}$ is terminating, all equalities in $P_k\sigma_k$ can be proved by normalization, and therefore the rule $P_k \Rightarrow L_k \rightarrow R_k$ can be applied to simplify T''. This contradicts the fact that T' is in normal form.

(ii) If the rule **Ground Confluence** applies, then by following the same reasoning as in (i), we conclude that $\mathcal{R}$ is ground confluent.

(iii) If the rule **Missing Patterns** applies, then there exists a pattern T of the form $(f(\overline{T}), \Gamma)$ where $f \in \mathcal{D}$ and $T_i \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ for every $T_i \in \overline{T}$, such that T is not strongly reducible and $\mathcal{IndVar}(T, \mathcal{R}) = \emptyset$.
Assume that $f(\overline{T})$ is not matched by any left-hand side of an axiom in $\mathcal{R}$. Let $f(\overline{T})\tau$ be a ground instance of $f(\overline{T})$ such that τ is built upon constructor symbols. Since $\mathcal{IndVar}(T, \mathcal{R}) = \emptyset$, then $f(\overline{T})\tau$ is not matched by any left-hand side of an axiom in $\mathcal{R}$. So $f(\overline{T})\tau$ is irreducible at the root. On the other hand, $f(\overline{T})\tau$ cannot be reduced to a non-root position, since the constructors are free. Therefore we conclude that $\mathcal{R}$ is not complete.
Otherwise, $f(\overline{T})$ is matched by the left-hand sides of n linearized rules $\mathcal{LR} = \{P_i \Rightarrow L_i \rightarrow R_i\}_{i \in [1..n]}$ but the formula $P_1\sigma_1 \vee \ldots \vee P_n\sigma_n$ is not an inductive theorem of $\mathcal{R}$. Then, there exists a ground and irreducible substitution τ such that $R \not\models P_1\sigma_1\tau \vee \ldots \vee P_n\sigma_n\tau$. Let us show that $f(\overline{T})\tau$ is irreducible by $\mathcal{R}$. The term $f(\overline{T})\tau$ is irreducible at the root since $R \not\models P_1\sigma_1\tau \vee \ldots \vee P_n\sigma_n\tau$. Assume otherwise, that there exists a linearized rule in $\mathcal{R} - \mathcal{LR}$ with left-hand side L that applies to $f(\overline{T})\tau$ and $f(\overline{T})\tau = L\lambda$ for a substitution λ. Note that every non variable position of L is a non variable position of $f(\overline{T})$ since $f(\overline{T})$ does not contain any induction variable. In particular $f(\overline{T})$ is not an instance of

L. Since L is linear we can define a substitution by $\rho(x) = f(\overline{T})|_w$ for every variable x that occurs at some position w of L. We have then $f(\overline{T}) = L\rho$, in contradiction with the assumption that $\mathcal{LR}$ contains all the rules whose left-hand sides match $f(\overline{T})$. On the other hand, $f(\overline{T})\tau$ cannot be reduced to a nonroot position, since the constructors are free. Therefore we conclude that $\mathcal{R}$ is not complete.

(iv) Assume that the rule **Ambiguous Patterns** applies, then there exists a pattern T of the form $(f(\overline{T}), \Gamma)$ where $f \in \mathcal{D}$ and $T_i \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ for every $T_i \in \overline{T}$, such that T is ambiguous and $\mathcal{I}nd\mathcal{V}ar(T, \mathcal{R}) = \emptyset$. Then there exist two rules $\overline{U} = \overline{V} \Rightarrow L \rightarrow R$ and $\overline{U'} = \overline{V'} \Rightarrow L' \rightarrow R'$ in $\mathcal{R}$ such that $f(\overline{T}) = L\sigma = L'\sigma'$, and the formula $\overline{U}\sigma = \overline{V}\sigma \wedge \overline{U'}\sigma' = \overline{V'}\sigma' \Rightarrow R\sigma = R'\sigma'$ is not a joinable-inductive theorem of $\mathcal{R}$. Hence, there exists a ground substitution τ such that: $\overline{U}\sigma\tau \downarrow_{\mathcal{R}} \overline{V}\sigma\tau$, $\overline{U'}\sigma'\tau \downarrow_{\mathcal{R}} \overline{V'}\sigma'\tau$ and $R\sigma\tau \not\downarrow_{\mathcal{R}} R'\sigma'\tau$. Then, we have $R'\sigma'\tau \ {}^{*}_{\mathcal{R}}\!\longleftarrow f(\overline{T})\tau \longrightarrow^{*}_{\mathcal{R}} R\sigma\tau$ but $R\sigma\tau \not\downarrow_{\mathcal{R}} R'\sigma'\tau$. We conclude that $\mathcal{R}$ is not ground confluent.

(v) Assume that the rule **Non-Valid Rules** applies, then $\mathcal{MP}$ and $\mathcal{AP}$ are empty, therefore $\mathcal{UR}$ is complete and ground confluent as shown in (i). On the other hand, there exists a rule $\overline{U} = \overline{V} \Rightarrow L \rightarrow R$ in $\mathcal{R} - \mathcal{UR}$ which is not an inductive theorem of $\mathcal{UR}$. Hence, there exists a ground substitution τ such that: $\mathcal{UR} \models \overline{U}\tau = \overline{V}\tau$ and $L\tau \not\downarrow_{\mathcal{UR}} R\tau$. Since $\mathcal{UR}$ is ground confluent and the rewrite relation $\longrightarrow_{\mathcal{UR}}$ is terminating, all equalities in $\overline{U}\tau = \overline{V}\tau$ can be proved by normalization, and therefore the rule $\overline{U} = \overline{V} \Rightarrow L \rightarrow R$ can be applied to simplify $L\tau$ into $R\tau$. Now, let L' (resp., R') be the normal form of $L\tau$ (resp., $R\tau$) via $\mathcal{UR}$. Then, we have $L' \ {}^{*}_{\mathcal{UR}}\!\longleftarrow L\tau \longrightarrow_{\mathcal{UR}\cup\{\overline{U}=\overline{V}\Rightarrow L\rightarrow R\}} R\tau \longrightarrow^{*}_{\mathcal{UR}} R'$. Since $\mathcal{UR}$ is complete, L' and R' are two constructor terms. On the other hand, the constructors are free in $\mathcal{R}$ and $L\tau \not\downarrow_{\mathcal{UR}} R\tau$. Hence, $L' \not\downarrow_{\mathcal{R}} R'$ and therefore, $\mathcal{R}$ is not ground confluent.

Received July 2006; revised December 2007; accepted April 2008