Parameterized conditional specifications : sufficient completeness and implicit induction

Adel Bouhoula

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Parameterized Specifications : Sufficient Completeness and Implicit Induction

Adel BOUHOULA

N° 2129
Décembre 1993

PROGRAMME 2

Calcul symbolique,
programmation
et génie logiciel

Parameterized Specifications: Sufficient Completeness and Implicit Induction *

ADEL BOUHOULA

CRIN & INRIA-Lorraine
BP 239, 54506 Vandoeuvre-lès-Nancy, France
email: bouhoula@loria.fr

January 18, 1994

Abstract

Theorem proving in parameterized specifications allows for shorter and more structured proofs. Moreover, a generic proof can be given just once and reused for each instantiation of the parameters. We present procedures to test sufficient completeness and to prove and disprove inductive properties automatically in parameterized conditional specifications. Our method relies on the notion of *test set*, which can be seen as a well-suited induction scheme. Previously, we could only compute a test set for conditional specifications if the constructors were free. Here, we give a new definition of test sets and an algorithm to compute them even if the constructors are not free. The method uses a new notion of provable inconsistency which allows us to refute more false conjectures than with previous approaches. This new method when limited to non-parameterized conditional specifications, can refute general clauses; refutational completeness is also preserved for boolean ground convergent rewrite systems even if the functions are not sufficiently complete and the constructors are not free. The method has been implemented in the prover SPIKE. Based on computer experiments, the method appears to be more practical and efficient than inductive theorem proving in non-parameterized specifications.

Keywords: Parameterized Conditional Specifications, Sufficient completeness, Theorem Proving, Implicit induction, Term rewriting systems.

*Partly supported by the PRC mécanisation du raisonnement and the Esprit BRA workshop COMPASS.

Spécifications paramétrées: Complétude suffisante et Induction implicite

ADEL BOUHOULA

CRIN & INRIA-Lorraine
BP 239, 54506 Vandoeuvre-lès-Nancy, France
email: bouhoula@loria.fr

Résumé

Les preuves dans les spécifications conditionnelles paramétrées ont l'avantage d'être courtes et structurées. De plus, une preuve générique peut être donnée une seule fois et est valide pour toutes les instanciations possibles du paramètre. Nous proposons deux procédures de test de complétude suffisante et de preuve par induction dans les spécifications conditionnelles paramétrées. Notre méthode se base sur la notion d'*ensemble test*. Dans les travaux précédents, nous ne pouvions calculer un *ensemble test* pour une spécification conditionnelle que dans le cas où les constructeurs sont libres. Ici, nous présentons une nouvelle définition des *ensembles test* et un algorithme permettant de les calculer même dans le cas où il y a des relations entre les constructeurs. La méthode utilise une nouvelle notion de *preuve d'inconsistance* qui permet de réfuter plus de conjectures non valides qu'avec les méthodes précédentes. La nouvelle procédure limitée aux spécifications non paramétrées, peut réfuter des clauses générales; la complétude réfutationnelle est aussi préservée pour les spécifications booléennes convergentes même si les fonctions ne sont pas suffisamment complètes et s'il y a des relations entre les constructeurs. Nous avons entièrement implanté cette nouvelle méthode dans le prouveur SPIKE. Sur la base de résultats expérimentaux, la méthode s'est avérée plus pratique et efficace que les prouveurs de théorèmes par induction dans les spécifications non paramétrées.

Mots clés: Spécifications Conditionnelles paramétrées, Complétude Suffisante, Preuve automatique, Induction implicite, Système de réécriture.

1 Introduction

Algebraic specifications provide a powerful method for the specification of abstract data types in programming languages and software systems. Often, algebraic specifications are built with equational or 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. One of the most important issues within the theory of algebraic specifications is the specification of parameterized data types. Most common data types like *list* are in fact parameterized types, *list(data)*. The key idea is to consider the parameter part *data* as a formal algebraic specification which can be actualized (i.e. instantiated) by other predefined algebraic specifications like *nat, int* or *bool*. Hence, we can obtain from the parameterized specification *list(data)* the three value specifications corresponding to lists of natural numbers, lists of integers and lists of boolean values. The benefit of this process is not only an economy of presentation but also the automatic correctness of all the value specifications provided that the parameterized specification *list(data)* is correct and the actual instantiation is valid. This is a very important property for building up larger data types and software systems from small pieces in a correct way. Sufficient completeness and consistency are fundamental notions for guaranteeing correctness of a parameterized specification. Also, they are very useful in proofs by induction. Informally, given a conditional specification S and a set of distinguished operators C, called *constructors*, S is said to be sufficiently complete, if any normal form of a ground term is a primitive term, i.e. a term only built from constructors. Guttag showed that this property is undecidable. However, some syntactic criteria can be given. Most of them are based on rewriting methods [Guttag, 1978; Kounalis, 1985; Lazrek *et al.*, 1990]. In the context of conditional parameterized specifications, the art is less developed. This is mostly due to the fact that the problem is much harder. In this paper, we give an effective method for testing this property for parameterized conditional specifications. This method is inspired by [Kounalis, 1985; Bouhoula *et al.*, 1992a] and it is based on the notion of *Pattern* trees.

Another direction is to make use of parameterization at the proof level and to develop a generic proof method. This approach allows us to have shorter and more structured proofs. A generic proof for a parameterized specification must be given only once and can be reused for each instantiation of the parameter. We are interested in automating proof by induction. Many tools for proof by induction have been developed for non-parameterized specifications: The first type applies explicit induction arguments on the term structure [Boyer and Moore, 1979; Bundy *et al.*, 1989; Walther, 1993]. The second type involves a proof by consistency [Musser, 1980; Huet and Hullot, 1982; Jouannaud and Kounalis, 1986; Fribourg, 1986; Kapur and Musser, 1987; Bachmair, 1988]. More recently, new methods were developed that do not rely on the completion framework [Kounalis and Rusinowitch, 1990; Reddy, 1990; Bouhoula *et al.*, 1992a; Bouhoula and Rusinowitch, 1993].

Inductive theory of a parameterized specification are studied by Navarro and Orejas [Navarro and Orejas, 1987]; their results generalize [Padawitz, 1985]. But they do not give effective methods to prove inductive theorems. H. Kirchner has studied proofs by induction in the unconditional case (where the parameter theory is equational) [Kirchner, 1991] using techniques of proof by consistency. K. Becker has dealt with proof by consistency in parameterized positive/negative conditional equational specifications [Becker, 1992]. To conclude, most of the work in proof by induction only considers the techniques of proofs by consistency. It is generally accepted that such techniques may be very inefficient since the completion procedure often diverges. For that reason, we adopt here a method which does not require completion.

The system SPIKE [Bouhoula *et al.*, 1992b] has been developed in this framework. It incorpo-

rates many optimizations such as powerful simplification techniques. To our knowledge, our system is the only one that can prove and disprove inductive theorems in conditional theories without any interaction. Note that NQTHM, CLAM and RRL were not designed to refute false conjectures. SPIKE has proved several interesting theorems in a completely automatic way, that is, without interaction with the user and without ad-hoc heuristics. It has also proved the challenging Gilbreath card trick with only 2 easy lemmas which are given in the beginning of the proof [Bouhoula and Rusinowitch, 1993a]. This example was treated by B. Boyer in NQTHM and H. Zhang in RRL. Unlike SPIKE, they require a lot of lemmas, some of them being non-obvious.

We give in this paper a new procedure for proof by induction in parameterized conditional specifications. Our procedure relies on the notion of *test set* which can be seen as a special induction scheme that allows us to refute false conjectures by the construction of a counter-example. Our definition of test set is more general than the previous one given in [Bouhoula and Rusinowitch, 1993]. It permits us to obtain a smaller test set, which improves efficiency. This definition together with a new notion of provable inconsistency permits us to refute more false conjectures than our previous definitions [Bouhoula and Rusinowitch, 1993], in particular if the specifications are not sufficiently complete and the constructors are not free. As in our previous procedure [Bouhoula *et al.*, 1992a; Bouhoula and Rusinowitch, 1993], to prove conjectures, we just instantiate them with terms from the test set at induction positions and simplify them by axioms, other conjectures or induction hypotheses. The method does not require any hierarchy between the lemmas. They are all stored in a single list and using conjectures for mutual simplification simulates *simultaneous induction*. Unlike our previous method [Bouhoula and Rusinowitch, 1993], this new procedure when limited to non-parameterized conditional specifications, can refute general clauses; refutational completeness is also preserved for boolean ground convergent rewrite systems even if the functions are not sufficiently complete and the constructors are not free. The method has been implemented in the prover SPIKE. Based on computer experiments, the method appears to be more practical and more efficient than inductive theorem provers in non-parameterized specifications.

The organization of this paper is as follows: In section 2, we briefly introduce basic concepts about term rewriting. In section 3, we characterize the inductive theory defined by a parameterized specification. We present in section 4 the procedure for testing sufficient completeness and we prove its correctness and completeness. We also describe a session with SPIKE to give an idea about the interaction with the user if the specification is not sufficiently complete. In section 5, we define the notions of induction variables and test sets and provide an algorithm to compute a test set even if the constructors are not free. We show how test sets can refute false conjectures and also illustrate with the help of an example the fact that we can refute more false conjectures than in [Bouhoula and Rusinowitch, 1993], in particular if the specifications are not sufficiently complete. In Section 6 we define the notions of inductive theory and inductive rewriting, which is a fundamental tool for proving inductive theorems. In section 7, we give a general inference system to perform induction and to refute false conjectures and we show its correctness. The strategy is proved refutationally complete for conditional equations with boolean preconditions if the defined functions are weakly complete (subsection 7.3). Section 8 is dedicated to a computer experiment with our SPIKE system. We give a comparison with our previous method for non-parameterized specifications and we show how proofs in parameterized specifications are shorter and more structured.

2 Basic concepts

We assume that the reader is familiar with the basic concepts of term rewriting, equational reasoning and mathematical logic. We introduce the essential terminology below and refer to [Dershowitz and Jouannaud, 1990] for a more detailed presentation.

A many sorted signature Σ is a pair (S, F) where S is a set of *sorts* and F is a finite set of function symbols. For short, a many sorted signature Σ will simply denoted by F. We assume that we have a partition of F in two subsets, the first one, C, contains the *constructor symbols* and the second, D, is the set of *defined symbols.*

Let X be a family of sorted variables and let $T(F, X)$ be the set of well-sorted F-terms. $Var(t)$ stands for the set of all variables appearing in t and $\sharp(x, t)$ denotes the number of occurrences of the variable x in t. A variable x in t is *linear* iff $\sharp(x, t) = 1$. If $Var(t)$ is empty then t is a *ground* term. By $T(F)$ we denote the set of all ground terms. From now on, we assume that there exists at least one ground term of each non-parameter sort.

Let N^* be the set of sequences of positive integers. For any term t, $occ(t) \subseteq N^*$ denotes its set of positions and the expression t/u denotes the *subterm of t at a position u.* We write $t[s]_u$ (resp. $t[s]$) to indicate that s is a subterm of t at position u (resp. at some position). The top position is written ε. Let $t(u)$ denote the symbol of t at position u. A position u in a term t is said to be *a strict position* if $t(u) = f \in F$, a *linear variable position* if $t(u) = x \in X$ and $\sharp(x, t) = 1$, a *non-linear variable position* if $t(u) = x \in X$ and $\sharp(x, t) > 1$. We use $sdom(t)$ to denote the set of strict positions in t. If u is a position, then $|u|$ (the *length* of the corresponding string) gives us its *depth.* If t is a term, then $|t|$ is the maximum of the depths of $occ(t)$. The symbol $\equiv$ is used for syntactic equality between two objects.

A F-substitution assigns F-terms of appropriate sorts to variables. Composition of substitutions σ and η is written by $\sigma\eta$. The F-term $t\eta$ obtained by applying a substitution η to t is called an *instance* of t. If η applies every variable of its domain to a ground term then we say that η is a ground substitution. If $t\eta$ is ground then it is a *ground instance* of t. A term t unifies with a term s if there exists a substitution σ such that $t\sigma \equiv s\sigma$.

A *conditional* F-equation is a F-equation of the following form: $s_1 = t_1 \wedge \cdots \wedge s_n = t_n \Rightarrow s_0 = t_0$ where $n \geq 0$ and $s_i, t_i \in T(F, X)$ are terms of the same sort. A F-clause is an expression of the form $\neg(s_1 = t_1) \vee \neg(s_2 = t_2) \vee \cdots \vee \neg(s_n = t_n) \vee (s'_1 = t'_1) \vee \cdots \vee (s'_m = t'_m)$. When F is clear from the context we omit the prefix F. A clause is *positive* if $\neg$ does not occur in it. Let c_1 and c_2 be two clauses such that $c_1\sigma$ is a subclause of c_2 for some substitution σ, then we say that c_1 subsumes c_2. Let H be a set of clauses and C be a clause, we say that C is a logical consequence of H if C is valid in any model of H. This will be denoted by $H \models C$.

In the following, we suppose that $\succ$ is a transitive irreflexive relation on the set of terms, that is noetherian, monotonic ($s \succ t$ implies $w[s] \succ w[t]$), stable ($s \succ t$ implies $s\sigma \succ t\sigma$) and satisfy the proper subterm property ($f(t_1, \cdots, t_n) \succ t$, for all $t \in T(F, X)$). We also assume that the ordering $\succ$ can be extended consistently when adding new constants to the signature. The multiset extension of $\succ$ will be denoted by $\gg$.

A conditional equation $a_1 = b_1 \wedge \cdots a_n = b_n \Rightarrow l = r$ will be written as $a_1 = b_1 \wedge \cdots \wedge a_n = b_n \Rightarrow l \rightarrow r$ if $\{l\sigma\} \gg \{t\sigma, a_1\sigma, b_1\sigma, \cdots, a_n\sigma, b_n\sigma\}$ for each substitution σ and $Var(l)$ contains $Var(r) \cup Var(p)$ where $p \equiv \wedge_{i=1}^{n} a_i = b_i$; in that case we say that $a_1 = b_1 \wedge \cdots a_n = b_n \Rightarrow l \rightarrow r$ is a *conditional rule.* The term l is the *left-hand side* of the rule. From now on, we assume that for each conditional rule $p \Rightarrow l \rightarrow r$, if $l \in T(C, X)$, then $r \in T(C, X)$. A conditional rule is used to rewrite terms by replacing an instance of the left-hand side with the corresponding instance of the

right-hand side (but not in the opposite direction) provided the conditions hold. The conditions are checked recursively. Termination is ensured because the conditions are smaller (w.r.t. to $\succ$) than the conclusion. A set of conditional rules is called a conditional rewrite system. We can define the one-step rewrite relation $\rightarrow_R$ and its reflexive-transitive closure $\rightarrow_R^*$ as follows:

Definition 1 (Conditional Rewriting) *Let R be a set of conditional equations. Let t be a term and u a position in t. We write: $t[l\sigma]_u \rightarrow_R t[r\sigma]_u$ if there is a substitution σ and a conditional equation $\wedge_{i=1}^n a_i = b_i \Rightarrow l = r$ in R such that:*

1. *$l\sigma \succ r\sigma$.*
2. *for all $i \in [1 \cdots n]$ there exists c_i such that $a_i\sigma \rightarrow_R^* c_i$ and $b_i\sigma \rightarrow_R^* c_i$.*
3. *$\{t[s\sigma]_u\} \gg \{a_1\sigma,\ b_1\sigma,\ \cdots,\ a_n\sigma,\ b_n\sigma\}$.*

A term t is *R-irreducible* (or in *normal form*) if there is no term s such that $t \rightarrow_R s$. We say that two terms s and t are joinable, denoted by $s \downarrow_R t$, if $s \rightarrow_R^* v$ and $t \rightarrow_R^* v$ for some term v. The rewrite relation $\rightarrow_R$ is said to be noetherian if there is no infinite chain of terms $t_1,\ t_2, \cdots, t_k, \cdots$ such that $t_i \rightarrow_R t_{i+1}$ for all i. The rewrite relation $\rightarrow_R$ is said to be *ground convergent* if the terms u and v are joinable whenever $u,\ v \in T(F)$ and $R \models u = v$.

3 Parameterized conditional specifications

A parameterized conditional specification is a pair $PS = (PAR, BODY)$ of specifications: $PAR = (F_{PAR}, E_{PAR})$ and $BODY = (F_{BODY}, E_{BODY})$ where E_{PAR} is the set of parameter *constraints* consisting of equational clauses in F_{PAR} and E_{BODY} is the set of axioms of the parameterized specification. We assume that these axioms are conditional rules over $F = F_{PAR} \cup F_{BODY}$, where F_{PAR} and F_{BODY} are signatures.

Example 1 *Consider the following parameterized specification: $S_{PAR} = \{bool,\ elem\}$, $F_{PAR} = \{true :\rightarrow bool,\ false :\rightarrow bool,\ \leq : elem \times elem \rightarrow bool,\ dif : elem \times elem \rightarrow bool\}$, E_{PAR} contains the following* constraints*:*

$$true \neq false$$
$$x \leq x = true$$
$$x \leq y = true \vee x \leq y = false$$
$$x \leq y = true \vee y \leq x = true$$
$$x \leq y = false \vee y \leq z = false \vee x \leq z = true$$
$$dif(x, x) = false$$
$$dif(x, y) = true \vee dif(x, y) = false$$

$S = S_{PAR} \cup S_{BODY}$ where $S_{BODY} = \{nat,\ list\}$, $F = F_{PAR} \cup C_{BODY} \cup D_{BODY}$ where $C_{BODY} = \{0 :\rightarrow nat,\ s : nat \rightarrow nat,\ nil :\rightarrow list,\ cons : elem \times list \rightarrow list\}$ and $D_{BODY} = \{count : elem \times list \rightarrow nat,\ sorted : list \rightarrow bool,\ insert : elem \times list \rightarrow list,\ isort : list \rightarrow list\}$, E_{BODY} contains the following conditional rules:

$$\begin{aligned} length(nil) &\rightarrow 0 \\ length(cons(x, y)) &\rightarrow s(length(y)) \\ sorted(nil) &\rightarrow true \end{aligned}$$

$$
\begin{aligned}
sorted(cons(x, nil)) &\rightarrow true && (1)\\
x \leq y = false \Rightarrow sorted(cons(x, cons(y, z))) &\rightarrow false\\
x \leq y = true \Rightarrow sorted(cons(x, cons(y, z))) &\rightarrow sorted(cons(y, z)) && (2)\\
insert(x, nil) &\rightarrow cons(x, nil)\\
x \leq y = true \Rightarrow insert(x, cons(y, z)) &\rightarrow cons(x, cons(y, z))\\
x \leq y = false \Rightarrow insert(x, cons(y, z)) &\rightarrow cons(y, insert(x, z))\\
isort(nil) &\rightarrow nil\\
isort(cons(x, l)) &\rightarrow insert(x, isort(l))
\end{aligned}
$$

3.1 The canonical term algebra

An actualization (see [Ehrig and Mahr, 1985]) of the parameter theory E_{PAR} is a model $\mathcal{A}$ of E_{PAR}. We shall describe $\mathcal{A}$ by its diagram (see [Ehrig and Mahr, 1985; Padawitz, 1987]). For this reason we enrich the signatures by adding new constants $\underline{a}$ for each element a of the carrier A of $\mathcal{A}$. Let $\mathcal{N}(\mathcal{A})$ be the set of new constants and let $F(\mathcal{A}) = F \cup \mathcal{N}(\mathcal{A})$. The diagram $\mathcal{D}(\mathcal{A})$ of $\mathcal{A}$ is the set of (directed) equations $f(\underline{a}_1, \cdots, \underline{a}_n) = \underline{a}$ such that $f \in F_{PAR}$; a_i, $a \in A$ and $f^{\mathcal{A}}(a_1, \cdots, a_n) = a$. We denote by $E_{BODY}(\mathcal{A})$ the set $E_{BODY} \cup \mathcal{D}(\mathcal{A})$. For any model $\mathcal{A}$ of E_{PAR}, we define a canonical term algebra $\mathcal{T}(\mathcal{A})$ representing the semantics of the result of an actualization: $\mathcal{T}(\mathcal{A}) = T(F(\mathcal{A}))_{/=_{E_{BODY}(\mathcal{A})}}$ where $=_{E_{BODY}(\mathcal{A})}$ is the smallest congruence on $T(F(\mathcal{A}))$ generated by $E_{BODY(\mathcal{A})}$. An interesting case is when $\mathcal{T}(\mathcal{A})$ is an initial model in the class of $F(\mathcal{A})$-algebras that are models of $E_{BODY}(\mathcal{A})$ for any model $\mathcal{A}$ of E_{PAR}. To guarantee this fact we need that $E_{BODY}(\mathcal{A})$ is consistent (i.e. has a model) for any model $\mathcal{A}$ of E_{PAR}. This result is shown by the following theorem which is analogous to theorem 2.8 from [Padawitz, 1987].

Theorem 1 *If $E_{BODY}(\mathcal{A})$ is consistent for any model $\mathcal{A}$ of E_{PAR}, then $\mathcal{T}(\mathcal{A})$ is initial in the class of $F(\mathcal{A})$-algebras that are models of $E_{BODY}(\mathcal{A})$ for any model $\mathcal{A}$ of E_{PAR}.*

Many works have already be made in order to check consistency of parameterized specifications (see for instance [Ehrig and Mahr, 1985; Padawitz, 1987; Kirchner, 1991; Becker, 1992]).

3.2 Proving inductive theorems w.r.t. parameterized specifications

We shall now define what is an inductive theorem in parameterized specifications. Note that the theorems to be proved are $F - clauses$.

Definition 2 *A $F - clause$ Γ is an inductive theorem for a parameterized specification PS (or inductively valid w.r.t. PS) iff $\mathcal{T}(\mathcal{A})$ is a model of Γ for any model $\mathcal{A}$ of E_{PAR}. This will be denoted by $PS \models_{ind} \Gamma$ or $E_{BODY}(\mathcal{A}) \models_{ind} \Gamma$ for any model $\mathcal{A}$ of E_{PAR}.*

The next lemma which is similar to lemma 9 from [Becker, 1992], gives us a useful characterization of inductive theorems.

Lemma 1 *Let Γ be a $F - clause$, $\Gamma \equiv \neg(u_1 = v_1) \vee \cdots \neg(u_n = v_n) \vee (s_1 = t_1) \cdots \vee (s_m = t_m)$. Then Γ is an inductive theorem w.r.t. PS iff for any model $\mathcal{A}$ of E_{PAR} and for any ground substitution σ over $T(F(\mathcal{A}))$:*

(for all i : $E_{BODY}(\mathcal{A}) \models u_i\sigma = v_i\sigma$) implies (there exists j such that $E_{BODY}(\mathcal{A}) \models s_j\sigma = t_j\sigma$)

4 Sufficient completeness for parameterized specifications

The property of sufficient completeness is in general undecidable. We now give a method for testing this property for conditional parameterized specifications. This method is inspired by [Kounalis, 1985; Bouhoula *et al.*, 1992a] and based on the notion of *Pattern* trees. Let $\mathcal{A}$ be a model of E_{PAR}. If any ground term in $T(F(\mathcal{A}))$ can be expressed only with constructors and elements of $\mathcal{N}(\mathcal{A})$, we say that PS is complete w.r.t. the constructors and parameter (or sufficiently complete). Here is a more formal definition:

Definition 3 (sufficient completeness) *We say that PS is sufficiently complete if and only if for any model $\mathcal{A}$ of E_{PAR}, for all t in $T(F(\mathcal{A}))$ there exists t' in $T(C_{BODY} \cup \mathcal{N}(\mathcal{A}))$ such that $t \rightarrow^*_{E_{BODY}(\mathcal{A})} t'$.*

4.1 How to check sufficient completeness

The main idea behind our test for sufficient completeness is to compute a pattern tree for every f in D_{BODY}. The leaves of the tree give a partition of the possible arguments for f. If all leaves are "pseudo reducible by PS", then the answer is affirmative. To compute pattern trees, we use the following notions: Let f be a function symbol in F_{BODY}, we say that t is in $Def(f)$, if t is of the form $f(w_1, \cdots, w_n)$ with for all i, $w_i \in T(F, X)$. Let t be a term and u a variable position in t, we say that u is *nullary* if there are only finitely many ground constructor terms with the same sort as $t(u)$. In the following, we present a definition which characterizes induction positions of function symbols in F.

Definition 4 (induction positions) *Let f in F_{BODY}, we define the set of induction positions of functions as follows: $pos\_ind(f) = \{u \mid$ there is $p \Rightarrow g \rightarrow d \in E_{BODY}$ such that $g \in Def(f)$ and u is either a strict and non-top position in g or a non-parameter and non-linear variable position in $g\}$.*

Example 2 (example 1 continued) *The output of the SPIKE procedure that computes induction positions of functions is given in figure 6.* ♦

Example 3 *Consider the following parameterized specification: $S_{PAR} = \{elem\}$, $F_{PAR} = \emptyset$, $E_{PAR} = \emptyset$. $S_{BODY} = \{nat, card\}$, $C_{BODY} = \{0 :\rightarrow nat,\ s : nat \rightarrow nat,\ R :\rightarrow card,\ B :\rightarrow card\}$ et $D_{BODY} = \{f : nat \times nat \times nat \rightarrow nat,\ g : card \times card \rightarrow nat,\ h : elem \times elem \rightarrow nat\}$. E_{BODY} contains the following rules:*

$$\begin{aligned} f(x,y,x) &\rightarrow x \\ f(x,x,y) &\rightarrow x \\ f(y,x,x) &\rightarrow x \\ g(x,y) &\rightarrow 0 \\ h(x,x) &\rightarrow 0 \end{aligned}$$

- *$pos\_ind(f) = \{1, 2, 3\}$. The positions 1, 2 and 3 are induction positions since they are non-parameter and non-linear variable positions.*
- *$pos\_ind(g) = \emptyset$. The positions 1 and 2 are not induction positions since they are either strict positions nor non-parameter and non-linear variable positions.*

- *pos_ind(h) = ∅. The positions 1 and 2 are not induction positions since they are parameter variable positions.* ♦

From any node of the tree labeled by the term $t = f(w_1, \cdots, w_n)$, with $w_i \in T(C_{BODY}, X)$ for all $i \in [1 \cdots n]$, we build the sons of this node by choosing a variable position u of t that is nullary or that is an induction position of f and by making a graft at this occurrence. Each son is thereby labeled by an element of a set of terms called $sons(t, u)$. In this case, we say that t is extensible.

Definition 5 *Let t be a term of the form $f(w_1, \cdots, w_n)$ where for all i, $w_i \in T(C_{BODY}, X)$. Let u be a variable position of t, that is nullary or that belongs to $pos\_ind(f)$. Suppose that $t(u)$ is of sort s. We define $sons(t, u)$ as follows: $sons(t, u) = \{t[u \leftarrow c] \mid c \equiv c_i(x_1, \cdots, x_n)$ where c_i is a constructor with codomain s, n the arity of c_i and $x_1, \cdots, x_n$ are distinct variables $\}$.*

We say that u is an extension position and that t is extensible. The transformation operation of t to $sons(t, u)$ is called the graft of t at the occurrence u. We denote by $pos\_ext(t)$ the set of extension positions of t.

Example 4 (example 3 continued) *Let $t = f(x, y, z)$ and $t' = g(x, y)$ then*

$$sons(t, 2) = \{f(x, 0, z),\ f(x, s(y), z)\} \text{ and } sons(t', 2) = \{g(x, R),\ g(x, B)\}$$

Note that 2 is a nullary *position in t' since the only constructors of sort* card *are R and B.* ♦

Definition 6 (case rewriting) *Let t be a term. Assume there exists a non-empty sequence of conditional rules $C_1 \Rightarrow t_1 \rightarrow r_1$, $C_2 \Rightarrow t_2 \rightarrow r_2$, $\cdots$ $C_n \Rightarrow t_n \rightarrow r_n$ in E_{BODY} and a sequence of positions $u_1, u_2, \cdots, u_n$ in t such that $t/u_1 = t_1\sigma_1$, $t/u_2 = t_2\sigma_2$, $\cdots$, $t/u_n = t_n\sigma_n$ and $C_1\sigma_1 \vee C_2\sigma_2 \vee \cdots \vee C_n\sigma_n$ is an inductive theorem w.r.t. PS. Then, we write:*

$$case\_rewriting(t) = \{C_1\sigma_1 \Rightarrow t[r_1\sigma_1]_{u_1}, \cdots, C_n\sigma_n \Rightarrow t[r_n\sigma_n]_{u_n}\}$$

In this case, t is said to be pseudo reducible *by PS. Otherwise, t is said to be* pseudo irreducible *by PS. This definition can be generalized to the case where t is a clause in a straightforward way.*

Thus, if a term t is *pseudo reducible* by PS, then all its ground instances are reducible.

Example 5 *Consider the following specifications which define* odd *and* even *for non-negative integers: $S_{PAR} = \emptyset$, $F_{PAR} = \emptyset$, $E_{PAR} = \emptyset$. $S_{BODY} = \{nat, bool\}$, $C_{BODY} = \{0 :\rightarrow nat,\ s : nat \rightarrow nat,\ true :\rightarrow bool,\ false :\rightarrow bool\}$ et $D_{BODY} = \{even : nat \rightarrow bool,\ odd : nat \rightarrow bool\}$. E_{BODY} contains the following conditional rules:*

$$\begin{aligned} even(0) &\rightarrow true \\ even(s(0)) &\rightarrow false \\ even(s(s(x))) &\rightarrow even(x) \\ even(x) = true \Rightarrow odd(x) &\rightarrow false \\ even(s(x)) = true \Rightarrow odd(x) &\rightarrow true \end{aligned}$$

The term $odd(x)$ is pseudo reducible *by PS since $even(x) = true \vee even(s(x)) = true$ is an inductive theorem w.r.t. PS. However, the term $even(x)$ is* pseudo irreducible *by PS.* ♦

It is useless to continue the graft process when we meet a node labeled by a term which is pseudo reducible by PS. Then, we can describe the construction of the pattern tree in the following way: from the tree initially constituted from the root $t = f(x_1, \cdots, x_n)$, where n is the arity of f and $x_1, \cdots, x_n$ are distinct variables. We check the pseudo reducibility by PS of t. If t is pseudo irreducible by PS, we build at every step the sons of a node s of the tree by choosing an occurrence in $pos\_ext(s)$ and by making a graft operation on s at this occurrence. The construction of the tree stops if each of its sons is either pseudo reducible by PS or we can no more split it.

stop: $\emptyset \vdash_C (\emptyset, Red, Irred)$

delete reducible leaf: $(Candidats \cup \{t\}, Red, Irred) \vdash_C (Candidats, Red \cup \{t\}, Irred)$
if t is pseudo reducible

decompose: $(Candidats \cup \{t\}, Red, Irred) \vdash_C (Candidats \cup fils(t,u), Red, Irred)$
if t is pseudo irreducible and $u \in pos\_ext(t)$.

delete irreducible leaf: $(Candidats \cup \{t\}, Red, Irred) \vdash_C (Candidats, Red, Irred \cup \{t\})$
if t is pseudo irreducible and $pos\_ext(t) = \emptyset$.

Figure 1: Inference System C

4.2 Inference rules

To check if an operator f in E_{BODY} is sufficiently complete, we apply the rules given in figure 1. *Candidates* is the set of terms candidate for the check of reducibility. *Red* is the set of leaves of the tree which are pseudo reducible. *Irred* is the set of leaves of the tree which are pseudo irreducible and not extensible.

The initial state is $(\{f(x_1, \cdots, x_n)\}, \emptyset, \emptyset)$, where n is the arity of f and $x_1, \cdots, x_n$ are distinct variables. The rule *stop* is applied if the set *candidates* is empty. Then, if *Irred* is empty, we conclude that all the leaves of the pattern tree are pseudo reducible by PS. Consequently, the operator f is sufficiently complete (see theorem 2). If we meet a term t that is pseudo reducible by PS, then the *delete reducible leaf* rule add it to the set *Red* and we continue the check of the pseudo reducibility of the other leaves of the tree. The *decompose* rule expresses the operation of decomposition of a term t at the occurrence u. This rule applies, if we meet a term t that is extensible and pseudo irreducible by PS. The graft operation produces the sons of t, for which we must check pseudo reducibility. Finally, the *delete irreducible leaf* rule is applied if we meet a leaf of the tree that is not extensible and pseudo irreducible by PS. In this case we add the term t to the set *Irred* and we continue the check of the pseudo reducibility of the other leaves of the tree.

Example 6 (example 1 continued) *The pattern tree of* insert *is computed by SPIKE (see figure 4). All the leaves are pseudo reducible by PS, then we conclude that* count *is sufficiently complete.* ♦

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

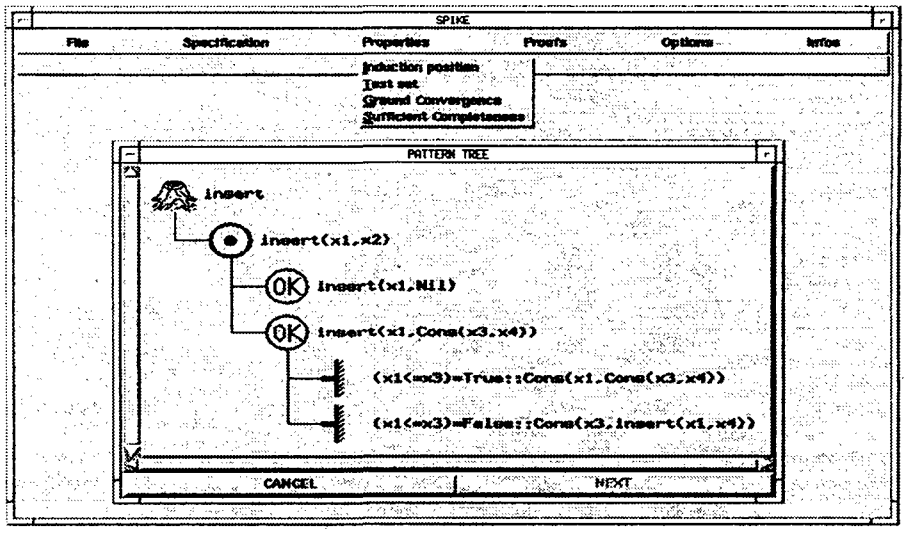

Figure 2: The function *insert* is sufficiently complete

Lemma 2 *Let t be a term $f(x_1, \cdots, x_n)$ with $f \in D_{BODY}$ and $x_1, \cdots x_n$ are distinct variables. The pattern tree of f, computed by C, is bounded.*

proof: The rules of R which have the function symbol f at the top is finite. This involves that the set $pos\_ind(f)$ is finite too. As a consequence the set $occ\_var(t) \cap pos\_ind(f)$ decreases during the construction of the tree since consecutive grafts in the same branch of the tree are made at deeper and deeper occurences. On the other hand, a nullary position correspond to a finite set of constructor terms. Consequently, the height of the pattern tree is bounded. □

4.2.1 Correctness and completeness

In the following we denote by $C_{BODY}(\mathcal{A})$ the set $C_{BODY} \cup \mathcal{N}(\mathcal{A})$. A term t is *strongly* irreducible by E_{BODY} (or *strongly E_{BODY}-irreducible*) if none of its non-variable subterms matches a left-hand side of E_{BODY}. Otherwise, we say that t is *strongly reducible* by E_{BODY}.

Theorem 2 *Let PS be a parameterized specification such that for all model $\mathcal{A}$ of E_{PAR} $\rightarrow_{E_{BODY(\mathcal{A})}}$ is ground convergent over $T(F(\mathcal{A}))$. If for any f in D_{BODY}, there exists a sequence of states $(\{f(x_1, \cdots, x_n)\}, \emptyset, \emptyset) \vdash_C \cdots \vdash_C (\emptyset, Red, \emptyset)$, then PS is sufficiently complete.*

proof: Let f be a function symbol and suppose that there exists a sequence of states

$$(\{f(x_1, \cdots, x_n)\}, \emptyset, \emptyset) \vdash_C \cdots \vdash_C (\emptyset, Red, \emptyset)$$

Let $\mathcal{A}$ be a model of E_{PAR}, we have to prove the following property

$$\mathcal{P}(t) : \forall t \in T(F(\mathcal{A})), \exists t' \in T(C_{BODY}(\mathcal{A})) \text{ such that } t \rightarrow^*_{E_{BODY}(\mathcal{A})} t'$$

We proceed by induction on t w.r.t. $\succ$ which is compatible [1] with $\rightarrow_{E_{BODY}(\mathcal{A})}$. Without loss of generality, we can assume that $t = f(t_1, \cdots, t_n)$ with f in D_{BODY} and for all i we have t_i in $C_{BODY}(\mathcal{A})$. Then, there exists a leaf s of the pattern tree and a ground substitution σ over $T(C_{BODY}(\mathcal{A}))$ such that $s\sigma = t$. Since s is pseudo reducible by PS, then there exists a non-empty

[1] Two noetherian orders $\succ_1$ and $\succ_2$ are compatible if they are both included in a noetherian order.

sequence of conditional rules $C_1 \Rightarrow t_1 \rightarrow r_1$, $C_2 \Rightarrow t_2 \rightarrow r_2$, $\cdots$ $C_n \Rightarrow t_n \rightarrow r_n$ in E_{BODY} and a sequence of positions $u_1, u_2, \cdots, u_n$ in s such that $s/u_1 = t_1\sigma_1$, $s/u_2 = t_2\sigma_2$, $\cdots$, $s/u_n = t_n\sigma_n$ and $C_1\sigma_1 \vee C_2\sigma_2 \vee \cdots \vee C_n\sigma_n$ is an inductive theorem w.r.t. PS. Then, there exists k such that $t_k\sigma_k \rightarrow_{E_{BODY}(\mathcal{A})} r_k\sigma_k$, since for all model $\mathcal{A}$ of E_{PAR} $\rightarrow_{E_{BODY}(\mathcal{A})}$ is ground convergent over $T(F(\mathcal{A}))$. This implies that $t_k\sigma_k\sigma \rightarrow_{E_{BODY}(\mathcal{A})} r_k\sigma_k\sigma$ since $\rightarrow_{E_{BODY}(\mathcal{A})}$ is stable by substitution. On the other hand, we have $r_k\sigma_k\sigma \prec t$ and $r_k\sigma_k\sigma \in T(F(\mathcal{A}))$ since $C_k \Rightarrow t_k \rightarrow r_k$ is a conditional rule. Then by induction hypothesis, we conclude that there exists t' in $T(C_{BODY}(\mathcal{A}))$ such that $r_k\sigma_k\sigma \rightarrow^*_{E_{BODY}(\mathcal{A})} t'$ and therefore there exists t'' in $T(C_{BODY}(\mathcal{A}))$ such that $t \rightarrow^*_{E_{BODY}(\mathcal{A})} t''$. □

The completeness of the procedure is shown by the following theorem:

Theorem 3 *Let PS be a parameterized specification. Suppose that the constructors are free, all parameter variables in the left-hand sides of E_{BODY} are linear and if the defined function* g *appears in a left-hand side of a conditional rule in E_{BODY}, then every rule in E_{BODY} that contains* g *in its left-hand side is linear. If PS is sufficiently complet, then there exists a sequence of states $(\{f(x_1, \cdots, x_n)\}, \emptyset, \emptyset) \vdash_C \cdots \vdash_C (\emptyset, Red, \emptyset)$.*

proof: Assume that PS is sufficiently complete. Suppose that there exists a leaf t which is not extensible and pseudo irreducible by PS. Then, there are two cases to be considered:

a) t is *strongly irreducible* by E_{BODY}. Let $\mathcal{A}$ be a model of E_{PAR} and assume that $\{x_1, \cdots, x_k\}$ be the set of non-parameter variables of t. Let us consider a ground substitution ϕ such that for all parameter variable x of t, $x\phi$ is an element from $\mathcal{N}(\mathcal{A})$ of the same sort as x, and for all $i \in [1 \cdots k]$, $x_i\phi$ is strongly $E_{BODY}(\mathcal{A})$-irreducible, and:

1. $\forall\, i \in [1 \cdots k], |x_i\phi| > |t|$,
2. $\forall\, i, j \in [1 \cdots k], i \neq j, \;\; ||x_i\phi| - |x_j\phi|| > |t|$.

Note that such ϕ exists thanks to the fact that t is not extensible and the constructors are free, so we can choose $x_i\phi$ among the terms built from constructor symbols and elements of $\mathcal{N}(\mathcal{A})$. Assume now that $t\phi$ contains an instance of a left-hand side g of a rule in E_{BODY}. Since any $x_i\phi$ is strongly E_{BODY}-irreducible, there is a strict position u in t such that $t\phi/u$ is an instance of g. Let v be a position of g such that g/v is a function symbol. t/uv is a function symbol since t is not extensible. We consider two cases:

a.1) Assume that g is linear. We can define a substitution σ such that for every variable x that occurs at position w of g we have $\sigma(x) = t/uw$. Such a substitution exists by the linearity of g. We then have $t/u = g\sigma$ which contradicts the assumption that t is strongly E_{BODY}-irreducible.

a.2) Assume that g is non-linear. Since t is not an instance of g (and $t/uw = g/w$ for every strict position w of g) there exist two occurrences u_1 and u_2 of a non-parameter variable x in g (since all parameter variables in the left-hand sides of E_{BODY} are linear) such that: $t/uu_1 \neq t/uu_2$ and $t\phi/uu_1 = t\phi/uu_2$. There are three cases to be considered:

a.2.1) if t/uu_1 and t/uu_2 are ground. In this case $t/uu_1 = t\phi/uu_1$ and $t/uu_2 = t\phi/uu_2$. Therefore $t/uu_1 = t/uu_2$, which is a contradiction.

a.2.2) if t/uu_1 is ground and t/uu_2 non-ground. Then some x_i occurs in t/uu_2. We have $|x_i\phi| > |t|$ by construction of ϕ and therefore $|t\phi/uu_2| > |t|$. On the other hand, $|t\phi/uu_2| = |t\phi/uu_1| = |t/uu_1| \leq |t|$, which is a contradiction.

a.2.3) if t/uu_1 and t/uu_2 are non-ground. Then there is an occurrence v and a variable x_k such that $t/uu_1v = x_k$ and $t/uu_2v \neq x_k$.

- If t/uu_2v is ground the proof is similar to a.2.2
- If t/uu_2v is non-ground let $Var(t/uu_2v) = \{x_{i_1}, \cdots, x_{i_m}\}$.
  - If $x_k \in Var(t/uu_2v)$ then $|t\phi/uu_1v| < |t\phi/uu_2v|$ and therefore we cannot have $t\phi/uu_1 = t\phi/uu_2$ as this leeds to a contradiction.
  - If $x_k \notin Var(t/uu_2v)$ then let x_j be the variable in $Var(t/uu_2v)$ such that $|x_j\phi| = max_{l=1,\cdots,m}|x_{j_l}\phi|$.
    * If $|x_k\phi| > |x_j\phi| + |t|$ then $|t\phi/uu_1v| = |x_k\phi| > |x_j\phi| + |t| > |t\phi/uu_2v|$
    * If $|x_j\phi| > |x_k\phi| + |t|$ then $|t\phi/uu_2v| \geq |x_j\phi| > |x_k\phi| + |t| > |t\phi/uu_1v| = |x_k\phi|$ and we derive a contradiction too.

Therefore $t\phi$ is strongly irreducible by E_{BODY}. On the other hand, t does not contain any parameter function, so $t\phi$ is ground and irreducible by $E_{BODY}(\mathcal{A})$. This contradicts the assumption.

b) Otherwise, t is *strongly reducible* by E_{BODY}. Let $L = \{c_1 \Rightarrow l_1 \rightarrow r_1, \cdots, c_n \Rightarrow l_n \rightarrow r_n\}$ be the non-empty set of all conditional rules in E_{BODY} such that there exists $u_1, \cdots, u_n$ with $t/u_1 = l_1\sigma_1, \cdots, t/u_n = l_n\sigma_n$. Since t is pseudo irreducible by PS, $C \equiv C_1\sigma_1 \vee \cdots \vee C_n\sigma_n$ is not an inductive theorem of PS. Then there exists a model $\mathcal{A}$ of E_{PAR} and a substitution τ over $T(F(\mathcal{A}))$ such that $E_{BODY}(\mathcal{A}) \not\models_{ind} C\tau$.

Then, $t\tau$ cannot be reducible at the top. Assume otherwise that there exists a rule $r \in E_{BODY} - L$ with left-hand side g that applies to $t\tau$ and $t\tau = g\sigma$. Note that every non-variable position of g is a non-variable position of t since t is not extensible. On the other hand, g is linear by hypotheses. So we can define a substitution ρ by $x\rho = t/w$ for every variable x that occurs at some position w of g. We have then $t = g\rho$, in contradiction with the assumption that L contains all the rules whose left-hand side matches t.

The term $t\tau$ cannot be reducible at another position since no proper subterm of $t\tau$ contains a defined symbol and since the constructors are free. This leads to a contradiction. □

4.3 Sufficient completeness with SPIKE

SPIKE checks automatically if an operator f in a specification PS is sufficiently complete. The program builds a pattern tree for f. The leaves of the tree give a partition of the possible arguments for f. If all the leaves are *pseudo reducible* by PS, the answer is affirmative. If one of the leaves is not extensible and *pseudo irreducible* by PS, then SPIKE suggests new rules for completing the specification. This rules are not entirely determined but rather possible schemes for them are proposed, every rule is of the form: *(condition, left-hand-side)*. Once the user has chosen the new rules, usually by simply giving their right-hand sides, SPIKE replays the test. Consider example 1 and suppose that *sorted* is defined by the rules 1 and 2 and therefore it is not sufficiently complete. Here we describe a session with SPIKE to give an idea about the interaction with the user if the

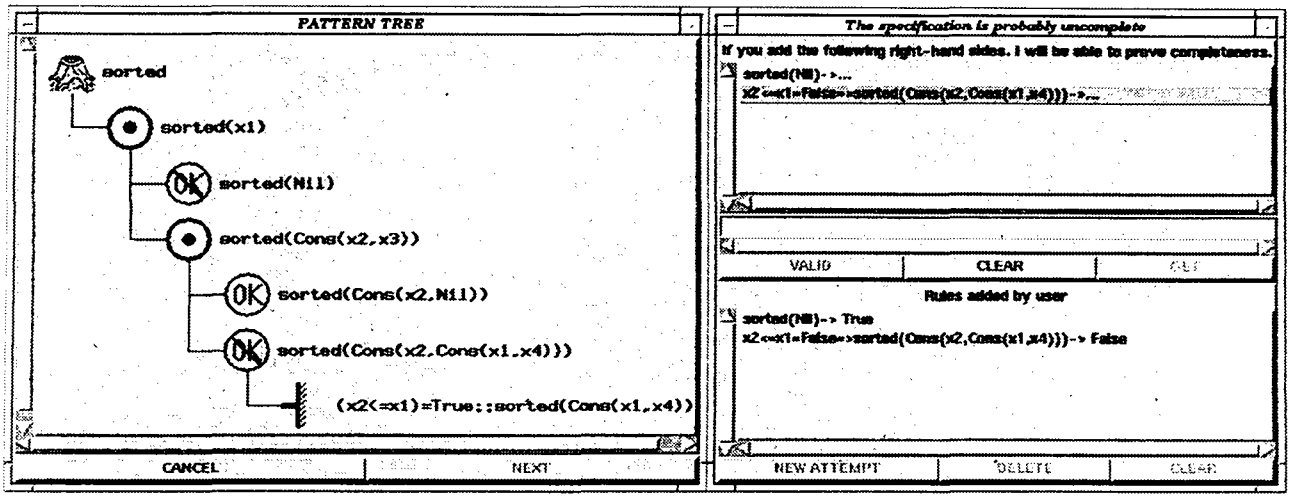

Figure 3: The function *sorted* is not sufficiently complete

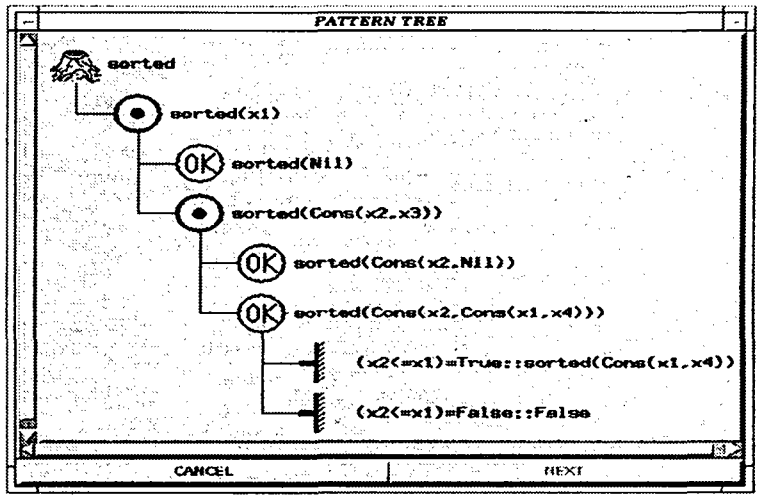

Figure 4: The function *sorted* is now sufficiently complete

specification is not sufficiently complete (see figure 3). Then we add two rules and try again (see figure 4).

5 Selection of induction schemes

To perform a proof by induction, it is necessary to provide some induction schema. In our framework these schema are defined *first* by a function which, given a conjecture, selects the positions of variables where induction will be applied and *second* by a special set of terms called a test set with which the induction variables are instantiated. In general the selection of good induction positions leads to drastic improvements.

5.1 How to get induction variables

Given a specification, we start by computing a set of induction positions of function symbols (see definition 4). This computation is done only once and it permits us to decide whether a variable position of a term t is an induction variable or not. We say that a variable x of sort s is *nullary* if for

all model $\mathcal{A}$ of E_{PAR}, there exists a finite set of ground terms of sort s (i.e. in $T(F(\mathcal{A}))$) irreducible by $E_{BODY}(\mathcal{A})$.

Definition 7 (induction variable) *Given a term t, an* induction variable *of t is a variable of non-parameter sort s that is* nullary *or that occurs at a position $u.v$ of t such that v is an induction position of the top of t/u (if t is a non-parameter variable then it is considered as an induction variable).*

Example 7 (example 1 continued) *y is the only induction variable of $insert(x, insert(x, y))$ because y occurs at position 2 of the subterm $insert(x, y)$ and 2 in an induction position of insert. There is no induction variable in $insert(x, cons(y, cons(z, t)))$.* ♦

5.2 Test set

A test set can be seen as a special induction scheme that permits us to refute false conjectures by the construction of a counter-example. The definition of a test set given below is more general than the one in [Bouhoula and Rusinowitch, 1993]. It permits us to refute false conjectures even if the constructors are not free. On the other hand, with the new definition, we obtain a smaller test set, which improves the efficiency of the proof procedure (see section 8).

To define test set, we use the following notions: A rewrite rule $c \Rightarrow l \rightarrow r$ is *left-linear* if l is linear. A rewrite system E_{BODY} is *left-linear* if every rule in E_{BODY} is left-linear, otherwise E_{BODY} is said to be *non-left-linear*. If t is a term, then the *depth* of t is the maximum of the depths of the positions in t and denoted *depth(t)*. The *strict depth* of t, written as *sdepth(t)*, is the maximum of the depths of the strict positions in t. The *depth of a rewrite system* E_{BODY}, denoted $depth(E_{BODY})$, is defined as the maximum of the depths of the left-hand sides of E_{BODY}. Similarly, the *strict depth* of E_{BODY} denoted by $sdepth(E_{BODY})$, is the maximum of the depths of the strict positions in the left-hand sides of E_{BODY}.

We define the number $D(PS)$ to be $depth(E_{BODY}) - 1$ if $sdepth(E_{BODY}) < depth(E_{BODY})$ and E_{BODY} is left linear, $depth(E_{BODY})$ otherwise.

A term t is weakly PS-irreducible if for all rule $p \Rightarrow g \rightarrow d$ in E_{BODY} such that g matches a subterm of t with a substitution τ, we have $p\tau$ unsatisfiable in PS (i.e. for all model $\mathcal{A}$ of E_{PAR} and for all ground substitution λ over $T(F(\mathcal{A}))$, we have $E_{BODY}(\mathcal{A}) \not\models p\tau\lambda$).

We say that a term t is *infinitary* if for any model $\mathcal{A}$ of E_{PAR} and for any position u in t for which t/u is a non-ground term, there exists infinitely many $E_{BODY}(\mathcal{A})$-irreducible ground instances of t whose subterms at position u are distinct.

Definition 8 (test set) *A test set $S(PS)$ for a parameterized specification PS is a finite set of terms over $T(F, X)$ that has the following properties:*

1. *For any model $\mathcal{A}$ of E_{PAR} and for any $E_{BODY}(\mathcal{A})$-irreducible term s in $T(F(\mathcal{A}))$, there exist a term t in $S(PS)$ and a substitution σ such that $t\sigma = s$;*

2. *any non-ground term in $S(PS)$ has non-parameter variables at depth greater than or equal to $D(PS)$;*

3. *For any model $\mathcal{A}$ of E_{PAR} and for any non-ground term t in $S(PS)$, we have: if E_{BODY} is left-linear, then t has at least one ground instance which is $E_{BODY}(\mathcal{A})$-irreducible, otherwise t is* infinitary.

The first property allows us to prove theorems by induction on the domain of irreducible terms rather than on the whole set of terms. Sets of terms with the property **a.** are usually called *cover sets* in the literature. Several proof procedures have been built on cover sets [Reddy, 1990; Zhang *et al.*, 1988]. Note that our method is also valid if we use cover sets rather than test set. However, *cover sets* cannot be used to refute false conjectures. The second and the third properties of test set are fundamental for this purpose.

The next definition provides us with the criteria to reject false conjectures. This definition permits us to refute more false conjectures than in [Kounalis and Rusinowitch, 1990; Bouhoula and Rusinowitch, 1993].

Definition 9 (provably inconsistent) *Given a parameterized specification PS and a test set $S(PS)$. Then a clause $C \equiv \neg(s_1 = t_1) \vee \cdots \vee \neg(s_m = t_m) \vee (g_1 = d_1) \vee \cdots \vee (g_n = d_n)$ is provably inconsistent with respect to PS if there is a test substitution σ of C (i.e. that maps any induction variable of C to a renaming of an element of $S(PS)$) such that:*

1. $E_{PAR} \not\models (g_1 = d_1 \vee \cdots \vee g_n = d_n)\sigma$;
2. *for all* $i \in [1 \cdots m] : s_i\sigma = t_i\sigma$ *is an inductive theorem w.r.t.* PS;
3. *for all* $j \in [1 \cdots n] :$ $g_j\sigma \not\equiv d_j\sigma$ *and the maximal elements of* $\{g_j\sigma, d_j\sigma\}$ *w.r.t.* $\succ$ *are weakly* PS*-irreducible.*

Now we give examples to illustrate the notion of provable inconsistency:

Example 8 *Consider example 1 in which we remove the rule 2. The equation:*

$$sorted(cons(x, cons(x, y)) = false$$

is provably inconsistent since $sorted(cons(x, cons(x, y))$ *does not contain any induction variable and it is weakly* PS*-irreducible since* $x \leq x = false$ *is unsatisfiable in* PS. ♦

Example 9 *Consider the specification of lists of natural numbers. Let R be the set E_{BODY} defined in example 1 in which we remove the rule 2 and add the following rules:*

$$\{0 \leq x \rightarrow true,\ s(x) \leq 0 \rightarrow false,\ s(x) \leq s(y) \rightarrow x \leq y\}$$

Note that sorted is not sufficiently complete. The equation:

$$sorted(cons(0, cons(0, y)) \quad = \quad false \tag{3}$$

is provably inconsistent since it does not contain any induction variable and it is weakly R-irreducible since $0 \leq 0 = false$ is unsatisfiable in R. With the method of [Kounalis and Rusinowitch, 1990; Bouhoula and Rusinowitch, 1993], the equation 3 is not provably inconsistent since it does not contain any induction variable and it contains an instance of a left-hand side of R. ♦

So even if the specifications are not sufficiently complete, we can easily refute false conjectures thanks to the new definition of provably inconsistent, which is not the case for the methods [Kounalis and Rusinowitch, 1990; Bouhoula and Rusinowitch, 1993]. The next result shows that a provably inconsistent clause cannot be inductively valid w.r.t. PS. This is proved by building a well-chosen ground instance of the clause which gives us a counter-example.

Theorem 4 *Given a parameterized specification PS such that all parameter variables in the left-hand sides of E_{BODY} are linear and $\rightarrow_{E_{BODY}(\mathcal{A})}$ is ground convergent for all model $\mathcal{A}$ of E_{PAR}. If a clause C is provably inconsistent, then C is not inductively valid w.r.t. PS.*

proof: Let $C \equiv \neg(s_1 = t_1) \vee \cdots \vee \neg(s_m = t_m) \vee g_1 = d_1 \vee \cdots \vee g_n = d_n$ be a clause which is provably inconsistent with respect to PS. Then there is a test substitution σ of C such that $E_{PAR} \not\models (g_1 = d_1 \vee \cdots \vee g_n = d_n)\sigma$ and for all $i \in [1 \cdots m]$, $PS \models_{ind} (s_i = t_i)\sigma$.

Let $\mathcal{I}$ be the set of the maximal elements of $\{g_j\sigma, d_j\sigma\}$ w.r.t. $\succ$

Then for all $j \in [1 \cdots n]$, $g_j\sigma \not\equiv d_j\sigma$ and every element in $\mathcal{I}$ is weakly PS-irreducible. In order to show that C is not an inductive theorem of PS, it is sufficient to show that there exists a model $\mathcal{A}$ of E_{PAR} and a ground substitution β over $T(F(\mathcal{A}))$ such that $E_{BODY}(\mathcal{A}) \not\models_{ind} (g_1 = d_1 \vee \cdots \vee g_n = d_n)\beta$ since all ground instances of $\neg(s_1 = t_1) \vee \cdots \vee \neg(s_m = t_m)$ are not inductively valid in PS.

Let $Q \equiv (g_1 = d_1 \vee \cdots \vee g_n = d_n)$. We have $E_{PAR} \not\models Q\sigma$, then there exists a model $\mathcal{A}$ of E_{PAR} and a substitution τ over $T(\mathcal{N}(\mathcal{A}))$ such that $\mathcal{D}(\mathcal{A}) \not\models Q\sigma\tau$, where $\mathcal{D}(\mathcal{A})$ is the diagram of $\mathcal{A}$ and $\mathcal{N}(\mathcal{A})$ is the set of new constants added to the initial signatures to describe $\mathcal{A}$. For all t in $\mathcal{I}$, $t\tau$ is weakly PS-irreducible since all parameter variables in the left-hand sides of E_{BODY} are linear. Let $Var(Q\sigma\tau) = \{x_1, \cdots, x_k\}$ and consider a ground substitution ϕ such that $\sigma\tau\phi$ is $E_{BODY}(\mathcal{A})$-irreducible, and if E_{BODY} is not left-linear, we have also:

1. $\forall\, i \in [1 \cdots k], |x_i\phi| > |Q\sigma\tau|$,
2. $\forall\, i, j \in [1 \cdots k], i \neq j, \quad ||x_i\phi| - |x_j\phi|| > |Q\sigma\tau|$.

Note such a substitution instance exists by using clause 3 of the definition of test set.

$\mathcal{D}(\mathcal{A}) \not\models Q\sigma\tau\phi$ since for all $i \in [1 \cdots k]$, we have x_i is a non-parameter variable and $x_i\phi$ is $\mathcal{D}(\mathcal{A})$-irreducible. Assume now that there exists t in $\mathcal{I}$ and a rule $p \Rightarrow g \rightarrow d$ in E_{BODY} and a substitution α such that $g\alpha$ is a subterm of $t\sigma\tau\phi$ and $E_{BODY}(\mathcal{A}) \models p\alpha$. Since $\sigma\tau\phi$ is $E_{BODY}(\mathcal{A})$-irreducible, there is a strict position u in t such that $t\sigma\tau\phi/u$ is an instance of g. Let v be a non-variable position of g. v is a non-variable position of $t\sigma\tau/u$. Otherwise, there are two cases to consider:

1. if $sdepth(E_{BODY}) < depth(E_{BODY})$ and E_{BODY} is left linear, then we have $|v| > D(PS)$, which implies that $|v| \geq depth(E_{BODY})$. Now, since $sdepth(E_{BODY}) < depth(E_{BODY})$ there is a rule whose left-hand side g' satisfies $depth(g') > |v| \geq depth(E_{BODY})$ and $depth(g') \leq depth(E_{BODY})$, absurd.
2. otherwise, we have $|v| > D(PS) = depth(E_{BODY})$ and $|v| \leq depth(E_{BODY})$, absurd.

So necessarily v is a non-variable position of $t\sigma\tau/u$, Now, we reason as in the proof of theorem 4.1. We conclude that $t\sigma\tau$ contains an instance of g with a substitution β such that $\alpha = \beta\lambda$. On the other hand $t\sigma\tau$ is weakly PS-irreducible, then $E_{BODY}(\mathcal{A}) \not\models p\alpha$, which is absurd.

Therefore, $E_{BODY}(\mathcal{A}) \not\models_{ind} Q\sigma\tau\phi$ since $\rightarrow_{E_{BODY}(\mathcal{A})}$ is ground convergent for all $\mathcal{A}$ a model of E_{PAR}. Thus, C is not an inductive theorem of PS. □

5.2.1 How to get test set

The known procedure for computing test set for conditional theories assume that the constructors are free. However, in this section, we give an algorithm to compute a test set in a non-parameterized conditional specification even if the constructors are not free. A term t is inductively reducible by

a rewrite systen R if every ground instance of t is reducible. Plaisted [Plaisted, 1985] proved the decidability of inductive reducibility for finitely many unconditional equations. Note that, it is easy to semi-decide that a term t is not inductively reducible by a conditional rewrite system.

Proposition 1 *Let R be a non-parameterized conditional specification such that $\rightarrow_R$ is noetherian. Assume that R is left-linear and sufficiently complete. Let $\mathcal{T} = \{t \mid t$ is a constructor term of depth $\leq D(R)$ where variables may occur only at depth $D(R)\}$. Then, the subset of $\mathcal{T}$ composed of terms that are not inductively reducible by R, is a test set for R.*

proof: Let $S(R)$ be the test set computed by the proposition and let t in $T(F)$. As R is sufficiently complete, then there exists t' in $T(C)$ such that $t \rightarrow_R^* t'$. On the other hand, $\rightarrow_R$ is noetherian and for each conditional rule $p \Rightarrow l \rightarrow r \in R$, if $l \in T(C,X)$, then $r \in T(C,X)$. Therefore, there exists $t'' \in T(C)$ such that $t' \rightarrow_R^* t''$ and t'' is irreducible by R. This implies that $t \rightarrow_R^* t''$. So any irreducible term in $T(F)$ is built only with constructors and therefore is an instance of an element of $S(R)$. The second property of definition 8 is trivially verified by construction. Let us check the third property of definition 8. Any non-ground term t in $S(R)$ has at least one ground instance which is R-irreducible since t is not inductively reducible by R. □

Note that if the constructors are specified by a set of unconditional equations, then we can decide inductive reducibility of constructor terms.

Example 10 ([Kaplan, 1984]) *Let R be the set of conditional rules:*

$$\begin{aligned}
0 < 0 &\rightarrow true \\
0 < p(0) &\rightarrow false \\
s(x) < y &\rightarrow x < p(y) \\
p(x) < y &\rightarrow x < s(y) \\
s(p(x)) &\rightarrow x \\
p(s(x)) &\rightarrow x \\
0 < x = true \Rightarrow 0 < s(x) &\rightarrow true \\
0 < x = false \Rightarrow 0 < p(x) &\rightarrow false
\end{aligned}$$

The test set here is:

$$\{0,\ p(0),\ p(p(x)),\ s(0),\ s(s(x)),\ true,\ false\}$$

♦

We can also compute a test set in a parameterized conditional specification if the constructors are free.

Proposition 2 *Assume that $\rightarrow_{E_{BODY}(A)}$ is noetherian [2], PS is sufficiently complete and the constructors are free. Then, the set $\mathcal{T}$ of constructor terms (up to variable renaming) of depth $\leq D(PS)$ where non-parameter variables are not nullary and may occur only at depth $D(PS)$ is a test set for PS.*

[2] to guarantee that $\rightarrow_{E_{BODY}(A)}$ is noetherian, it is sufficient to assume that $\rightarrow_{E_{BODY}}$ is noetherian and no left-hand side of an equation of E_{BODY} contains a symbol from F_{PAR}.

proof: Let $\mathcal{A}$ be a model of E_{PAR} and t in $T(F(\mathcal{A}))$. As PS is sufficiently complete, then there exists t' in $T(C_{BODY}(\mathcal{A}))$ such that $t \rightarrow^*_{E_{BODY}(\mathcal{A})} t'$. On the other hand, $\rightarrow_{E_{BODY}(\mathcal{A})}$ is noetherian and for each conditional rule $p \Rightarrow l \rightarrow r \in E_{BODY}$, if $l \in T(C_{BODY}, X)$, then $r \in T(C_{BODY}, X)$. Therefore, there exists $t'' \in T(C_{BODY}(\mathcal{A}))$ such that, $t' \rightarrow^*_{E_{BODY}(\mathcal{A})} t''$ and t'' is irreducible by $E_{BODY}(\mathcal{A})$. This implies that $t \rightarrow^*_{E_{BODY}(\mathcal{A})} t''$. So any irreducible term in $T(F(\mathcal{A}))$ is built only with constructors and elements of $\mathcal{N}(\mathcal{A})$ and therefore is an instance of an element of $\mathcal{T}$. The second property of definition 8 is trivially verified by construction. Let us check the third property of definition 8. Since the constructors are free, any non-parameter and non-nullary variable x may be substituted by infinitely many different constructor terms. Therefore, any non-ground term in $\mathcal{T}$ is infinitary. □

Example 11 (example 1 continued) *The output of the SPIKE procedure that computes the test set is given in figure 6.* ♦

6 Inductive rewriting

To simplify goals, we generalize the inductive rewriting relation [Bouhoula and Rusinowitch, 1993]. Remember that with inductive rewriting we can check the conditions of a rule to be applied to a clause C with inductive hypothesis, other conjectures (not necessary proved) and the premisses of C, considered as an implication formula. Let us first introduce a few notations. Let $C \equiv \neg(a_1 = b_1) \vee \cdots \vee \neg(a_n = b_n) \vee (c_1 = d_1) \vee \cdots \vee (c_m = d_m)$. Then we denote by $prem(C)$ the set of negated atoms of C: $\{a_i = b_i\}_{i=1,n}$. The expression $(a = b)^\varepsilon$ denotes the literal $a = b$ if $\varepsilon = +$ and the literal $\neg(a = b)$ if $\varepsilon = -$. The skolemized clause $\overline{C}$ of C is the clause obtained by substituting every variable of C by a new constant. We recall that $\succ$ can be extended consistently to terms with new symbols.

The well-founded ordering on clauses is defined by first introducing the complexity of an equation. The complexity of an equation $g = h$ is defined as in [Bouhoula and Rusinowitch, 1993]:

$$C(g = h) = \begin{cases} (\{g\}, \{h\}) & if \quad g \succ h \\ (\{h\}, \{g\}) & if \quad g \prec h \\ (\{g, h\}, \bot) & otherwise \end{cases}$$

where the new symbol $\bot$ is taken to be minimal in $\ll$. We define an ordering on equations as follows: $(a = b) \prec_e (c = d)$ iff $C(a = b)$ is smaller than $C(c = d)$ for the lexicographic composition of $\ll$ on the first and second components of the complexity. The multiset extension of $\prec_e$ will be denoted by $\prec\!\!\prec_e$.

Let C be a clause of type $\wedge_i a_i = b_i \Rightarrow \vee_j c_j = d_j$. We define $Rep(C) = \{C(a_i = b_i)\}_i \cup \{C(c_j = d_j)\}_j$. Given two clauses C_1, C_2, we say that $C_1 \prec_c C_2$ if lexicographically $Rep(C_1) \prec\!\!\prec_e Rep(C_2)$ or $nln(C_1) \dot{<} nln(C_2)$, where $nln(C)$ is the number of negative literals of C.

Definition 10 *Let R be a set of conditional rules and W a set of conditional equations. Consider a clause $C \equiv (a = b)^\varepsilon \vee r$ and its skolemized version $\overline{C} \equiv (\overline{a} = \overline{b})^\varepsilon \vee \overline{r}$. We write:*

$$a \xmapsto{C;r}_{R<W>} a'$$

if either $\overline{a} \rightarrow_{prem(\overline{r})} \overline{a'}$ *and* $a' \prec a$,
or *there exists a position u in a, a substitution σ and a conditional equation $\mathcal{R} \equiv \wedge_{i=1}^n a_i = b_i \Rightarrow s = t$ in $R \cup W$ such that:*

1. $a \equiv a[s\sigma]_u$ *and* $a' \equiv a[t\sigma]_u$.

2. *if* $\mathcal{R} \in W$, *then* $\mathcal{R}\sigma \prec_c C$ *and* $\{a\} \gg \{a_1\sigma,\ b_1\sigma,\ \cdots,a_n\sigma,\ b_n\sigma\}$.

3. $\forall i \in [1 \cdots n]\ \ \exists c'_i, d'_i$ *such that* $a_i\sigma \overset{C;r}{\longmapsto}^*_{R<W>} c'_i$ *and* $b_i\sigma \overset{C;r}{\longmapsto}^*_{R<W>} d'_i$ *and* $\overline{c'_i} =_{prem(\overline{r})} \overline{d'_i}$.

where $=_{prem(\overline{r})}$ *is the congruence generated by* $prem(\overline{r})$.

Definition 11 (inductive rewriting) *Let R be a set of conditional rules and W a set of conditional equations. Consider a clause $C \equiv (a = b)^e \vee r$ and its skolemized version $\overline{C} \equiv (\overline{a} = \overline{b})^e \vee \overline{r}$. We write: $C[a] \mapsto_{R<W>} C[a']$ if and only if $a \overset{C;r}{\longmapsto}_{R<W>} a'$ and $C[a'] \prec_c C[a]$.*

The set W in the definition is intended to contain induction hypotheses and conjectures which are not necessary proved in the proof system described below.

The inductive rewriting is stable by substitution:

Lemma 3 *For all substitution τ: $C \mapsto_{R<W>} C'$ implies $C\tau \mapsto_{R<W>} C'\tau$.*

7 An inductive procedure for parameterized specifications

7.1 Inference rules

Our procedure is defined by a set of transition rules (see figure 5) which are applied to pairs (E, H), where E is the set of conjectures and H is the set of inductive hypotheses. The *generate* rule allows us to derive lemmas and initiates induction steps. The *case simplify* rule simplifies a conjecture with conditional rules where the disjunction of all conditions is inductively valid (note that this case analysis is more general than our previous definition given in [Bouhoula and Rusinowitch, 1993]). The *simplify* rule reduces a clause C with axioms from $E_{BODY} \cup E_{PAR}$, induction hypotheses from H, other conjectures which are not yet proved. The premisses of C considered as a conditional axiom can also help to check that the preconditions of a rule being applied to C are valid. Note that *simplify* permits mutual simplification of conjectures. This rule implements simultaneous induction and is crucial for efficiency. The *subsumption* rule deletes clauses C subsumed by an element of $E_{BODY} \cup E_{PAR} \cup H \cup E$. The role of *deletion* is obvious. The *disproof* rule is applied if a provably inconsistent clause is detected. The *fail* rule is applied to (E, H) if no other rule can be applied to $C \in E$.

An I-derivation is a sequence of states:

$$(E_0, H_0) \vdash_I (E_1, H_1) \vdash_I \cdots \vdash_I (E_n, H_n) \vdash_I \cdots$$

An I-derivation fails if it terminates with the rule *fail* or *disproof*.

7.2 Correctness

The correctness of a procedure based on our inference system relies on a fairness assumption: every conjecture to be checked must be considered at some step. More formally, a derivation $(E_0, H_0) \vdash_I (E_1, H_1) \vdash_I \cdots$ is *fair* if either it fails or it is infinite and the set of persisting clauses $(\cup_{i\geq 0} \cap_{j\geq i} E_j)$ is empty. Then we reason by contradiction: if a non-valid clause is generated in an

generate: $(E \cup \{C\}, H) \vdash_I (E \cup (\cup_\sigma E_\sigma), H \cup \{C\})$

if $C \equiv p \Rightarrow q$ and for every test substitution σ of C
if $E_{PAR} \models q\sigma$, then $E_\sigma = \emptyset$;
if $C\sigma \mapsto_{E_{BODY} \cup E_{PAR} < H \cup E \cup \{C\}>} C'$, then $E_\sigma = \{C'\}$;
otherwise, $E_\sigma = case\_rewriting(C\sigma)$.

case simplify: $(E \cup \{C\}, H) \vdash_I (E \cup E', H)$

if E' = case_rewriting(C)

simplify: $(E \cup \{C\}, H) \vdash_I (E \cup \{C'\}, H)$

if $C \mapsto_{E_{BODY} \cup E_{PAR} < H \cup E>} C'$

subsumption: $(E \cup \{C\}, H) \vdash_I (E, H)$

if C is subsumed by another clause of $E_{BODY} \cup E_{PAR} \cup H \cup E$.

delete: $(E \cup \{C\}, H) \vdash_I (E, H)$

if C is a tautology.

disproof: $(E \cup \{C\}, H) \vdash_I$ *Disproof*

if C is provably inconsistent.

fail: $(E \cup \{C\}, H) \vdash_I \Box$

if no condition of the previous rules hold for C

Figure 5: Inference System I

unfailing derivation then a minimal one is generated too. We show that no inference step can apply to this clause. In other words, this clause persists in the derivation. This contradicts the fairness hypothesis. Therefore, we obtain the following result:

Theorem 5 (correctness) *Let* $(E_0, \emptyset) \vdash_I (E_1, H_1) \vdash_I \cdots$ *be a fair I-derivation. If it does not fail then* $PS \models_{ind} E_0$.

proof: We reason by absurd. Suppose that $PS \not\models_{ind} E_0$ and let $\Im = min_{\prec_c}\{C\sigma \mid C \in \cup_i E_i$ and there is a model $\mathcal{A}$ of E_{PAR} and a ground substitution σ over $T(F(\mathcal{A}))$ that is irreducible by $E_{BODY}(\mathcal{A})$ such that $PS \not\models_{ind} C\sigma\}$. $\Im \neq \emptyset$ since $PS \not\models_{ind} E_0$ and $\prec_c$ is well-founded. Consider a clause C which is minimal in $\Im$ with respect to the subsumption ordering. It is sufficient to prove that C cannot be simplified nor deleted, and that *generate* cannot be applied to C; this shows that *fail* or *disproof* applies since the clause C must not persist in the derivation by the fairness hypothesis. Hence let us assume that $C \in E_j$ and $(E_j, H_j) \vdash_I (E_{j+1}, H_{j+1})$ by some rule applied to C. We discuss now the situation according to which rule is applied. In every case we shall derive a contradiction. In order to simplify the notations we write E for E_j and H for H_j. We show now that wathever rule is applied to C, we obtain a contradiction.

generate: If $(E_j, H_j) \vdash_I (E_{j+1}, H_{j+1})$ by *generate* on $C \equiv p \Rightarrow q$. since σ is a ground substitution over $T(F(\mathcal{A}))$ that is irreducible by $E_{BODY}(\mathcal{A})$, there exists a test substitution σ_0 of C and a substitution θ such that $\sigma = \sigma_0\theta$. $E_{PAR} \not\models q\sigma_0$ since $PS \not\models_{ind} C\sigma_0$, we have two possibilities:

1. if there exists a clause C' such that $C\sigma_0 \mapsto_{E_{BODY} \cup E_{PAR} < H \cup E \cup \{C\}>} C'$ then [3], by the lemma 3, we conclude that $C\sigma \mapsto_{E_{BODY} \cup E_{PAR} < H \cup E \cup \{C\}>} C'\theta$. For each instance $S\tau$ of clauses of $H \cup E \cup \{C\}$ used in the rewriting step, we have $S\tau \prec_c C\sigma$. Then, we have $PS \models_{ind} S\theta$. The premisses of $C\sigma$ are also valid since $PS \not\models_{ind} C\sigma$. Therefore, $PS \not\models_{ind} C'\theta$. On the other hand, $C'\theta \prec_c C\sigma$ and $C' \in \cup_i E_i$. This shows a contradiction since it proves that we can find an instance of a clause in $\cup_i E_i$ which is not valid and smaller than $C\sigma$ with respect to $\prec_c$.

2. Assume that the rule *case_rewriting* is applied to $C\sigma$. Then, consider all the rules: $C_1 \Rightarrow t_1 \rightarrow r_1$, $C_2 \Rightarrow t_2 \rightarrow r_2$, $\cdots$ $C_n \Rightarrow t_n \rightarrow r_n$ in E_{BODY} such that there exists a sequence of positions $u_1, u_2, \cdots, u_n$ in $C\sigma$ and $C\sigma/u_1 = t_1\sigma_1$, $C\sigma/u_2 = t_2\sigma_2$, $\cdots$, $C\sigma/u_n = t_n\sigma_n$ and $C_1\sigma_1 \vee C_2\sigma_2 \vee \cdots \vee C_n\sigma_n$ is an inductive theorem w.r.t. PS. Hence, the result of the application of *case_rewriting* is:

$$\{C_1\sigma_1 \Rightarrow C\sigma[r_1\sigma_1]_{u_1}, \cdots, C_n\sigma_n \Rightarrow C\sigma[r_n\sigma_n]_{u_n}\}$$

Then there exists k such that $PS \models_{ind} C_k\sigma_k$. Let $C' \equiv C_k\sigma_k \Rightarrow C\sigma[r_k\sigma_k]_{u_k}$, we have $PS \models_{ind} (C_k \Rightarrow t_k \rightarrow r_k)\sigma_k$. Therefore, $PS \models_{ind} r_k\sigma_k = t_k\sigma_k$. Putting everything together, we get $PS \not\models_{ind} C'\theta$. On the other hand, $C' \in \cup_i E_i$ and $C'\theta \prec_c C\sigma$, this is also absurd.

case simplify: this case is similar to the previous one.

simplify: Suppose that the *simplify* rule applies to C, then, there exists a clause C' such that $C \mapsto_{E_{BODY} \cup E_{PAR} < H \cup E>} C'$, then, by the lemma 3, we conclude that

$$C\sigma \mapsto_{E_{BODY} \cup E_{PAR} < H \cup E>} C'\sigma$$

For each instance $S\tau$ of clauses of $H \cup E$ used in the rewriting step, we have $S\tau \prec_c C\sigma$. Then, we have $PS \models_{ind} S\tau$. The premisses of $C\sigma$ are also valid since $PS \not\models_{ind} C\sigma$. Therefore, $PS \not\models_{ind} C'\sigma$. On the other hand, $C'\sigma \prec_c C\sigma$ and $C' \in \cup_i E_i$, which is absurd.

subsumption: Since $PS \not\models_{ind} C\sigma$, C cannot be subsumed by a clause of $E_{BODY} \cup E_{PAR}$. If there is $C' \in H \cup (E \setminus \{C\})$ such that $C \equiv C'\tau \vee r$, we have $PS \not\models_{ind} C'\tau\sigma$, so $r = \emptyset$ and $\tau = \mathcal{I}$ since C is minimal in $\Im$ w.r.t. the subsumption ordering. As a consequence $C' \notin (E \setminus \{C\})$. On the other hand, $C' \notin H$. Otherwise, the *generate* rule has been applied to C'. Therefore, *generate* can be also applied to C in contradiction with a previous case. Hence, this rule cannot be applied to C.

delete: Since $PS \not\models_{ind} C\sigma$, C is not a tautology and this rule need not be considered. □

Since every I-derivation from $(E, \emptyset)$ to $(\emptyset, H)$, where H is some set of clauses, is fair then the conjectures of E are inductive consequences of PS. This remark is important from a practical point of view. Note also that E is valid even when the derivation is infinite.

[3] Let R' and W' be two sets of clauses and suppose that R (resp. W) is the set of all conditional rules (resp. equations) of R' (resp. W'). By abuse of notation, the relation $\mapsto_{R' < W'>}$ will be denoted by $\mapsto_{R < W>}$.

If *disproof* is applied at step k, then a provably inconsistent clause is detected and therefore, from theorem 4, we conclude that some conjecture in E_k is false, if for all model $\mathcal{A}$ of E_{PAR} $\rightarrow_{E_{BODY(\mathcal{A})}}$ is ground convergent over $T(F(\mathcal{A}))$ and all parameter variables in the left-hand sides of E_{BODY} are linear. The initial conjectures E_0 is not inductively valid in PS too. This is a consequence of the next result:

Lemma 4 *Let* $(E_0, \emptyset) \vdash_J (E_1, H_1) \vdash_J \cdots$ *be an I-derivation. If for all* i *such that* $i \leq j$ *we have* $PS \models_{ind} E_i$ *then* $PS \models_{ind} E_{j+1}$.

proof: Let C be a clause in E_j and assume that $(E_j, H_j) \vdash_I (E_{j+1}, H_{j+1})$ by application of an inference rule on C. Let us show that $\forall i \leq j \;\; PS \models_{ind} E_i$ implies $PS \models_{ind} E_{j+1}$:

generate: If $(E_j, H_j) \vdash_I (E_{j+1}, H_{j+1})$ by *generate* on $C \equiv p \Rightarrow q$. Let σ be a test substitution of C. If $E_{PAR} \not\models q\sigma$, then there are two cases to consider:

1. if there exists C' such that $C\sigma \rightarrow_{E_{BODY} \cup Rule(E_{PAR})} C'$, then the clauses which are used for the rewriting step occur in some E_k $(k \leq j)$ and therefore are inductively valid in PS by hypothesis. On the other hand, we can assume that the premisses of $C\sigma$ are valid (otherwise the proof is obvious). Hence, E_{j+1} is inductively valid too in PS.

2. Otherwise, there exists a non-empty sequence of conditional rules $P_1 \Rightarrow t_1 \rightarrow r_1$, $P_2 \Rightarrow t_2 \rightarrow r_2$, $\cdots$ $P_n \Rightarrow t_n \rightarrow r_n$ in E_{BODY} and a sequence of positions $u_1, u_2, \cdots, u_n$ in $C\sigma$ such that $C\sigma/u_1 = t_1\tau_1$, $C\sigma/u_2 = t_2\tau_2$, $\cdots$, $C\sigma/u_n = t_n\tau_n$ and $P_1\tau_1 \vee P_2\tau_2 \vee \cdots \vee P_n\tau_n$ is an inductive theorem w.r.t. PS. Hence, the result of the application of *case_rewriting* is:

$$\{P_1\tau_1 \Rightarrow C\sigma[r_1\tau_1]_{u_1}, \cdots, P_n\tau_n \Rightarrow C\sigma[r_n\tau_n]_{u_n}\}$$

Assume that there exists k such that: $PS \not\models_{ind} C_k \equiv P_k\tau_k \Rightarrow C\sigma[r_k\tau_k]_{u_k}$. In other words there is a ground instance $C_k\theta$ over $T(F(\mathcal{A}))$ (we can assume that $C\sigma\theta$ is ground without loss of generality) such that: $PS \not\models_{ind} C_k\theta$, then $PS \models_{ind} P_k\tau_k\theta$ and $PS \not\models_{ind} C\sigma\theta[r_k\tau_k\theta]$. Therefore, $PS \models_{ind} t_k\tau_k\theta = r_k\tau_k\theta$. This implies that $PS \not\models_{ind} C\sigma\theta[t_k\tau_k\theta]$, which is absurd.

case simplify: this case is similar to the previous one.

simplify: If $(E_j, H_j) \vdash_I (E_{j+1}, H_{j+1})$ by *simplify*, then the clauses which are used for simplification occur in some E_k $(k \leq j)$ and therefore are inductively valid in PS by hypothesis. On the other hand, we can assume that the premisses of the clause to be simplified are valid (otherwise the proof is obvious). Hence, E_{j+1} is inductively valid too in PS.

subsumption and delete: If C is deleted then $PS \models_{ind} E_{j+1}$ since $E_{j+1} \subseteq E_j$ in this case. □

The next theorem is a straightforward consequence of the above results:

Theorem 6 (refutation) *Given a parameterized specification PS such that for all model* $\mathcal{A}$ *of* E_{PAR}, $\rightarrow_{E_{BODY(\mathcal{A})}}$ *is ground convergent over* $T(F(\mathcal{A}))$ *and all parameter variables in the left-hand sides of* E_{BODY} *are linear. Let* $(E_0, \emptyset) \vdash_I (E_1, H_1) \vdash_I \cdots$ *be an I-derivation. If there exists* j *such that* disproof *applies to* (E_j, H_j) *then* $PS \not\models_{ind} E_0$.

7.3 Refutational completeness for parameterized specifications with boolean preconditions

In this section, we shall consider axioms that are conditional rules with boolean preconditions. To be more specific, we assume there exists a sort *bool* with two free constructors $\{true,\ false\}$. Every rule in E_{BODY} is of type: $\wedge_{i=1}^{n}\ p_i = p'_i \Rightarrow s \rightarrow t$ where for all i in $[1 \cdots n]$, $p'_i \in \{true,\ false\}$. For $\alpha \in \{true,\ false\}$ we denote by $\overline{\alpha}$ the complementary *bool* symbol of α. Conjectures will be *boolean clauses*, i.e. clauses whose negative literals are of type $\neg(p = p')$ where $p' \in \{true, false\}$. Let f be a function symbol in D_{BODY}. If for all the rules in E_{BODY} of the form $p_i \Rightarrow s \rightarrow t_i$ such that $s \in Def(f)$, we have $PS \models_{ind} \vee_i p_i$, then we say that f is weakly complete w.r.t. PS. We say that PS is weakly complete if any function in D_{BODY} is weakly complete w.r.t. PS.

Note that a *weakly complete* specification is not necessary *sufficiently complete.*

Example 12 *The following rules define the predicates* $\leq$ *and* P *over the constructors* 0 *et* s*:*

$$\begin{aligned} 0 \leq x &\rightarrow true \\ s(x) \leq 0 &\rightarrow false \\ s(x) \leq s(y) &\rightarrow x \leq y \\ s(x) \leq y = true \Rightarrow P(s(x), y) &\rightarrow false \\ s(x) \leq y = false \Rightarrow P(s(x), y) &\rightarrow true \end{aligned}$$

The predicates $\leq$ *and* P *are weakly complete. But* P *is not sufficiently complete since* $P(0,0)$ *cannot be reducible to a constructor term.* ♦

Now, We can define a new inference system J from I by adding the following complement rule which transforms negative clauses to positive clauses that are easier to refute.

complement: $(E \cup \{\neg(a = \alpha) \vee r\}, H) \vdash_J (E \cup \{(a = \overline{\alpha}) \vee r\}, H)$ if $\alpha \in \{true,\ false\}$.

We also remove the fail rule and reformulate *disproof* as follows:

disproof: $(E \cup \{C\}, H) \vdash_J Disproof$ if no condition of the previous rules hold for C

Let us assume that E_0 only contains boolean clauses. The only rule that permits us to introduce negative clauses is *case_rewriting.* Since the axioms have boolean preconditions, all the clauses generated in a J-derivation are boolean. If *disproof* is applied in a J-derivation, then there exists a positive clause C such that *generate* cannot be applied to C. Therefore there exists a test substitution σ such that $E_{PAR} \not\models C\sigma$. Moreover $C\sigma$ does not match any left-hand side of E_{BODY}. Otherwise, the *inductive rewriting* or the *case rewriting* rule can be applied to $C\sigma$ since PS is weakly complete. As a consequence, C is a provably inconsistent clause. So, the new inference system J can be proved refutationally complete for boolean clauses.

Theorem 7 *Given a weakly complete parameterized specification* PS *such that all parameter variables in the left-hand sides of* E_{BODY} *are linear and* $\rightarrow_{E_{BODY}(\mathcal{A})}$ *is ground convergent for all model* $\mathcal{A}$ *of* E_{PAR}*. Let* $(E_0, \emptyset) \vdash_J (E_1, H_1) \vdash_J \cdots$ *be a fair J-derivation such that* E_0 *only contains boolean clauses. Then* $PS \not\models_{ind} E_0$ *iff the derivation ends with disproof.*

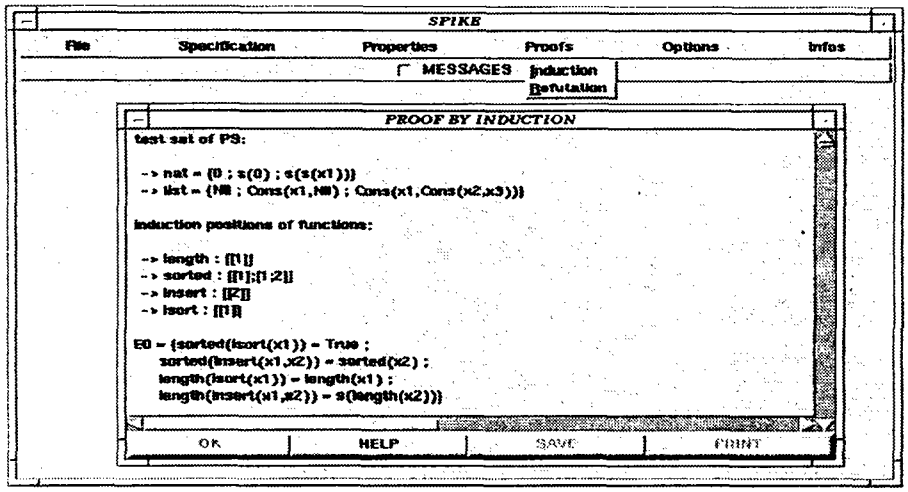

Figure 6: Example

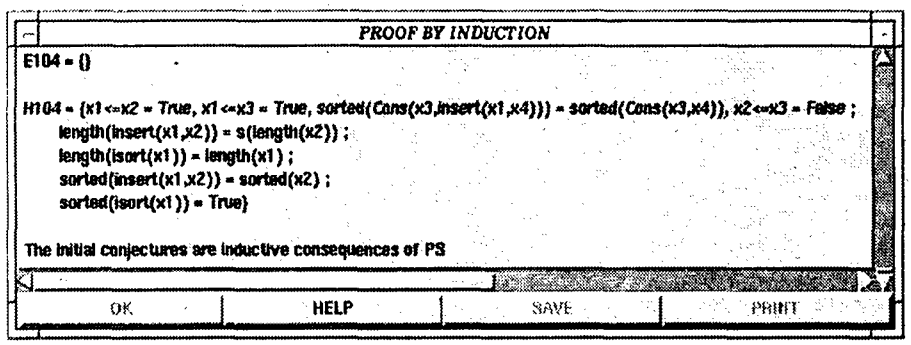

Figure 7: Success

8 Implementation and experimental results

Our implementation is based on the previous inference system. The program is able to prove the validity of a set of clauses in parameterized conditional specifications. Here is an overview of the algorithm. The main data structures are: the list E_{BODY} of axioms, that are conditional rules built with the constructor discipline, the list E of conjectures (clauses) to be checked, the list E_{PAR} of parameter constraints, that are equational clauses in Σ_{PAR} and finally, the set H of induction hypotheses (initialized by $\emptyset$). The first step in a proof session is to check if all defined functions are completely defined. The second step is to compute test set for PS and also induction positions. After these preliminary tasks, the proof starts.

Consider example 1 with E_0 the set of conjectures to be proved (see figure 6). SPIKE can prove these conjectures in a completely automatic way, using *104* steps and taking *200* seconds as shown in figure 7. Note that one lemma (was generated automatically) is sufficient to prove the initial conjectures.

$$x1 \leq x2 = True, x1 \leq x3 = True, sorted(Cons(x3, insert(x1, x4))) = sorted(Cons(x3, x4)), x2 \leq x3 = False$$

Now consider the same example with lists of natural numbers, using the method in [Bouhoula and Rusinowitch, 1993], we have the following test sets (see figure 8). To prove the same conjectures without parameters, SPIKE used *246* steps and took *1656* seconds. In addition, *40* lemmas were

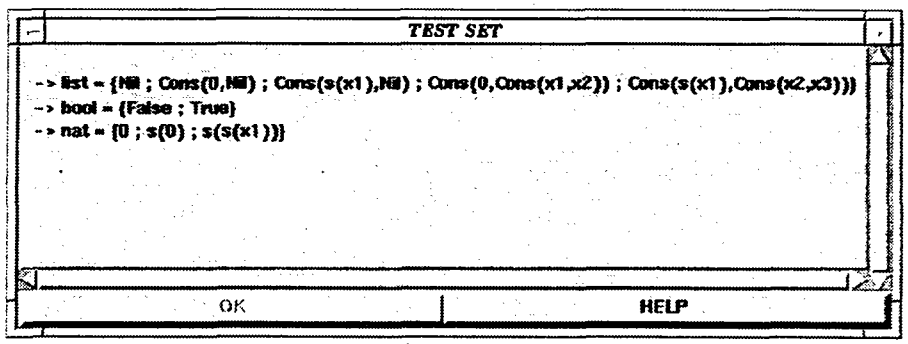

Figure 8: Test sets

generated automatically to prove the conjectures. This example illustrates that with parameterized specifications we have a smaller test set and fewer induction positions, permitting us to obtain shorter and more structured proofs.

9 Conclusion

We have proposed a new procedure for proof by induction in parameterized conditional specifications. Like our previous procedure [Bouhoula *et al.*, 1992a; Bouhoula and Rusinowitch, 1993], it allows simultaneous induction and can handle non-orientable equations. It can also refute non-valid conjectures. Our method is also compatible with simplification rules given in [Bouhoula and Rusinowitch, 1993]. An extension to theories that are presented by non-Horn clauses along the lines of [Becker, 1992] should be easy. The property of sufficient completeness is very important for inductive reasoning but is in general undecidable. We have given a procedure for testing this property for parameterized conditional specifications. Previously, we could compute a test set for conditional specifications only if the constructors were free. Here, we have given a new definition of test set and an algorithm to compute them even if the constructors are not free. We have also proposed a new notion of provable inconsistency which allows us to refute more false conjectures than our previous definition [Bouhoula and Rusinowitch, 1993], in particular if the specifications are not sufficiently complete. Unlike our previous method [Bouhoula and Rusinowitch, 1993], this new procedure, when limited to non-parameterized conditional specifications, can refute general clauses; refutational completeness is also preserved for boolean ground convergent rewrite systems even if the functions are not sufficiently complete and the constructors are not free. Note that our method remains valid in theories without constructors. The method is implemented in the prover SPIKE. This system has proved interesting examples in a completely automatic way, that is, without interaction with the user and without ad-hoc heuristics. Experiments illustrate that proofs in parameterized specifications are shorter and more structured.

We plan to generalize the method to get refutational completeness for a larger class of rewrite systems. Another powerful extension is to allow for generalization techniques, such as in the traditional induction method. How this can be done and the possible implications with respect to soundness and refutational completeness, still have to be studied very carefully.

Acknowledgment: I am very grateful to Hélène Kirchner, Pierre Lescanne, Michaël Rusinowitch and Toby Walsh for valuable discussions and comments on an earlier version of this paper. I would

also like to thank Peter Padawitz, Jürgen Avenhaus and our colleagues from the Indus/Mind group for stimulating discussions.

References

[Bachmair, 1988] L. Bachmair. *Proof by consistency in equational theories.* In *Proceedings 3rd IEEE Symposium on Logic in Computer Science, Cambridge (Mass., USA)*, pages 228–233, 1988.

[Becker, 1992] K. Becker. Inductive Proofs in Specifications Parameterized by a Built-in Theory. SEKI-Report, SR-92-02, 1992.

[Bouhoula *et al.*, 1992a] A. Bouhoula, E. Kounalis, and M. Rusinowitch. Automated mathematical induction. Technical Report 1636, INRIA, 1992.

[Bouhoula *et al.*, 1992b] A. Bouhoula, E. Kounalis, and M. Rusinowitch. Spike: an automatic theorem prover. In *Proceedings of LPAR'92*, volume 624 of *LNAI*, Saint Petersbourg, Russia, July 1992. Springer-Verlag.

[Bouhoula and Rusinowitch, 1993] A. Bouhoula and M. Rusinowitch. Automatic Case Analysis in Proof by Induction. In *Proccedings of the 13th International Joint Conference on Artificial Intelligence*, volume 1, page 88–94. Chambéry France, 1993.

[Bouhoula and Rusinowitch, 1993a] A. Bouhoula and M. Rusinowitch. Implicit induction in conditional theories. Technical Report 2045, INRIA, 1993.

[Boyer and Moore, 1979] R. S. Boyer and J. S. Moore. *A Computational Logic.* Academic Press, New York, 1979.

[Bundy *et al.*, 1989] A. Bundy, F. van Harmelen, A. Smaill, and A. Ireland. Extensions to the rippling-out tactic for guiding inductive proofs. In M. E. Stickel, editor, *10th International Conference on Automated Deduction*, volume 449 of *LNAI*, pages 132–146. Springer-Verlag, July 1989.

[Dershowitz and Jouannaud, 1990] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leuven, editor, *Handbook of Theoretical Computer Science.* Elsevier Science Publishers North-Holland, 1990.

[Ehrig and Mahr, 1985] H. Ehrig, B. Mahr. Fundamentals of Algebraic Specification 1. Equations and Initial Semantics. Springer Verlag, 1985.

[Fribourg, 1986] L. Fribourg. A strong restriction of the inductive completion procedure. In *Proceedings 13th International Colloquium on Automata, Languages and Programming*, volume 226 of *LNCS*, pages 105–115. Springer-Verlag, 1986.

[Guttag, 1978] J. V. Guttag and J. J. Horning The Algebraic Specification of Abstract Data Types. In *Acta Informatica*, 10:27-52, 1978.

[Huet and Hullot, 1982] G. Huet and J.-M. Hullot. Proofs by induction in equational theories with constructors. *Journal of Computer and System Sciences*, 25(2):239–266, October 1982.

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

[Kaplan, 1984] S. Kaplan. Fair conditional term rewriting systems: Unification, termination, and confluence. Technical report, University of Paris Sud, Orsay (France), Orsay, 1984.

[Kapur and Musser, 1987] D. Kapur and D. Musser. Proof by consistency Artificial Intelligence, volume 31(2), pages 125–157, 1987

[Kirchner, 1991] H. Kirchner. Proofs in Parameterized Specifications. In *4th RTA*, volume 488 of *LNCS*, pages 174–187. Springer-Verlag, 1991.

[Kounalis, 1985] E. Kounalis. Completeness in Data Type Specifications. In *Proceeding EUROCAL Conference*. Springer-Verlag, Linz (Austria), 1991.

[Kounalis and Rusinowitch, 1990] E. Kounalis and M. Rusinowitch. Mechanizing inductive reasoning. In *Proceedings of the American Association for Artificial Intelligence Conference, Boston*, pages 240–245. AAAI Press and MIT Press, July 1990.

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

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

[Navarro and Orejas, 1987] M. Navarro and F. Orejas. Parameterized Horn clause specifications: proof theory and correctness. In *Proceeding TAPSOFT Conference*, volume 249 of *LNCS*. pages 202–216, Springer-Verlag, 1987.

[Padawitz, 1985] P. Padawitz. Towards a proof theory of parameterized specifications. In *Semantics of data type's*, volume 173 of *LNCS*, pages 375–391. Springer-Verlag, 1984.

[Padawitz, 1987] P. Padawitz. Parameter-Preserving Data Type Specifications. In *Journal of Computer and System Science*, volume 34, pages 179–209, 1987.

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

[Reddy, 1990] U. S. Reddy. Term rewriting induction. In M. E. Stickel, editor, *Proceedings 10th International Conference on Automated Deduction, Kaiserslautern (Germany)*, volume 449 of *LNCS*, pages 162–177. Springer-Verlag, 1990.

[Walther, 1993] C. Walther. *Combining Induction Axioms By Machine.* In *Proceedings of 13th IJCAI*, Morgan Kaufmann Publishers, volume 1, pages 95–100, 1993

[Zhang *et al.*, 1988] H. Zhang, D. Kapur, and M. S. Krishnamoorthy. A mechanizable induction principle for equational specifications. In E. Lusk and R. Overbeek, editors, *Proceedings 9th International Conference on Automated Deduction, Argonne (Ill., USA)*, volume 310 of *LNCS*, pages 162–181. Springer-Verlag, 1988.

Unité de Recherche INRIA Lorraine
Technopôle de Nancy-Brabois - Campus Scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 VILLERS LES NANCY Cedex (France)

Unité de Recherche INRIA Rennes IRISA, Campus Universitaire de Beaulieu 35042 RENNES Cedex (France)
Unité de Recherche INRIA Rhône-Alpes 46, avenue Félix Viallet - 38031 GRENOBLE Cedex (France)
Unité de Recherche INRIA Rocquencourt Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 LE CHESNAY Cedex (France)
Unité de Recherche INRIA Sophia Antipolis 2004, route des Lucioles - B.P. 93 - 06902 SOPHIA ANTIPOLIS Cedex (France)

EDITEUR
INRIA - Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 LE CHESNAY Cedex (France)

ISSN 0249 - 6399