Fundamental Results on Automated Theorem Proving by Test Set Induction

Adel Bouhoula

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Fundamental Results on Automated Theorem Proving by Test Set Induction

Adel BOUHOULA

N° 2478
Janvier 1995

PROGRAMME 2

Fundamental Results on Automated Theorem Proving by Test Set Induction

ADEL BOUHOULA

INRIA-Lorraine & CRIN
BP 101, 54600 Villers-lès-Nancy, France
email: Adel.Bouhoula@loria.fr

January 18, 1995

Abstract

We present in this paper a general scheme for test set induction procedure and describe a simple technique to prove the correctness of this procedure. Previously, we could only compute a test set for a conditional specification if the constructors were free. Here, we give a new definition of test sets and a procedure to compute them even if the constructors are not free. The method uses a new notion of provable inconsistency and induction positions (that need to be instantiated by induction schemes) which allows us to refute more false conjectures than with previous approaches. We also present an algorithm to compute all the induction positions of a conditional specification. Finally, we propose an induction procedure which is refutationally complete for conditional specifications (not restricted to boolean specifications), in that it refutes any conjecture which is not an inductive theorem. The method has been implemented in SPIKE. Based on computer experiments, SPIKE appears to be more practical and efficient than related systems.

Keywords: Automated reasoning, Theorem Proving, Test set induction, Simultaneous Induction, Term rewriting systems.

Résultats Fondamentaux sur les Preuves par Récurrence avec Ensemble Test

ADEL BOUHOULA

INRIA-Lorraine & CRIN
BP 101, 54600 Villers-lès-Nancy, France
email: bouhoula@loria.fr

Résumé

Nous proposons un schéma général de procédure de preuve par récurrence avec ensemble test. Avec cette procédure, nous décrivons une technique simple de preuve de correction. Dans les travaux précédents, nous ne pouvions calculer un *ensemble test* pour une spécification conditionnelle que dans le cas où les constructeurs étaient 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 des nouvelles notions de *témoin d'incohérence* et de *position de récurrence* qui permettent de réfuter plus de conjectures non valides qu'avec les méthodes précédentes. Nous proposons également un algorithme permettant de calculer toutes les positions de récurrence d'une spécification conditionnelle. Enfin, nous présentons une procédure opérationnelle de preuves par récurrence avec ensembles test. Par opposition aux travaux précédents, la complétude réfutationnelle de cette procédure n'est pas restreinte aux spécifications booléennes. La méthode est implémentée dans le prouveur SPIKE. Nous illustrons les avantages de SPIKE à l'aide de quelques problèmes en le comparant avec les prouveurs de théorèmes les plus connus: Nqthm et RRL.

Mots clés: Preuve automatique, Récurrence avec ensemble test, Récurrence simultanée, Réécriture conditionnelle.

Contents

1 Introduction

Nowadays, computer science is applied in an increasing numbers of safety critical systems. Consider for example, the supervision of nuclear power stations or anaesthetist control devices. Therefore, the safety of computer systems is vital. However, such systems may in general contain errors, and to ensure they work safely, we must use a battery of tests. Unless we can perform an infinity of tests, we take the risk of not detecting some mistakes which will be discovered in exceptional cases. The use of formal methods can be considered as a remedy! What are formal methods? Typically they use mathematically based notions.

In this context, equational reasoning plays a critical role in many computer science and artificial intelligence applications, in particular, in program verification and specification of systems. The use of equations is motivated by the existence of an initial model, and proof methods for this model are usually based on an induction scheme such as the one on the structure of terms. Many approaches have been developed to prove a theorem by induction. The first one applies explicit induction arguments on the structure of terms [Aubin, 1979; Boyer and Moore, 1979; Burstall, 1969; Boyer and Moore, 1988; Bundy *et al.*, 1989; Walther, 1993; Chadha and Plaisted, 1992]. The Nqthm system [Boyer and Moore, 1988] (**N**ew **q**uantified **th**eorem prover) was developed in this framework and is considered as one of the most powerful theorem provers. Many of the heuristics in Nqthm have been rationally reconstructed in the prover Clam [Bundy *et al.*, 1989; Bundy *et al.*, 1991]. RRL [Zhang *et al.*, 1988] (a **R**ewrite **R**ule **L**aboratory) is another theorem proving system that supports a *cover set* method which is closely related to Boyer and Moore's approach. Within the last decade, the proof by consistency approach which is based on rewriting and completion techniques has been developed [Musser, 1980] and refined in several ways in [Huet and Hullot, 1982; Jouannaud and Kounalis, 1986; Fribourg, 1986; Kapur *et al.*, 1986; Bachmair, 1988; Gramlich, 1989]. However, both approaches have many limitations. Indeed, guiding a proof by explicit induction requires some skill in finding the right axioms or hypotheses to apply. On the other hand, the proof by consistency technique does not require guidance from the user since the generation of lemmas is performed automatically through the completion procedure. However, the completion often misses *good* lemmas and fails where explicit induction succeeds. Moreover, the correctness of proof by consistency procedure requires either the convergence or the ground convergence property of the set of axioms. This is restrictive because convergence is often hard to achieve and ground convergence is known to be undecidable [Kapur *et al.*, 1990]. More recently, a new approach has been proposed which combines the full power of explicit induction and proof by consistency [Kounalis and Rusinowitch, 1990; Reddy, 1990; Bouhoula *et al.*, 1995; Bouhoula and Rusinowitch, 1993; Bouhoula and Rusinowitch, 1994;

Bouhoula, 1994d]. We have developed the system SPIKE [Bouhoula, 1994c] on this principle. SPIKE has proved several interesting theorems using a minimum of interaction with the user [Bouhoula, 1994b]. For instance, it has proved the Gilbreath Card Trick using two user lemmas [Bouhoula and Rusinowitch, 1994] while classical induction provers like COQ [Dowek *et al.*, 1991; Huet, 1991], Nqthm, and RRL require no less than fifteen lemmas! However, the SPIKE system is restrictive since the computation of test sets is done only if the constructors are free and the strategy is refutationally complete only with respect to boolean specifications. Note also that the set of false conjectures that can be refuted is very limited and the correctness proof of the procedure is long and delicate.

In this paper, we propose a general scheme for test set induction procedure. With this procedure, we describe a simple technique for proof of correctness. Our technique is easier and more general than the one given in [Reddy, 1990; Bouhoula and Rusinowitch, 1994]. This procedure relies on the notion of *cover set* and *test set* which can be seen as special induction schemes. Our definition of *test set* is more general than the previous one given in [Kounalis and Rusinowitch, 1990; Bouhoula and Rusinowitch, 1994]. This new definition together with a new notion of provable inconsistency and induction positions (i.e. which define the subset of variables of a conjecture that can be instanciated by induction schemes) permits us to refute more false conjectures than our previous definition [Kounalis and Rusinowitch, 1990; Bouhoula and Rusinowitch, 1994], in particular, if the axioms are not sufficiently complete. Previously, we could only compute a test set for a conditional specification if the constructors were free. Here, we give a procedure to compute them even if the constructors are not free. We give also an algorithm to compute all the induction positions. Finally, we present a procedure of proof by test set induction for a conditional specification which is refutationally complete for conditional specifications (not restricted to boolean specifications), in that it refutes any conjecture which is not an inductive theorem. Our approach does not need any hierarchy for managing the subgoals. This point is crucial for handling mutually recursive definitions. Recently, we have noted that our approach has some advantages concerning this problematic aspect of explicit induction techniques [Bouhoula, 1994a].

The paper is organised as follows. In Section 2, we introduce the basic notions about term rewriting and inductive theory. In section 3, we present a general scheme for the test set induction procedure and we give a simple technique for proving correctness. We define in section 4 a procedure of proof by test set induction for a conditional specification: Section 4.1 presents an algorithm for computing all induction positions of a given conditional specification, and section 4.2 introduces an algorithm for computing test sets if the constructors are not free. In section 4.3 we define a generalisation of our inductive rewriting given in [Bouhoula and Rusinowitch, 1993]. We give in Section 4.4 an infer-

ence system to perform induction and show its correctness. When the axioms are ground convergent and the functions are completely defined over free constructors, the system is proved refutationally complete (see Section 4.4.2). Computer experiments with SPIKE are discussed in Section 5. These examples show the superiority of SPIKE concerning mutual induction over explicit induction based systems such as Nqthm or RRL.

2 Terminology and Notation

We assume that the reader is familiar with the basic concepts of rewriting and induction. We introduce the notations used later and refer to [Dershowitz and Jouannaud, 1990; Padawitz, 1988] 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 be denoted by F. We assume that we have a partition of F into two subsets, the first one, C, contains the *constructor symbols* and the second, D, is the set of *defined symbols.* We say that a sort $\mathcal{S}$ is *finitary*, if there is no infinite set of ground terms of sort $\mathcal{S}$. Otherwise $\mathcal{S}$ is said to be *infinitary.*

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 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 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 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 symbol $\equiv$ is used for syntactic equality between two objects. The identity substitution will be denoted by $\mathcal{I}$. An 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 formula of the 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 pairwise terms of the same sort. An F-clause is a formula 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. 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 (there is no infinite sequence $t_1 \succ t_2 \succ \cdots$), monotonic ($s \succ t$ implies $w[s] \succ w[t]$) and stable ($s \succ t$ implies $s\sigma \succ t\sigma$). The multiset extension of $\succ$ will be denoted by $\gg$.

A conditional equation $a_1 = b_1 \wedge \cdots \wedge a_n = b_n \Rightarrow l = r$ will be written as $a_1 = b_1 \wedge \cdots \wedge a_n = b_n \Rightarrow l \rightarrow r$ if $\{l\sigma\} \gg \{r\sigma, a_1\sigma, b_1\sigma, \cdots, a_n\sigma, b_n\sigma\}$ for each substitution σ; 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. Let c be a constructor symbol. If for any rule $r \in R$, the top of the left-hand side of r is different to c, then we say that c is a free constructor. A rewrite rule $c \Rightarrow l \rightarrow r$ is *left linear* if l is linear. A rewrite system R is *left linear* if every rule in R is left linear, otherwise R is said to be *non-left linear*.

The *depth of a rewrite system R*, denoted *depth(R)*, is defined as the maximum of the depths of the left-hand sides of R. Similarly, the *strict depth* of R denoted by *sdepth(R)*, is the maximum of the depths of the strict positions in the left-hand sides of R. 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 that conditions hold. The conditions are checked recursively. Termination is ensured because the conditions are smaller (w.r.t. to $\succ$) than the left-hand side. A set of conditional rules is called a conditional rewrite system. Now we introduce the notion of term rewriting (w.r.t. $\succ$) with conditional rules:

Definition 2.1 (Conditional rewriting) *Let R be a set of conditional rules. 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 rule $\bigwedge_{i=1}^{n} a_i = b_i \Rightarrow l = r$ in R such that: for all $i \in [1 \cdots n]$ there exists c_i such that $a_i\sigma \rightarrow_R^* c_i$ and $b_i\sigma \rightarrow_R^* c_i$, where $\rightarrow_R^*$ denotes the reflexive and transitive closure of $\rightarrow_R$.*

A term t is *R-irreducible* if there is no term s such that $t \rightarrow_R s$. A term t is *strongly R-irreducible* if none of its non-variable subterms matches a left-hand side of R. We say that two terms s and t are joinable 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 $R \models u = v$. An operator $f \in D$ is *sufficiently complete* iff for all $t_1, \cdots, t_n$ in $T(C)$, there exists $t \in T(C)$ such that $f(t_1, \cdots, t_n) \rightarrow_R^* t$. If every $f \in D$ is sufficiently complete, then we say that R is sufficiently complete.

The case analysis used in this paper simplifies a conjecture with conditional rules where the disjunction of all conditions is inductively valid.

Definition 2.2 (Case analysis) *Let R be a set of conditional rules and let C be a clause. We define G as the set $\{< C[d\sigma]_u, P\sigma >|$ there exists $P \Rightarrow g \rightarrow d$ in R, a position u in C such that $C/u = g\sigma\}$. If $R \models_{ind} (\bigvee_{<C',P> \in G} P)$, then* case_analysis(C) $= \{P \Rightarrow C' | < C', P > \in G\}$.

where $\models_{ind}$ is written for inductive consequence w.r.t. the initial model.

3 Principle of Test Set Induction

To perform a proof by induction it is necessary to provide induction schemes. In our framework, these schemes are defined first by a function, which, given a conjecture, selects the positions of variables where the induction will be applied (induction variables), and second by a special set of terms called a *cover set* or a *test set* by which these variables are to be instantiated. Let us first consider the problem of choosing induction variables.

3.1 Selection of Induction Schemes

3.1.1 Induction variables

Consider a simple example to show that the problem of choosing the induction variables is fundamental for efficiency. Let $R = \{x+0 \rightarrow x,\ x+s(y) \rightarrow s(x+y)\}$. In order to prove $C \equiv x + y = y + x$, we instantiate C by induction schemes 0 and $s(x)$, and derive four lemmas: $0+0 = 0+0$, $0+s(x) = s(x)+0$, $s(x)+0 = 0+s(x)$ and $s(x)+s(y) = s(y)+s(x)$. However, only two are really necessary for proving C: $x+0 = 0+x$ and $x+s(y) = s(y)+x$. This means that only certain variables should be instantiated by induction schemes. We shall now define the set of these variables.

Definition 3.1 (Induction variables) *Let R be a conditional rewrite system and C be a term or a clause. The set of induction variables of C, noted by $ind\_var(C)$, is the smallest set such that: if x is a variable of a* finitary *sort or it appears in a non-variable subterm t of C at position u ($t(u) \equiv x$) and there exists a rule $\bigwedge_{i=1}^{n} g_i = d_i \Rightarrow g \rightarrow d$ in R such that t is unifiable with g and:*

i) u is a strict position of g or

ii) $g(u)$ is a non linear variable in g or

iii) $g(u) \in (\bigcup_{i=1,n}\{ind\_var(g_i),\ ind\_var(d_i)\})$.

Then $x \in ind\_var(C)$.

This recursive definition is correct. As R is a rewrite system, then for each rule $p \Rightarrow g \rightarrow d$ in R, the precondition p is smaller than the left-hand side g w.r.t a well founded ordering, and therefore *ind_var* is applied to terms which are smaller than C. Note that this definition is more general than the one given in [Bouhoula and Rusinowitch, 1994] and allows us to refute more false conjectures (see example 3.9), particularly, if the axioms are not sufficiently complete.

3.1.2 Cover sets and test sets

Our method is based on the notion of *cover sets* and *test sets.* Let us introduce first the following definition:

Definition 3.2 (Weakly irreducible term) *Let R be a conditional rewrite system and let t be a term. t is* weakly R-irreducible *if for all subterms s of t such that there exist a rule $p \Rightarrow g \rightarrow d$ in R and a substitution σ with $s = g\sigma$ then $p\sigma$ is unsatisfiable for R (i.e. for all ground substitution τ: $R \not\models p\sigma\tau$).*

Note that a strongly R-irreducible term is necessarily weakly R-irreducible.

Definition 3.3 (Cover set) *A* cover set, *denoted by CV, for a conditional rewrite system R is a finite set of R-irreducible terms such that for all ground R-irreducible term s, there exist a term t in CV and a ground substitution σ such that: $t\sigma \equiv s$.*

Cover sets are fundamental for the correctness of our method. However, they cannot help us to refute false conjectures. Therefore, we will introduce the notion of *test set.*

Definition 3.4 (Test set) *A set of terms TS is a* test set *for a conditional rewrite system R if TS is a* cover set *for R satisfying the following property: let t be a term and σ a TS-substitution [1] of t (which maps any induction variable of t by an element of TS),*

[1] In the following, a TS-substitution (resp. CV-substitution) will be denoted by ***test substitution*** (resp. ***cover substitution***)

if $t\sigma$ is weakly R-irreducible then there exists a ground substitution τ such that $t\sigma\tau$ is ground and R-irreducible.

Test sets can be considered as refined induction schemes since they allow us to avoid the failure of the procedure of proof by induction in some cases. This can be illustrated by the following example:

Example 3.5 *The following rules define* odd *and* even *for nonnegative integers:*

$$\begin{aligned} true &\neq false \\ even(0) &\rightarrow true \\ even(s(0)) &\rightarrow false \\ even(s(s(x))) &\rightarrow even(x) \\ even(x) = true \Rightarrow odd(x) &\rightarrow false \\ even(x) = false \Rightarrow odd(x) &\rightarrow true \end{aligned} \tag{1}$$

The conjecture $even(x) = true \vee odd(x) = true$ is valid in the initial model of R. A test set *of R is $\{0,\ s(0),\ s(s(x)),\ true,\ false\}$. The proof of the conjecture is immediate. The methods of proof by induction based on the notion of cover sets fails to prove this conjecture if we consider the cover set $\{0,\ s(x),\ true,\ false\}$.* ♦

Test sets also permit us to refute false conjectures by constructing a counterexample. We propose a new notion of provable inconsistency which allows us to refute more false conjectures than previous approaches [Kounalis and Rusinowitch, 1990; Bouhoula *et al.*, 1995; Bouhoula and Rusinowitch, 1994].

Definition 3.6 (Provably inconsistent) *Let R be a conditional rewrite system and TS a test set of R. Let $C \equiv \neg(a_1 = b_1) \vee \cdots \vee \neg(a_n = b_n) \vee (c_1 = d_1) \vee \cdots \vee (c_m = d_m)$ be a clause. C is provably inconsistent if and only if there exists a test substitution σ such that:*

i) for all i, $R \models_{ind} a_i\sigma = b_i\sigma$.

ii) for all j, $c_j\sigma \not\equiv d_j\sigma$ and the maximal element of $\{c_j\sigma,\ d_j\sigma\}$ w.r.t. $\prec$ is weakly R-irreducible.

If C is *provably inconsistent* and the axioms are ground convergent, then C is not an inductive consequence of R. This result can be proved by the construction of a ground substitution which gives a counterexample.

Theorem 3.7 *Let R be a conditional ground convergent rewrite system. If a clause C is* provably inconsistent, *then C is not an inductive consequence of R.*

Proof: Let $C \equiv \neg(a_1 = b_1) \vee \cdots \vee \neg(a_n = b_n) \vee (c_1 = d_1) \vee \cdots \vee (c_m = d_m)$ be a *provably inconsistent* clause. Then there exists a test substitution σ of C such that for all i, $R \models_{ind} a_i\sigma = b_i\sigma$ and for all j, $c_j\sigma \not\equiv d_j\sigma$. Let $\mathcal{I} = \cup_{j=1}^{m} S_j$ where S_j is the set of maximal terms in $\{c_j\sigma,\ d_j\sigma\}$ w.r.t. $\prec$. Every element in $\mathcal{I}$ is weakly R-irreducible by hypothesis. To show that C is not an inductive consequence of R, it is sufficient to prove that $(c_1 = d_1 \vee \cdots \vee c_m = d_m)\sigma$ is not an inductive consequence of R, since no ground instance of $(\neg(a_1 = b_1) \vee \cdots \vee \neg(a_n = b_n))\sigma$ is an inductive consequence of R. Suppose that $\mathcal{I} = \{g'_1, \cdots, g'_k\}$ and let A be the term $f(g'_1, \cdots, g'_k)$ where f is a new function symbol of arity k. $A\sigma$ is weakly R-irreducible, then by definition 3.4, there exists a ground substitution τ such that the term $A\sigma\tau$ is ground and R-irreducible. On the other hand R is ground convergent. Therefore $C\sigma\tau$ is not an inductive consequence of R. □

Our notion of provably inconsistent allows us to refute more false conjectures than previous methods [Kounalis and Rusinowitch, 1990; Bouhoula *et al.*, 1995; Bouhoula and Rusinowitch, 1994]. In particular, we can now refute false conjectures even when the axioms are not sufficiently complete.

Example 3.8 *Consider the following conditional specification which defines the predicate $\leq$ on the natural numbers and the predicate* ordered *which checks whether a list is ordered.*

$$\begin{aligned} true &\neq false \\ 0 \leq x &\rightarrow true \\ s(x) \leq 0 &\rightarrow false \\ s(x) \leq s(y) &\rightarrow x \leq y \\ ordered(cons(x, nil)) &\rightarrow true \\ x \leq y = false \Rightarrow ordered(cons(x, cons(y, z))) &\rightarrow false \end{aligned}$$

The predicate ordered *is not sufficiently complete and the equation*

$$ordered(cons(0, cons(0, y))) = false$$

is provably inconsistent. *Indeed, $ordered(cons(0, cons(0, y)))$ does not contain any induction variable and is* weakly R-irreducible *since $0 \leq 0 = false$ is unsatisfiable in R. With the methods [Kounalis and Rusinowitch, 1990; Bouhoula* et al., *1995; Bouhoula and Rusinowitch, 1994], this equation is not* provably inconsistent *since it contains an instance of a left-hand side of R.* ♦

Example 3.9 *Consider example 3.5 and remove rule 1, then the predicate* odd *is not sufficiently complete. Consider the conjecture* $odd(x) = true$, x *is an induction variable w.r.t. definition 3.1. Instantiating* x *by* 0 *yields* $odd(0) = true$ *which is simplified by* R *into:* $false = true$, *which is* provably inconsistent. *We conclude that* $odd(x) = true$ *is not an inductive consequence of* R.

Now, with the method [Bouhoula and Rusinowitch, 1994], x *is not an induction variable, on the other hand* $odd(x) = true$ *is not provably inconsistent since* $odd(x)$ *is strongly reducible. Therefore, the method [Bouhoula and Rusinowitch, 1994] fails to refute the conjecture* $odd(x) = true$. ♦

3.2 A General Technique of Proof of the Correctness of an Inference System

To present the procedure of proof by induction, we use a formalism of inference rules as Bachmair [Bachmair, 1988] and Reddy [Reddy, 1990]. The inference system I uses the following data structure (E, H), where E and H are two sets of clauses, E contains the conjectures to be checked, and H contains clauses, previously in E, that can be reduced and therefore used as inductive hypotheses. The sets E and H are constructed incrementally by the inference system I. The rewrite system R is considered as global component and therefore does not appear in the inference system since it does not change during the proof.

An I-derivation is a sequence of states:

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

3.2.1 Correctness

We present a general technique to prove the correctness of the inference system I. It is obtained by defining a well-founded ordering on clauses noted by $\prec_c$ and a notion of *fair derivation.* Fairness roughly means that every clause in the set of conjectures will be eventually modified by some inference. More formally: A derivation $(E_0, H_0) \vdash_I (E_1, H_1) \vdash_I \cdots$ is fair if the set of persistent clauses $(\cup_i \cap_{j \geq i} E_j)$ is empty. Now, given a fair derivation, we reason by contradiction to prove the correctness of I: if a non valid clause is generated in the 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 is a contradiction with the fairness hypothesis.

Definition 3.10 (Correct inference system) *Let R be a rewrite system and $(E_0, \emptyset) \vdash_I (E_1, H_1) \vdash_I \cdots$ be an I-derivation. Suppose that $R \not\models_{ind} E_0$ and let C' be a minimal element w.r.t. $\prec_c$, of the set $\mathcal{F} = \{D\theta \mid D \in \cup_i E_i$ and θ is a ground R-irreducible substitution such that $R \not\models_{ind} D\theta\}$. Then there exists $C \in \cup_i E_i$ minimum w.r.t. $\prec_c$ and a ground R-irreducible substitution σ such that $C' \equiv C\sigma$. The inference system I is* correct *if no rules can be applied to C.*

The next theorem expresses that if an inference system I is correct and the derivation is fair then the conjectures of E_0 are inductive consequences of R.

Theorem 3.11 (Correctness) *Let R be a rewrite system and $(E_0, \emptyset) \vdash_I (E_1, H_1) \vdash_I \cdots$ a fair I-derivation. If I is correct, then we have $R \models_{ind} E_0$.*

Let I be a correct inference system. Every I-derivation from the state $(E, \emptyset)$ to $(\emptyset, H)$ where E and H are two sets of clauses, is fair. Therefore, the conjectures of E are inductive consequences of R. Note also that if $(E_0, \emptyset) \vdash_I (E_1, H_1) \vdash_I \cdots$ is an infinite fair I-derivation, then the clauses of E_0 are inductive consequences of R.

3.2.2 Refutation of conjectures

We present now our technique to prove that the inference system I is refutationally correct. Suppose that R is a ground convergent rewrite system. We add to I the following rule:

refute: $(E \cup \{C\}, H) \vdash_I$ *Refutation*
if C is provably inconsistent.

The rule *refute* permits us to detect false conjectures. This result is a consequence of the theorem 3.7 and can be expressed by the following corollary:

Corollary 3.12 *Let $(E_0, \emptyset) \vdash_I (E_1, H_1) \vdash_I \cdots$ an I-derivation. If R is a ground convergent rewrite system and there exists $k \geq 0$ such that the rule* refute *is applied to (E_k, H_k) then $R \not\models_{ind} E_k$.*

A refutationally correct inference system satisfy the following property: If at step k, we find that $R \not\models_{ind} E_k$, then E_0 is not valid either in R. Formally:

Definition 3.13 (Refutationally correct inference system) *Let R be a rewrite system and let $(E_i, H_i) \vdash_I (E_{i+1}, H_{i+1})$ a derivation step. The inference system I is refutationally correct if $H_0 = \emptyset$ and $\forall j \leq i$ $R \models_{ind} E_j$ implies $R \models_{ind} E_{i+1}$.*

Suppose that I is refutationally correct. and let $(E_0, \emptyset) \vdash_I (E_1, H_1) \vdash_I \cdots$ an I-derivation. If there exists k such that the rule *refute* is applied to (E_k, H_k), then the conjectures of E_0 are not inductive consequences of R. This result is expressed by the following theorem:

Theorem 3.14 *Let R be a ground convergent rewrite system and I a refutationally correct system. Let $(E_0, \emptyset) \vdash_I (E_1, H_1) \vdash_I \cdots$ be an I-derivation. If there exists j such that the rule* refute *is applied to (E_j, H_j) then $R \not\models_{ind} E_0$.*

Proof: Let $(E_0, \emptyset) \vdash_I (E_1, H_1) \vdash_I \cdots$ an I-derivation. Suppose that there exists j such that the rule *refute* is applied to (E_j, H_j). From Corollary 3.12 we conclude that $R \not\models_{ind} E_j$. Now since I is refutationally correct, we deduce that $R \not\models_{ind} E_0$. □

3.3 A Generic Procedure for Test Set Induction

Let R be a conditional rewrite system and CV a cover set for R. Our generic procedure formalised as a transition system presented in figure 1: *generate* derives lemmas, *simplify* and *delete* eliminate redundancies, *refute* witnesses inconsistencies. In these rules, $\approx_c$ is a stable congruence on clauses compatible with $\succ_c$.

The correctness of the inference system I is expressed by the following lemma.

Lemma 3.15 *The inference system I is correct w.r.t. definition 3.10.*

Proof: Let R be a rewrite system and let $(E_0, \emptyset) \vdash_I (E_1, H_1) \vdash_I \cdots$ be an I-derivation. Suppose that $R \not\models_{ind} E_0$ and let C' be a minimal element w.r.t. $\prec_c$, of the set $\mathcal{F} = \{D\theta \mid D \in \cup_i E_i$ and θ is a ground R-irreducible substitution such that $R \not\models_{ind} D\theta\}$. Then there exists $C \in \cup_i E_i$ minimum w.r.t. $\prec_c$ and a ground R-irreducible substitution σ such that $C' \equiv C\sigma$. We show now that whatever rule is applied to C, we obtain a contradiction.

1. Suppose that the rule *generate* is applied to C. As the substitution σ is ground and R-irreducible, there exists σ_0 a cover substitution of C and a ground substitution τ such that: $\sigma = \sigma_0\tau$. Therefore, there exists a clause $C_k \in \cup_i E_i$ such that:

$$(R \cup \{S\theta \mid S \in E \cup H \cup \{C\} \text{ and } S\theta \prec_c C\sigma\}) \models_{ind} (C\sigma \Leftrightarrow C_k\tau)$$

On the other hand, $R \models_{ind} \{S\theta \mid S \in E \cup H \cup \{C\}$ *and* $S\theta \prec_c C\sigma\}$, since $C\sigma$ is minimum w.r.t. $\prec_c$ in $\mathcal{F}$. Then, $R \not\models_{ind} C_k\tau$ and $C_k\tau \prec_c C\sigma$, contradiction, since we have proved the existence of an instance of a clause of $\cup_i E_i$ which is not valid and smaller than $C\sigma$ w.r.t. $\prec_c$.

generate: $(E \cup \{C\}, H) \vdash_I (E \cup \{C_1, \cdots, C_n\}, H \cup \{C\})$

if $\forall \sigma$ cover substitution of C and $\forall \tau$ ground substitution, $\exists k$ such that:
$(R \cup \{S\theta \mid S \in E \cup H \cup \{C\} \text{ and } S\theta \prec_c C\sigma\tau\}) \models_{ind} (C\sigma\tau \Leftrightarrow C_k\tau)$
and $C_k\tau \prec_c C\sigma\tau$.

simplify: $(E \cup \{C\}, H) \vdash_I (E \cup \{C_1, \cdots, C_n\}, H)$

if $\forall \sigma$ ground substitution, $\exists k$ such that:
$(R \cup \{S\theta \mid S \in E \text{ and } S\theta \prec_c C\sigma\} \cup \{S\theta \mid S \in H \text{ and } S\theta \preceq_c C\sigma\}) \models_{ind} (C\sigma \Leftrightarrow C_k\sigma)$
and $C_k\sigma \prec_c C\sigma$.

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

if $\forall \sigma$ ground substitution :
$(R \cup \{S\theta \mid S \in E \text{ and } S\theta \prec_c C\sigma\} \cup \{S\theta \mid S \in H \text{ and } S\theta \preceq_c C\sigma\}) \models_{ind} C\sigma$.

refute: $(E \cup \{C\}, H) \vdash_I \mathit{Refutation}$

if C is provably inconsistent.

Figure 1 Inference System I

2. Suppose that the rule *simplify* is applied to C. Then there exists a clause $C_k \in \cup_i E_i$ such that:

$$(R \cup \{S\theta \mid S \in E \textit{ and } S\theta \prec_c C\sigma\} \cup \{S\theta \mid S \in H \textit{ and } S\theta \preceq_c C\sigma\}) \models_{ind} (C\sigma \Leftrightarrow C_k\sigma)$$

On the other hand, $R \models_{ind} \{S\theta \mid S \in E \cup H \textit{ and } S\theta \prec_c C\sigma\}$. Suppose there exists $S' \in H$ such that $R \not\models_{ind} S'\theta$ and $S'\theta \approx_c C\sigma$. Then, the clause $S'\theta$ is also minimal w.r.t. $\prec_c$ in $\mathcal{F}$. Or the presence of S' in H proves that the rule *generate* has been applied to S' in contradiction with a previous case. Therefore, $R \not\models_{ind} C_k\sigma$ and $C_k\sigma \prec_c C\sigma$, contradiction.

3. Suppose that the rule *delete* is applied to C. Then we have:

$$(R \cup \{S\theta \mid S \in E \textit{ and } S\theta \prec_c C\sigma\} \cup \{S\theta \mid S \in H \textit{ and } S\theta \preceq_c C\sigma\}) \models_{ind} C\sigma$$

We follow the same reasoning as in 2, and conclude that:

$$R \models_{ind} \{S\theta \mid S \in E \textit{ and } S\theta \prec_c C\sigma\} \cup \{S\theta \mid S \in H \textit{ and } S\theta \preceq_c C\sigma\}$$

Contradiction since $R \not\models_{ind} C\sigma$. □

If R is a ground convergent rewrite system, then the inference system I can refute false conjectures. This result is a consequence to the following lemma:

Lemma 3.16 *The inference system I is refutationally correct w.r.t. definition 3.13.*

Proof: Let C be a clause in E_j and $(E_j, H_j) \vdash_I (E_{j+1}, H_{j+1})$ a derivation step obtained by the application of a rule to C. Let us show that $R \models_{ind} E_{j+1}$ if $H_0 = \emptyset$ and for all $i \leq j$ we have $R \models_{ind} E_i$. We will anlyse the situation according to the applied rule to C.

1. Suppose that the rule *generate* is applied to C with the *cover substitution* σ. Let τ be a ground substitution. According to the hypotheses, we have:

$$R \models_{ind} \{S\theta \mid S \in E \cup H \cup \{C\} \text{ and } S\theta \prec_c C\sigma\tau\}$$

   On the other hand $R \models_{ind} C\sigma\tau$ and therefore the *generate* rule generate only valid conjectures.

2. Suppose that the rule *simplify* is applied to C. Let σ be a ground substitution. According to the hypotheses, we have:

$$R \models_{ind} \{S\theta \mid S \in E \text{ and } S\theta \prec_c C\sigma\} \cup \{S\theta \mid S \in H \text{ and } S\theta \preceq_c C\sigma\}$$

   On the other hand $R \models_{ind} C\sigma\tau$ and therefore the *simplify* rule generate only valid conjectures.

3. Suppose that the rule *delete* is applied to C, then $R \models_{ind} E_{j+1}$ since $E_{j+1} \subseteq E_j$ in this case. □

The inference system I can be combined with other proof techniques which permit us to increase the class of conjectures to be proved. This claim is justified by the correctness of the inference system I when we add the rule *induction*:

induction: $(E, H) \vdash_I (E, H \cup \{C\})$
if $R \models_{ind} C$

where the clause C is proved by another method. This rule can avoid the divergence of the procedure in some cases since it permit us to add some lemmas during the proof.

Now we will present an instance of the generic procedure which, as opposed to the methods given in [Bouhoula and Rusinowitch, 1994; Bouhoula, 1994d], is refutationally complete even when we have a non boolean specification.

4 Test Set Induction in Conditional Theories

In this section we propose a procedure of proof by test set induction for a conditional specification. We first present algorithms to compute *induction variables* and *test sets*. Then, we define a generalisation of our inductive rewriting given in [Bouhoula and Rusinowitch, 1993]. Finally, we introduce our procedure of proof by induction and we prove its correctness and refutational completeness.

4.1 How to Compute Induction Variables

Given a rewrite system R, we start by the computation of induction positions of function symbols. This computation is done only once and permit us to determine whether a variable of a term t is an induction variable or not. Let f be a function symbol, we say that t is in $Def(f)$, if t is of the form $f(w_1, \cdots, w_n)$ such that for all i, $w_i \in T(F, X)$. The algorithm presented in figure 2 computes for all functions $f \in F$, the set of induction positions of f, denoted by $ind\_pos(f)$. We can easily prove that the algorithm terminates since there are only a finite number of rules in R.

Proposition 4.1 *Given a term t, a variable x of t of sort s is an* induction variable *of t w.r.t. definition 3.1, if s is* finitary, *or x appears in a subterm $h(\vec{s})$ of t at position u ($h(\vec{s})/u \equiv x$), and u is an induction position of h. If t is a variable, then it is considered as an* induction variable.

4.2 How to Compute Test Sets

The computation of test sets for equational specifications is decidable (see [Kounalis, 1990; Hofbauer and Huber, 1994]). Unfortunately, no algorithm exists for the general case of conditional specifications. However, in [Kounalis and Rusinowitch, 1990; Bouhoula and Rusinowitch, 1994; Bouhoula, 1994d] some methods are described for computing test sets for conditional specifications over *free set of constructors*. In this section we present a procedure to compute test sets even if the constructors are not free. Let us introduce first the following notions: We say that a term t is *infinitary* if for any position u in t with t/u non-ground, there exists infinitely many R-irreducible ground instances of t whose subterms at position u are distinct. The *bound* for R, denoted $D(R)$, is equal to $depth(R) - 1$ if $sdepth(R) < depth(R)$ and R is left linear, $depth(R)$ otherwise.

Proposition 4.2 *Let R be a conditional rewrite system and TS' a finite set of R-irreducible terms such that:*

For all $f \in F$:

- S_f := $\{s \mid$ there exists $\bigwedge_i u_i = v_i \Rightarrow g \rightarrow d \in R$ and s is an element of $\{u_i\}_i \cup \{v_i\}_i \cup \{g\}$ such that $s \in Def(f)\}$
- $pos_0(f)$:= $\{u \mid \exists s \in S_f$ and $u \in (socc(s) \setminus \{\varepsilon\})$ or $s(u) \notin var\_lin(s)\}$

i := 0; *saturation* := *false*;

While $\neg$ *saturation* **do**

- **For all** $f \in F$:

  For all $p \Rightarrow g[x]_u \rightarrow d$ in R such that $g \in Def(f)$ and $x \in var\_lin(g)$:

  - E_{hv} := $\{(h, v) \mid \exists$ a subterm s of p that contains x at the position v and $s \in Def(h)\}$
  - If there exists $(h, v) \in E_{hv}$ such that $v \in pos_i(h)$

    then $pos_{i+1}(f)$:= $pos_i(f) \cup \{u\}$

    else $pos_{i+1}(f)$:= $pos_i(f)$

- **If** for all $f \in F$: $pos_{i+1}(f) = pos_i(f)$

  then for all $f \in F$: $pos\_rec(R, f) = pos_i(f)$; *saturation* := *true*

  else $i := i + 1$

End.

Figure 2 Computing induction positions

1. *for any R-irreducible ground term s, there exist a term in TS′ and a ground substitution σ such that: tσ ≡ s;*
2. *any non ground term in TS′ contains only variables at depth greater than or equal to D(R);*
3. *let t be a non ground term in TS′. If R is left linear then there exists a R-irreducible instance of t, otherwise t is infinitary.*

Then TS′ is a test set of R w.r.t. definition 3.4.

Proof: Let t be a term and σ a TS'-substitution of t. Suppose that $t\sigma$ is weakly irreducible by R. Let $var(t\sigma) = \{x_1, \cdots, x_k\}$ (recall that for all $i \in [1 \cdots k]$, the sort of x_i is infinitary) and consider a ground substitution ϕ such that $\sigma\phi$ is R-irreducible and if R is not *left*

linear, we have also:

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

Such a substitution instance exists by using clause 3 of the proposition 4.2. Let us show that $t\sigma\phi$ is R-irreducible:

Assume that there exists a rule $p \Rightarrow g \rightarrow d$ in R and a substitution α such that $g\alpha$ is a subterm of $t\sigma\phi$ and $R \models p\alpha$. Since $\sigma\phi$ is R-irreducible, there is a strict position u in t such that $t\sigma\phi/u$ is an instance of g. Let v be a non-variable position of g. v is a non-variable position of $t\sigma/u$. Otherwise, there are two cases to consider:

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

So necessarily v is a non-variable position of $t\sigma/u$. Now, we show that there exists a substitution β such that $t\sigma/u \equiv g\beta$. We consider two cases:

a.1) Assume that g is linear. We can define a substitution β such that for every variable x that occurs at position w of g, we have $x\beta \equiv x\alpha$ if $x\alpha$ appears in $t\sigma$, otherwise $x\beta \equiv t\sigma/uw$. Note that such substitution exists since g is linear. Then we have $t\sigma/u \equiv g\beta$.

a.2) Assume that g is not linear. We can also consider the same substitution β defined in a.1. Otherise there exists two positions u_1 and u_2 of a variable x in g such that: $t\sigma/uu_1 \not\equiv t\sigma/uu_2$ and $t\sigma\phi/uu_1 \equiv t\sigma\phi/uu_2$. There are three cases to be considered:

a.2.1) if $t\sigma/uu_1$ and $t\sigma/uu_2$ are ground. In this case $t\sigma/uu_1 \equiv t\sigma\phi/uu_1$ and $t\sigma/uu_2 \equiv t\sigma\phi/uu_2$ and therefore $t\sigma/uu_1 \equiv t\sigma/uu_2$, contradiction.

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

a.2.3) if $t\sigma/uu_1$ and $t\sigma/uu_2$ are non-ground, then there exists a position v and a variable x_k such that $t\sigma/uu_1v \equiv x_k$ and $t\sigma/uu_2v \not\equiv x_k$.

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

Therefore, $t\sigma/u \equiv g\beta$ and $\alpha \equiv \beta\lambda$. Since $t\sigma$ is weakly R-irreducible then $R \not\models p\alpha$, contradiction. Finally, we conclude that $t\sigma\phi$ is R-irreducible. □

The following proposition gives us a procedure to compute a test set if the constructors are not free. We first recall that a term t is inductively reducible by a rewrite system 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 [Kaplan and Choquer, 1986].

Proposition 4.3 *Let R be a conditional rewrite systems. 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 TS 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 TS. The second property of proposition 4.2 is trivially verified by construction. Let us check the third property of proposition 4.2. Any non-ground term t in TS has at least one ground instance which is R-irreducible since t is not inductively reducible by R. □

If the constructors are specified by a set of unconditional equations, then we can decide inductive reducibility of constructor terms and Therefore, we obtain an algorithm to compute test sets.

Example 4.4 ([Kaplan, 1984]) *Let R be the set of conditional rules:*

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

Note that the constructors s and p are not free, the test set here is:

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

4.3 Inductive Rewriting

To simplify goals, we generalize the inductive rewriting relation defined in [Bouhoula and Rusinowitch, 1993], so that we can use, as well as axioms, induction hypothesis and other conjectures not necessary proved during the simplification. 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. In the following, $\approx$ is a stable congruence on terms compatible with $\succ$. The complexity of an equation $g = h$ is defined as in [Bouhoula and Rusinowitch, 1993]:

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

where the new symbol $\perp$ 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 $\bigwedge_i a_i = b_i \Rightarrow \bigvee_j c_j = d_j$. 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 there exists $\tau \not\equiv \mathcal{I}$ such that $C_2 \equiv C_1\tau$ or $nln(C_1) < nln(C_2)$, where $nln(C)$ is the number of negative literals of C.

Definition 4.5 *Let R be a set of conditional rules and W a set of conditional equations. Consider a clause $C \equiv (a = b)^\epsilon \vee r$ and its skolemized version $\overline{C} \equiv (\overline{a} = \overline{b})^\epsilon \vee \overline{r}$. We write:*

$$a \overset{C;\tau}{\longmapsto}_{R<W>} a'$$

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 $\mathcal{R} \equiv \bigwedge_{i=1}^n a_i = b_i \Rightarrow s = t$ 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* $a_i\sigma \overset{C;\tau}{\longmapsto}{}^*_{R<W>} c_i'$ *and* $b_i\sigma \overset{C;\tau}{\longmapsto}{}^*_{R<W>} d_i'$ *and* $\overline{c_i'} =_{prem(\overline{r})} \overline{d_i'}$.

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

Definition 4.6 (Inductive rewriting) *Let R be a set of conditional rules and W a set of conditional equations. Consider a clause $C \equiv (a = b)^\epsilon \vee r$ and its skolemized version $\overline{C} \equiv (\overline{a} = \overline{b})^\epsilon \vee \overline{r}$. We write: $C[a] \mapsto_{R<W>} C[a']$ if and only if $a \overset{C;\tau}{\longmapsto}_{R<W>} a'$ and $C[a'] \prec_c C[a]$.*

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

Example 4.7 *Consider a specification with the only axiom $s(s(0)) = 0$, then the proposition $s(s(x)) = x$ is an inductive property. The methods of [Reddy, 1990; Bouhoula and Rusinowitch, 1994] fail to prove this conjecture if we consider the cover set $\{0,\ s(x)\}$. Indeed, the instantiation of x by $s(y)$ give us the equation $s(s(s(y))) = s(y)$ which cannot be simplified by the axiom. Now, thanks to the new inductive rewriting, $s(s(s(y))) = s(y)$ can be simplified into $s(y) = s(y)$, using the conjecture $s(s(x)) = x$, since $s(s(y)) = y \prec_c s(s(s(y))) = s(y)$.* ♦

Inductive rewriting is stable by substitution:

Lemma 4.8 *For all substitution* τ*:* $C \mapsto_{R<W>} C'$ *implies* $C\tau \mapsto_{R<W>} C'\tau$.

The proof is trivial.

4.4 A proof procedure for Conditional Theories

generate: $(E \cup \{C\}, H) \vdash_J (E \cup (\cup_\sigma E_\sigma), H \cup \{C\})$
if for all cover substitution σ of C:
either $C\sigma$ is a tautology and $E_\sigma = \emptyset$
or $C\sigma \mapsto_{R<H \cup E \cup \{C\}>} C'$ and $E_\sigma = \{C'\}$
otherwise $E_\sigma = case\_analysis(C\sigma)$.

case simplify: $(E \cup \{C\}, H) \vdash_J (E \cup E', H)$
if $E' = case\_analysis(C)$.

simplify: $(E \cup \{C\}, H) \vdash_J (E \cup \{C'\}, H)$
if $C \mapsto_{R<H \cup E>} C'$

subsume: $(E \cup \{C\}, H) \vdash_J (E, H)$
if C is subsumed by another clause of $R \cup H \cup E$.

delete tautology: $(E \cup \{C\}, H) \vdash_J (E, H)$
if C is a tautology.

Figure 3 Inference System J

Let R be a conditional rewrite system and CV a cover set for R. Our procedure is defined by a set of transition rules (see figure 3). This procedure is a generalisation and an extension of our previous procedures [Bouhoula and Rusinowitch, 1993; Bouhoula and Rusinowitch, 1994; Bouhoula, 1994d]. The *generate* rule allows us to derive lemmas. The *case simplify* rule simplifies a conjecture with conditional rules where the disjunction of all conditions is inductively valid. The *simplify* rule reduces a clause C with axioms from R, induction hypotheses from H, and other conjectures which are not yet proved. Note that

simplify permits mutual simplification of conjectures. This rule implements simultaneous induction and is crucial for efficiency. The rules *subsume* and *delete tautology* delete redundant clauses.

4.4.1 Correctness

The correctness of the inference system J is expressed by the following lemma.

Lemma 4.9 *The inference system J is correct w.r.t. definition 3.10.*

Proof: Let R be a rewrite system and let $(E_0, \emptyset) \vdash_J (E_1, H_1) \vdash_J \cdots$ be a J-derivation. Suppose that $R \not\models_{ind} E_0$ and let C' be a minimal element w.r.t. $\prec_c$, of the set $\mathcal{F} = \{D\theta \mid D \in \cup_i E_i$ and θ is a ground R-irreducible substitution θ such that $R \not\models_{ind} D\theta\}$. Then there exists $C \in \cup_i E_i$ minimum w.r.t. $\prec_c$ and a ground R-irreducible substitution σ such that $C' \equiv C\sigma$. We show whatever rule is applied to C, a contradiction is obtained.

Generate: Suppose that the rule *generate* is applied to C. The substitution σ is ground and R-irreducible then there exists σ_0 a cover substitution of C and a ground substitution τ such that: $\sigma = \sigma_0\tau$. $C\sigma$ cannot be a tautology, we have then two possibilities:

1) If there exists a clause C' such that $C\sigma_0 \mapsto_{R<H \cup E \cup \{C\}>} C'$ [2] then by Lemma 4.8, we have $C\sigma \mapsto_{R<H \cup E \cup \{C\}>} C'\tau$. The instances of clauses of $H \cup E \cup \{C\}$ used in the rewriting step are smaller than $C\sigma$ w.r.t. $\prec_c$ and Therefore, there are valid in R. The premisses of $C\sigma$ are also valid since $R \not\models_{ind} C\sigma$. Then, $R \not\models_{ind} C'\tau$. On the other hand, we have $C'\tau \prec_c C\sigma$ and $C' \in \cup_i E_i$, contradiction, since we have proved the existence of an instance of a clause of $\cup_i E_i$ which is not valid and smaller than $C\sigma$ w.r.t. $\prec_c$.

2) Assume that the rule case_analysis is applied to $C\sigma_0$, then there exists a non-empty sequences of conditional rules:

$$P_1 \Rightarrow g_1 \rightarrow d_1,\ P_2 \Rightarrow g_2 \rightarrow d_2,\ \cdots\ P_n \Rightarrow g_n \rightarrow d_n \in R$$

and a sequence of positions $u_1, u_2, \cdots, u_n$ in $C\sigma_0$ such that:

$$C\sigma_0/u_1 = g_1\theta_1,\ C\sigma_0/u_2 = g_2\theta_2,\ \cdots, C\sigma_0/u_n = g_n\theta_n$$

and $R \models_{ind} P_1\theta_1 \vee P_2\theta_2 \vee \cdots \vee P_n\theta_n$. The result of applying the case_analysis is:

$$\{P_1\theta_1 \Rightarrow C\sigma_0[d_1\theta_1]_{u_1}, \cdots, P_n\theta_n \Rightarrow C\sigma_0[d_n\theta_n]_{u_n}\}$$

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

Then there exists k such that $R \models_{ind} P_k\theta_k$. Let $C_k \equiv P_k\theta_k \Rightarrow C\sigma_0[d_k\theta_k]_{u_k}$, we have $R \not\models_{ind} C_k\tau$ since $R \models_{ind} P_k\theta_k\tau$, $R \models_{ind} g_k\theta_k\tau = d_k\theta_k\tau$ and $R \not\models_{ind} C\sigma$. On the other hand, $P_k\theta_k\tau \ll \{g_k\theta_k\tau\}$ [3] and $d_k\theta_k\tau \prec g_k\theta_k\tau$ since $P_k \Rightarrow g_k \rightarrow d_k$ is a rewrite rule so $C_k\tau \prec_c C\sigma$. Contradiction.

Case simplify: This case is similar to the previous one.

Simplify: Suppose that the rule *simplify* is applied to C, then there exists a clause C' such that $C \mapsto_{R<H\cup E>} C'$, by the Lemma 4.8, we have $C\sigma \mapsto_{R<H\cup E>} C'\sigma$. The instances of clauses of $H \cup E$ used in the rewriting step are smaller than $C\sigma$ w.r.t. $\prec_c$ and Therefore, there are valid in R. Hence, $R \not\models_{ind} C'\sigma$. On the other hand, we have $C'\sigma \prec_c C\sigma$ and $C' \in \cup_i E_i$, contradiction.

Subsume: Since $R \not\models_{ind} C\sigma$, C cannot be subsumed by an axiom of R. If there exists $C' \in H \cup (E \setminus \{C\})$ such that $C \equiv C'\tau \vee r$, we have $R \not\models_{ind} C'\tau\sigma$, then, $r = \emptyset$ and $\tau = \mathcal{I}$ since C is minimal in $\cup_i E_i$ w.r.t. $\prec_c$. Therefore, $C' \notin (E \setminus \{C\})$. On the other hand, $C' \notin H$, otherwise the rule *generate* can be also be applied to C, in contradiction to a previous case. Hence, this rule cannot be applied to C.

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

If R is a ground convergent rewrite system, the inference system J can refute false conjectures. This result is a consequence to the following lemma:

Lemma 4.10 *The inference system J is refutationally correct w.r.t. definition 3.13.*

Proof: Let C be a clause in E_j and $(E_j, H_j) \vdash_I (E_{j+1}, H_{j+1})$ by the application of an inference rule on C. Let us show that $R \models_{ind} E_{j+1}$ if $H_0 = \emptyset$ and for all $i \leq j$ we have $R \models_{ind} E_i$. We will discuss the situation according to the rule which is applied to C:

Generate: Suppose that the rule *generate* is applied to C with the *cover substitution* σ. If $C\sigma$ is not a tautology, there are two possibilities:

1) If there exists C' such that $C\sigma \mapsto_{R<H\cup E\cup\{C\}>} C'$. Let τ be a ground substitution, by the Lemma 4.8, we have $C\sigma\tau \mapsto_{R<H\cup E\cup\{C\}>} C'\tau$. By hypothesis, the instances of clauses

[3] Suppose that $P_k \equiv \bigwedge_{i=1}^{n} u_i = v_i$. Abusing notation, the set $\{u_1\theta_k\tau,\ v_1\theta_k\tau, \cdots, u_n\theta_k\tau,\ v_n\theta_k\tau\}$ will be noted by $P_k\theta_k\tau$.

of $H \cup E \cup \{C\}$ which are used during the rewriting step, are valid. On the other hand, we can assume that the premisses of C are valid (otherwise the proof is obvious). Hence $R \models_{ind} C'\tau$ and Therefore, $R \models_{ind} C'$.

2) If case_analysis is applied to $C\sigma$, then there exists a non-empty sequence of conditional rules:

$$P_1 \Rightarrow g_1 \rightarrow d_1,\ P_2 \Rightarrow g_2 \rightarrow d_2,\ \cdots\ P_n \Rightarrow g_n \rightarrow d_n \in R$$

and a sequence of positions $u_1, u_2, \cdots, u_n$ in $C\sigma$ such that:

$$C\sigma/u_1 = g_1\theta_1,\ C\sigma/u_2 = g_2\theta_2,\ \cdots, C\sigma/u_n = g_n\theta_n$$

and $R \models_{ind} P_1\theta_1 \vee P_2\theta_2 \vee \cdots \vee P_n\theta_n$. The result of the application of case_analysis is:

$$\{P_1\theta_1 \Rightarrow C\sigma[d_1\theta_1]_{u_1}, \cdots, P_n\theta_n \Rightarrow C\sigma[d_n\theta_n]_{u_n}\}$$

Suppose that there exists k such that $R \not\models_{ind} C_k \equiv P_k\theta_k \Rightarrow C\sigma[d_k\theta_k]_{u_k}$. Then there exists a ground substitution τ (we can assume that $C\sigma\tau$ is ground without loss of generality) such that $R \not\models_{ind} C_k\tau$, then $R \models_{ind} P_k\theta_k\tau$ and $R \not\models_{ind} C\sigma\tau[d_k\theta_k\tau]$. Therefore, $R \models_{ind} g_k\theta_k\tau = d_k\theta_k\tau$, This implies that $R \not\models_{ind} C\sigma\tau[g_k\theta_k\tau]$, contradiction by hypothesis.

Case simplify: This case is similar to the previous one.

Simplify: Suppose that the rule *simplify* is applied to C, then there exists a clause C' such that $C \mapsto_{R<H\cup E>} C'$. Let σ be a ground substitution, by Lemma 4.8, we have $C\sigma \mapsto_{R<H\cup E>} C'\sigma$. The instances of clauses of $H \cup E$ used in the rewriting step are valid by hypothesis. On the other hand, we can assume that the premisses of C are valid (otherwise the proof is obvious). Hence $R \models_{ind} C'\sigma$ and Therefore, $R \models_{ind} C'$.

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

4.4.2 Refutational Completeness

In this section we shall study the refutational completeness of the proof by induction procedure. Let us introduce first the following definitions:

Definition 4.11 (Inductively irreducible term) *A term t is* quasi-irreducible *by a rewrite system R if for all ground R-irreducible substitution σ, $t\sigma$ is R-irreducible.*

Definition 4.12 (Strongly complete function, strongly complete system) *Let $f \in D$ a sufficiently complete defined function symbol. If for all the rules $p_i \Rightarrow f(\vec{t_i}) \rightarrow d_i$ whose left-hand sides are identical up to a renaming μ_i, we have $R \models_{ind} \vee_i p_i\mu_i$, then f is strongly complete w.r.t. R. We say that R is strongly complete if any function symbol is strongly complete w.r.t. R.*

Note that a *sufficiently complete* rewrite system is not necessarily *strongly complete.* This can be shown by the following example:

Example 4.13 *Let R be a conditional rewrite system which define the predicates $\leq$ and P with the constructors 0 and s:*

$$\begin{aligned} 0 \leq x &\rightarrow true \\ s(x) \leq 0 &\rightarrow false \\ s(x) \leq s(y) &\rightarrow x \leq y \\ x \leq y = true \Rightarrow P(x,y) &\rightarrow false \\ s(x) \leq y = false \Rightarrow P(s(x),y) &\rightarrow true \end{aligned} \tag{2}$$

We can easily show that R is sufficiently complete and not strongly complete since if we consider the axiom 2, the precondition $x \leq y = true$ is not an inductive consequence of R. ♦

The transformation of a sufficiently complete rewrite system to another one which is strongly complete is obvious if the functions are sufficiently complete over free constructors. For example:

Example 4.14 *Consider the example 4.13, the system R is equivalent to the following one which is strongly complete.*

$$\begin{aligned} 0 \leq x &\rightarrow true \\ s(x) \leq 0 &\rightarrow false \\ s(x) \leq s(y) &\rightarrow x \leq y \\ P(0,x) &\rightarrow false \\ s(x) \leq y = true \Rightarrow P(s(x),y) &\rightarrow false \\ s(x) \leq y = false \Rightarrow P(s(x),y) &\rightarrow true \end{aligned} \tag{3}$$

♦

Now, we can define a new inference system K from J by using test sets rather than cover sets and adding the following rules (see figure 4). The rules *positive (resp. negative)*

positive decomposition: $(E \cup \{f(\vec{s}) = f(\vec{t}) \vee r\}, H) \vdash_K (E \cup (\cup_i \{s_i = t_i \vee r\}), H)$
if f is a free constructor.

negative decomposition: $(E \cup \{\neg f(\vec{s}) = f(\vec{t}) \vee r\}, H) \vdash_K (E \cup \{\vee_i \neg(s_i = t_i) \vee r\}, H)$
if f is a free constructor.

positive clash: $(E \cup \{f(\vec{s}) = g(\vec{t}) \vee r\}, H) \vdash_K (E \cup \{r\}, H)$
if f and g are two distinct free constructors.

eliminate trivial equation: $(E \cup \{\neg(s = s) \vee r\}, H) \vdash_K (E \cup \{r\}, H)$

delete: $(E \cup \{\vee_{i=1}^n \neg(x_i = t_i) \vee r\}, H) \vdash_K (E, H)$
if for all i : $x_i \notin var(t_i)$ and $r\sigma$ is a tautology where $\sigma = \{x_i \leftarrow t_i \mid i \in [1 \cdots n]\}$.

occur check: $(E \cup \{\vee_{i=1}^n \neg(x_i = t_i) \vee r\}, H) \vdash_K (E, H)$
if there exist i such that $x_i \not\equiv t_i$, $x_i \in var(t_i)$ and t_i is inductively irreducible by R.

negative clash: $(E \cup \{\neg f(\vec{s}) = g(\vec{t}) \vee r\}, H) \vdash_K (E, H)$
if f and g are two distinct free constructors.

Figure 4 Inference System K

decomposition, positive clash and *eliminate trivial equation* take advantage of the fact that constructors are free to simplify clauses. The rules *delete*, *occur check* and *negative clash* delete redundant clauses.

The correctness of the inference system K w.r.t. definition 3.10 is expressed by the following lemma:

Lemma 4.15 *Let R be a ground convergent rewrite system. Then the inference system K is correct.*

Proof: Let R be a rewrite system and let $(E_0, \emptyset) \vdash_K (E_1, H_1) \vdash_K \cdots$ be a K-derivation. Suppose that $R \not\models_{ind} E_0$ and let C' be a minimal element w.r.t. $\prec_c$, of the set $\mathcal{F} = \{D\theta \mid D \in \cup_i E_i$ and θ is a ground R-irreducible substitution such that $R \not\models_{ind} D\theta\}$. Then there exists $C \in \cup_i E_i$ minimum w.r.t. $\prec_c$ and a ground R-irreducible substitution σ such that

$C' \equiv C\sigma$. We show whatever rule is applied to C, we obtain a contradiction.

Positive decomposition: Suppose that the rule *positive decomposition* is applied to

$$C \equiv f(s_1, \cdots, s_n) = f(t_1, \cdots, t_n) \vee r$$

Then, we have $R \not\models_{ind} f(s_1, \cdots, s_n)\sigma = f(t_1, \cdots, t_n)\sigma$. Then there exists i such that $R \not\models_{ind} s_i\sigma = t_i\sigma$. Let $Q \equiv s_i = t_i \vee r$, then we have $Q \in \cup_i E_i$, $R \not\models_{ind} Q\sigma$ and $Q\sigma \prec_c C\sigma$, contradiction.

Negative decomposition: Suppose that the rule *negative decomposition* is applied to

$$C \equiv \neg(f(s_1, \cdots, s_n) = f(t_1, \cdots, t_n)) \vee r$$

Then, we have $R \models_{ind} f(s_1, \cdots, s_n)\sigma = f(t_1, \cdots, t_n)\sigma$. Since R is a ground convergent rewrite system and f is a free constructor, then for all i, $R \models_{ind} s_i\sigma = t_i\sigma$. Let $Q \equiv \vee_i \neg(s_i = t_i) \vee r$, then we have $Q \in \cup_i E_i$, $R \not\models_{ind} Q\sigma$ and $Q\sigma \prec_c C\sigma$, contradiction.

Positive clash: Suppose that the rule *positive clash* is applied to

$$C \equiv f(s_1, \cdots, s_n) = g(t_1, \cdots, t_n) \vee r$$

Then, we have $R \not\models_{ind} r$ and $r \in \cup_i E_i$. On the other hand, $r\sigma \prec_c C\sigma$, contradiction.

Eliminate trivial equation: Suppose that the rule *eliminate trivial equation* is applied to

$$C \equiv \neg(s = s) \vee r$$

Then, we have $R \not\models_{ind} r$ and $r \in \cup_i E_i$. On the other hand, $r\sigma \prec_c C\sigma$, contradiction.

Delete: Suppose that the rule *delete* is applied to

$$C \equiv \vee_{i=1}^{n} \neg(x_i = t_i) \vee r$$

Then there exists i such that $x_i \not\equiv t_i$, $x_i \in var(t_i)$ and t_i is inductively irreducible by R. Therefore, $R \models_{ind} C\sigma$, contradiction.

Occur check: Suppose that the rule *occur check* is applied to

$$C \equiv \vee_{i=1}^{n} \neg(x_i = t_i) \vee r$$

Then, for all i : $x_i \notin var(t_i)$ and $r\tau$ is a tautology where $\tau = \{x_i \leftarrow t_i \mid i \in [1 \cdots n]\}$. Therefore, $R \models_{ind} C\sigma$, contradiction.

Negative clash: Suppose that the rule *negative clash* is applied to

$$C \equiv \neg(f(s_1, \cdots, s_n) = g(t_1, \cdots, t_n)) \vee r$$

Since f and g are two distinct free constructors and R is ground convergent, then we have $R \not\models_{ind} (f(s_1, \cdots, s_n) = g(t_1, \cdots, t_n))\sigma$ and Therefore, $R \models_{ind} C\sigma$, contradiction. □

If R is a ground convergent rewrite system, the inference system K can refute false conjectures. This result is a consequence to the following lemma:

Lemma 4.16 *Let R be a ground convergent rewrite system. Then the inference system K is refutationally correct w.r.t. definition 3.13.*

Proof: Let C be a clause in E_j and $(E_j, H_j) \vdash_I (E_{j+1}, H_{j+1})$ by the application of an inference rule on C. Let us show that $R \models_{ind} E_{j+1}$ if $H_0 = \emptyset$ and for all $i \leq j$ we have $R \models_{ind} E_i$. We will discuss the situation according to the rule which is applied to C:

Positive decomposition: Suppose that the rule *positive decomposition* is applied to

$$C \equiv f(s_1, \cdots, s_n) = f(t_1, \cdots, t_n) \vee r$$

Let σ be a ground substitution, since $R \models_{ind} C\sigma$, we have either $R \models_{ind} r\sigma$ or $R \models_{ind} f(s_1, \cdots, s_n)\sigma = f(t_1, \cdots, t_n)\sigma$. The first possibility gives immediately the desired conclusion. Hence let us assume the second one. Since R is a ground convergent rewrite system and f is a free constructor, then for all i we have $R \models_{ind} s_i\sigma = t_i\sigma$. Hence every clause $Q \equiv s_i = t_i \vee r$ verifies $R \models_{ind} Q\sigma$.

Negative decomposition: Suppose that the rule *negative decomposition* is applied to

$$C \equiv \neg(f(s_1, \cdots, s_n) = f(t_1, \cdots, t_n)) \vee r$$

Let σ be a ground substitution, since $R \models_{ind} C\sigma$, we have either $R \models_{ind} r\sigma$ or $R \not\models_{ind} f(s_1, \cdots, s_n)\sigma = f(t_1, \cdots, t_n)\sigma$. The first possibility gives immediately the desired conclusion. Hence let us assume the second one. There exist necessarily i such that $R \not\models_{ind} s_i\sigma = t_i\sigma$. Therefore, $R \models_{ind} (\vee_{i=1}^{n} \neg(s_i = t_i) \vee r)\sigma$.

Positive clash: Suppose that the rule *positive clash* is applied to

$$C \equiv f(s_1, \cdots, s_n) = g(t_1, \cdots, t_n) \vee r$$

Let σ be a ground substitution, since R is ground convergent and f and g are two distincts free constructors, then $R \not\models_{ind} (f(s_1, \cdots, s_n) = g(t_1, \cdots, t_n))\sigma$ and Therefore, $R \models_{ind} r\sigma$.

Eliminate trivial equation: Suppose that the rule *eliminate trivial equation* is applied to

$$C \equiv \neg(s = s) \vee r$$

Let σ be a ground substitution, then we have $R \not\models_{ind} \neg(s = s)\sigma$ and Therefore, $R \models_{ind} r\sigma$.

Delete, occur check and negative clash: If C is deleted then $R \models_{ind} E_{j+1}$ since $E_{j+1} \subseteq E_j$ in this case. □

Now, let us add to the inference system K the following rule:

refute: $(E \cup \{C\}, H) \vdash_K$ *Refutation*
if for all (E', H'), $(E \cup \{C\}, H) \vdash_K (E', H')$ implies $C \in E'$.

The inference system K is refutationally complete, in that any conjecture that is not valid in the initial model will be disproved. This result is a consequence of the following results:

Lemma 4.17 *Let R be a ground convergent rewrite system which is strongly complete over free constructors. If the rule* refute *is applied to C, then C is not an inductive consequence of R.*

Proof: Let C be a clause such that no condition of the rules of K rather than *refute* hold for C. Let us show first that C does not contain any defined function. Otherwise, C contains a term s of the form $f(t_1, \cdots, t_n)$ where f is a defined symbol and for all $i \in [1 \cdots n]$, $t_i \in T(C, X)$. Let σ be a test substitution of C, the terme $s\sigma$ is an instance of a left-hand side of R, otherwise, by the proposition 4.2, there exists a substitution τ such that $s\sigma\tau$ is ground and R-irreducible, which contradicts the fact that f is strongly complete. Then inductive rewriting or case analysis can be applied, and therefore the *generate* rule is applied to C, contradiction.

By Lemma 4.17, we conclude that the clause C contains only constructor terms. Now, since the constructors are free and no simplification rules can be applied to C, then C is necessarily is of the form $\bigwedge_{i=1}^{n} x_i = t_i \Rightarrow r$ such that for all i, $x_i \not\equiv t_i$. On the other hand, for all i, $x_i \notin var(t_i)$, otherwise the rule *occur check* is applied to C since for all i, t_i is inductively irreducible by R. Consider the substitution $\tau = \{x_i \leftarrow t_i \mid i \in [1 \cdots n]\}$. The clause $C\tau$ is not a tautology, otherwise the rule *delete* is applied to C. Then the clause $C\tau$ is provably inconsistent, and Therefore, by theorem 3.7, $C\tau$ is not an inductive consequence of R, and finally C is not also an inductive consequence of R. □

The inference system K is refutationally complete, this result is expressed by the following theorem:

Theorem 4.18 *Let R be a ground convergent rewrite system that is strongly complete over free constructors. If a derivation issued from $(E_0, \emptyset)$ terminates by application of the rule* refute, *then $R \not\models_{ind} E_0$. Conversely, if $R \not\models_{ind} E_0$, then all fair derivations issued from $(E_0, \emptyset)$ terminate by application of the rule* refute.

Proof:

$\Rightarrow$ by the lemmas 4.9 and 4.15.

$\Leftarrow$ by the lemmas 4.10, 4.16 and 4.17. □

5 Computer Experiments

Example 5.1 *Suppose we have a* forward counter *(see figure 5) and a* backward counter *(see figure 6). To prove the termination of the axioms, we can use the* lrpo *ordering $\prec$ (see [Dershowitz, 1987]) with the following precedence on functions:*

$$init \prec input \prec + \prec * \prec not \prec c \approx c_1 \approx q \approx q_1$$

The theorem to be proved is:

$$q(i,t) = not(q_1(i,t))$$

The specification of these two counters is immediate (see figure 7).

This problem cannot be proved by Nqthm (without modifying the axioms) due to the presence of mutually recursive operators. Note also that the specification is not sufficiently complete and the function not *is not defined over constructors* (not(not(x)) $\rightarrow$ x)*. The scheme of the proof generated by SPIKE is presented in figure 8.*

Here is the direct proof generated by SPIKE.

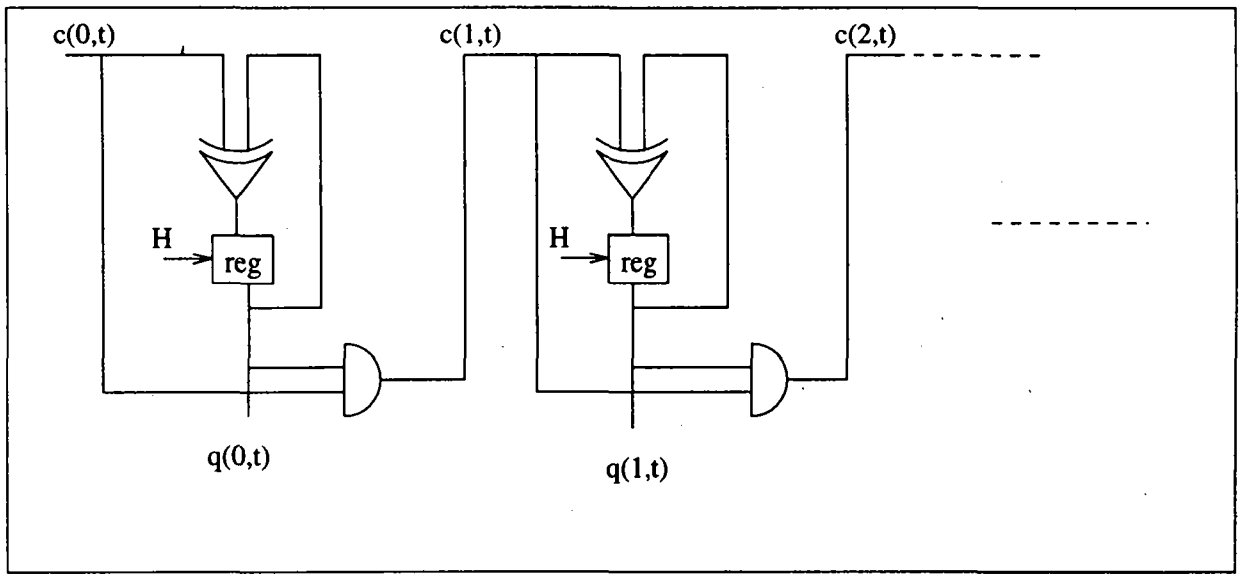

Figure 5 Forward counter

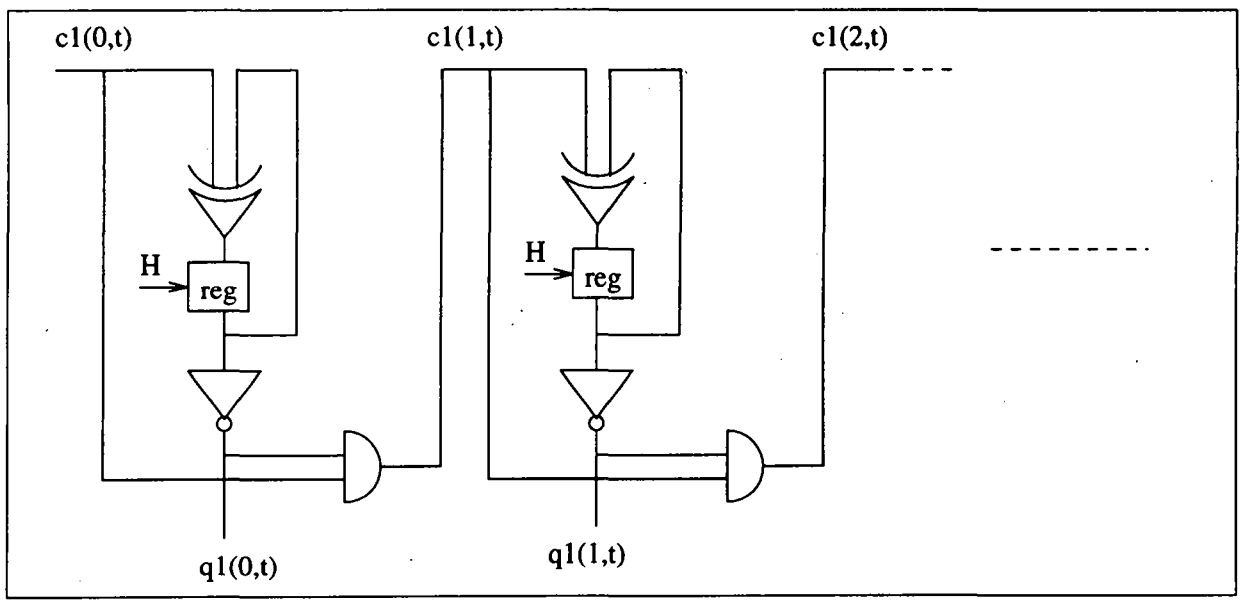

Figure 6 Backward counter

specification: counter

sorts nat, bool;

constructors:

True:	$\rightarrow$ bool;
False:	$\rightarrow$ bool;
$+$: bool $\times$ bool	$\rightarrow$ bool;
$*$: bool $\times$ bool	$\rightarrow$ bool;
0:	$\rightarrow$ nat;
s: nat	$\rightarrow$ nat;
input: nat	$\rightarrow$ bool;
init: nat	$\rightarrow$ bool;

defined functions:

c: nat $\times$ nat	$\rightarrow$ bool;
q: nat $\times$ nat	$\rightarrow$ bool;
c_1: nat $\times$ nat	$\rightarrow$ bool;
q_1: nat $\times$ nat	$\rightarrow$ bool;
not: bool	$\rightarrow$ bool;

axioms:

$c(0,t) = input(t)$;
$c(s(i),t) = c(i,t) * q(i,t)$;
$q(i,0) = init(i)$;
$q(i,s(t)) = q(i,t) + c(i,t)$;
$c_1(0,t) = input(t)$;
$c_1(s(i),t) = c_1(i,t) * not(q_1(i,t))$;
$q_1(i,0) = not(init(i))$;
$q_1(i,s(t)) = q_1(i,t) + c_1(i,t)$;
$not(not(x)) = x$;
$not(x+y) = not(x) + y$;

Figure 7 Specification of Counters

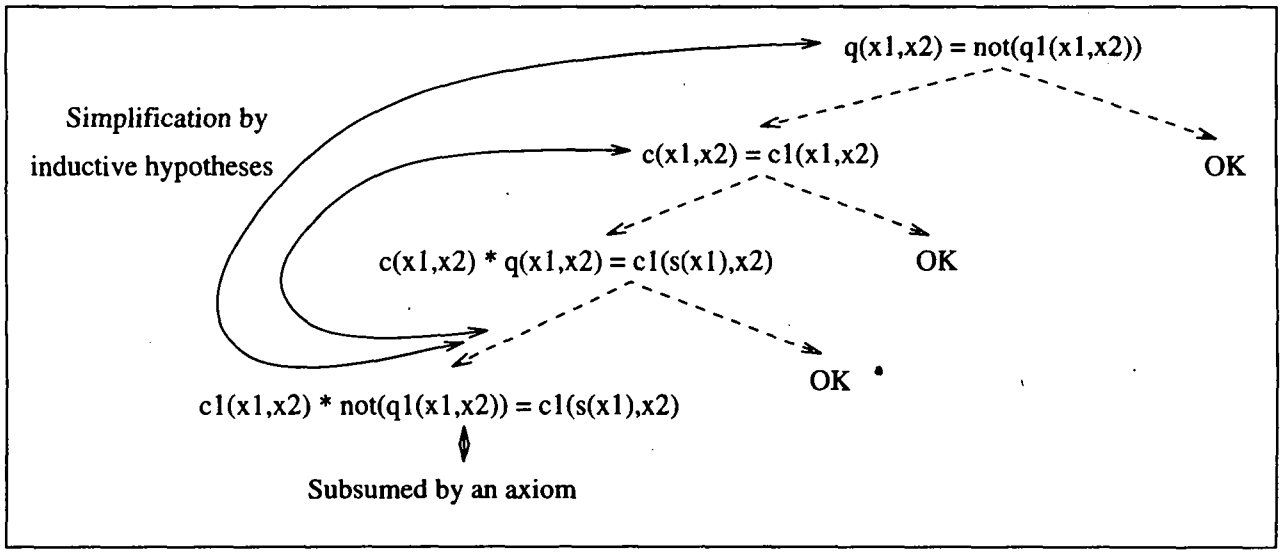

Figure 8 Proof of Example 5.1

```
cover set of R:

 -> bool = {False ; True ; not(x1) ; x1+x2 ; x1*x2}
 -> nat = {0 ; s(x1)}

induction positions of functions:

 -> c : [[1]]
 -> q : [[2]]
 -> c1 : [[1]]
 -> q1 : [[2]]
 -> not : [[1]]

E0 = {q(x1,x2) = not(q1(x1,x2))}

Application of generate on:
    q(x1,x2) = not(q1(x1,x2)) :
 1) init(x1) = not(q1(x1,0)) ;
 2) q(x1,x2)+c(x1,x2) = not(q1(x1,s(x2)))


E1 = {init(x1) = not(q1(x1,0)) ;
      q(x1,x2)+c(x1,x2) = not(q1(x1,s(x2)))}

H1 = {q(x1,x2) = not(q1(x1,x2))}

Simplification of:
```

```
    q(x1,x2)+c(x1,x2) = not(q1(x1,s(x2))) by H1:
    not(q1(x1,x2))+c(x1,x2) = not(q1(x1,s(x2)))

E2 = {init(x1) = not(q1(x1,0)) ;
      not(q1(x1,x2))+c(x1,x2) = not(q1(x1,s(x2)))}

H2 = {q(x1,x2) = not(q1(x1,x2))}

Simplification of:
    init(x1) = not(q1(x1,0)) by R[H2 U E2]:
    init(x1) = init(x1)

Simplification of:
    not(q1(x1,x2))+c(x1,x2) = not(q1(x1,s(x2))) by R[H2 U E2]:
    not(q1(x1,x2))+c(x1,x2) = not(q1(x1,x2))+c1(x1,x2)

E3 = {init(x1) = init(x1) ;
      not(q1(x1,x2))+c(x1,x2) = not(q1(x1,x2))+c1(x1,x2)}

H3 = {q(x1,x2) = not(q1(x1,x2))}

Delete  init(x1) = init(x1)

Simplification of:
    not(q1(x1,x2))+c(x1,x2) = not(q1(x1,x2))+c1(x1,x2)
     1) not(q1(x1,x2)) = not(q1(x1,x2)) ;
     2) c(x1,x2) = c1(x1,x2)

E4 = {not(q1(x1,x2)) = not(q1(x1,x2)) ;
      c(x1,x2) = c1(x1,x2)}

H4 = {q(x1,x2) = not(q1(x1,x2))}

Delete  not(q1(x1,x2)) = not(q1(x1,x2))

Application of generate on:
    c(x1,x2) = c1(x1,x2) :
 1) input(x2) = c1(0,x2) ;
 2) c(x1,x2)*q(x1,x2) = c1(s(x1),x2)

E5 = {input(x2) = c1(0,x2) ;
      c(x1,x2)*q(x1,x2) = c1(s(x1),x2)}

H5 = {c(x1,x2) = c1(x1,x2) ;
      q(x1,x2) = not(q1(x1,x2))}
```

```
Delete  input(x2) = c1(0,x2)
 it is subsumed by:c1(0,x1) = input(x1) of R

E6 = {c(x1,x2)*q(x1,x2) = c1(s(x1),x2)}

H6 = {c(x1,x2) = c1(x1,x2) ;
      q(x1,x2) = not(q1(x1,x2))}

Simplification of:
    c(x1,x2)*q(x1,x2) = c1(s(x1),x2) by H6:
    c(x1,x2)*not(q1(x1,x2)) = c1(s(x1),x2)

E7 = {c(x1,x2)*not(q1(x1,x2)) = c1(s(x1),x2)}

H7 = {c(x1,x2) = c1(x1,x2) ;
      q(x1,x2) = not(q1(x1,x2))}

Simplification of:
    c(x1,x2)*not(q1(x1,x2)) = c1(s(x1),x2) by H7 U E7[R]:
    c1(x1,x2)*not(q1(x1,x2)) = c1(s(x1),x2)

E8 = {c1(x1,x2)*not(q1(x1,x2)) = c1(s(x1),x2)}

H8 = {c(x1,x2) = c1(x1,x2) ;
      q(x1,x2) = not(q1(x1,x2))}


Delete  c1(x1,x2)*not(q1(x1,x2)) = c1(s(x1),x2)
 it is subsumed by:c1(s(x1),x2) = c1(x1,x2)*not(q1(x1,x2)) of R

E9 = {}

H9 = {c(x1,x2) = c1(x1,x2) ;
      q(x1,x2) = not(q1(x1,x2))}

The initial conjectures are inductive consequences of R
```

♦

Example 5.2 (Factorial) *Consider the following circuit [Pierre, 1990] (see figure 9) which computes the factorial of I (a natural number) and returns the result in the register R. J is a register and the functionalities of the different modules are:*

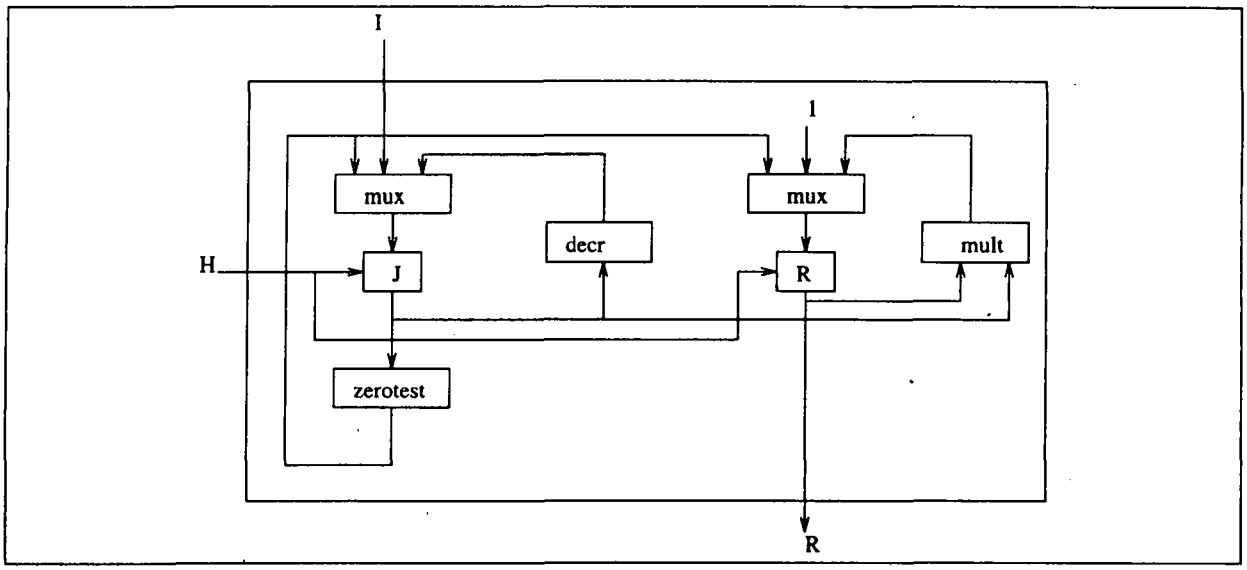

Figure 9 Factorial

$$
\begin{aligned}
mux(c, e_1, e_2) &\Leftrightarrow \text{if } c \text{ then } e_1 \text{ else } e_2 \\
zerotest(e, s) &\Leftrightarrow s \equiv (e = 0) \\
decr(e, s) &\Leftrightarrow s = (e - 1) \\
mult(e_1, e_2, s) &\Leftrightarrow s = e_1 * e_2
\end{aligned}
$$

The specification of this problem is given in figure 10. To prove the termination of the axioms, we can use the lrpo *ordering* $\prec$ *with the following precedence on functions:*

$$0 \prec s \prec + \prec * \prec fact \prec f \prec g$$

and the following status:

$$(f, RL);\ (*, RL);\ (+, LR);\ (g, LR)$$

The theorem to be proved is:

$$g(i) = fact(i)$$

which can be generalised to:

$$f(i, j, j) = i * fact(j)$$

To prove this theorem, we need the following lemmas: $x*s(0) = x$ *and* $x*(y*z) = (x*y)*z$ *which can be easily proved with SPIKE.*

Here is the direct proof generated by SPIKE:

specification: factorial

sorts nat;

constructors:

0: $\rightarrow$ nat;

s: nat $\rightarrow$ nat;

defined functions:

fact: nat $\rightarrow$ nat;

f: nat $\times$ nat $\times$ nat $\rightarrow$ nat;

g: nat $\rightarrow$ nat;

$+$: nat $\times$ nat $\rightarrow$ nat;

$*$: nat $\times$ nat $\rightarrow$ nat;

axioms:

$fact(0) = s(0)$;

$fact(s(n)) = s(n) * fact(n)$;

$f(r, j, 0) = r$;

$f(r, 0, s(i)) = f(s(0), s(i), i)$;

$f(r, s(j), s(i)) = f(r * s(j), j, i)$;

$g(i) = f(s(0), i, i)$;

$x + 0 = x$;

$x + s(y) = s(x + y)$;

$x * 0 = 0$;

$x * s(y) = x + (x * y)$;

Figure 10 Factorial

```
test set of R:

 -> nat = {0 ; s(x1)}

induction positions of functions:

 -> fact : [[1]]
 -> f : [[2];[3]]
 -> + : [[2]]
 -> * : [[2]]

E0 = {f(x1,x2,x2) = x1*fact(x2)}

L = {x1*s(0) = x1 ;
     x1*(x2*x3) = (x1*x2)*x3}

Application of generate on:
    f(x1,x2,x2) = x1*fact(x2) :
 1) x1 = x1*fact(0) ;
 2) f(x1+(x1*x2),x2,x2) = x1*fact(s(x2))

E1 = {x1 = x1*fact(0) ;
      f(x1+(x1*x2),x2,x2) = x1*fact(s(x2))}

H1 = {f(x1,x2,x2) = x1*fact(x2)}

Simplification of:
    f(x1+(x1*x2),x2,x2) = x1*fact(s(x2)) by H1:
    (x1+(x1*x2))*fact(x2) = x1*fact(s(x2))

E2 = {x1 = x1*fact(0) ;
      (x1+(x1*x2))*fact(x2) = x1*fact(s(x2))}

H2 = {f(x1,x2,x2) = x1*fact(x2)}

Simplification of:
    x1 = x1*fact(0) by R U L[H2 U E2]:
    x1 = x1

Simplification of:
    (x1+(x1*x2))*fact(x2) = x1*fact(s(x2)) by R U L[H2 U E2]:
    (x1+(x1*x2))*fact(x2) = (x1+(x1*x2))*fact(x2)

E3 = {x1 = x1 ;
      (x1+(x1*x2))*fact(x2) = (x1+(x1*x2))*fact(x2)}
```

```
H3 = {f(x1,x2,x2) = x1*fact(x2)}

Delete  x1 = x1

Delete  (x1+(x1*x2))*fact(x2) = (x1+(x1*x2))*fact(x2)

The initial conjectures are inductive consequences of R
```

♦

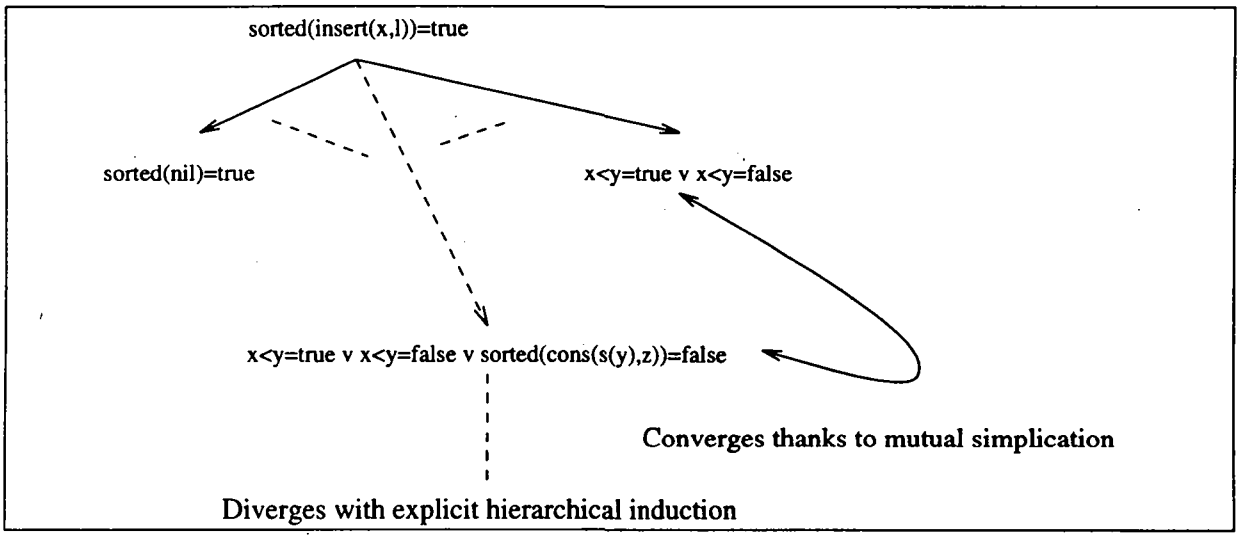

Figure 11 Proof of Example 5.2

Example 5.3 *Consider the specification of lists over natural numbers with an "insert" operation, a "sorted" predicate on lists that is true iff a list is ordered and an "isort" operation which sorts (by insertion) a list of natural numbers. SPIKE can prove the following conjectures:*

$$\{isort(l) = true,\ sorted(insert(x, l)) = sorted(l)\}$$

in a completely automatic way (i.e. without any lemmas). Note that RRL is unable to succeed with this example unless the user suggests some well-chosen lemmas. Figure 11 shows how SPIKE can prove the first conjecture without lemmas. ♦

Example 5.4 (Example with non sufficiently complete function) *Consider the following axioms which define the function* append1 *which add an element to the end of a*

list and length *which compute the length of a list.*

$$\begin{aligned} append1(Nil, x) &\rightarrow Cons(x, Nil) \\ append1(Cons(y, z), x) &\rightarrow Cons(y, append1(z, x)) \\ length(Cons(y, z)) &\rightarrow s(length(z)) \end{aligned}$$

The function length *is not sufficiently complete. Let us prove the conjecture:*

$$length(append1(y, x)) = s(length(y))$$

Note that such problems cannot be proved by Nqthm which need that all the functions are sufficiently complete.

Here is the direct proof generated by SPIKE:

```
cover set of R:

 -> list = {Nil ; Cons(x1,x2)}
 -> nat = {0 ; s(x1)}

induction positions of functions:

 -> append1 : [[1]]
 -> length : [[1]]

E0 = {length(append1(x1,x2)) = s(length(x1))}

Application of generate on:
    length(append1(x1,x2)) = s(length(x1)) :
 1) s(length(Nil)) = s(length(Nil)) ;
 2) s(length(append1(x3,x2))) = s(length(Cons(x1,x3)))

Delete  s(length(Nil)) = s(length(Nil))

E1 = {s(length(append1(x3,x2))) = s(length(Cons(x1,x3)))}

H1 = {length(append1(x1,x2)) = s(length(x1))}

Simplification of:
    s(length(append1(x3,x2))) = s(length(Cons(x1,x3))) by H1:
    s(s(length(x3))) = s(length(Cons(x1,x3)))

E2 = {s(s(length(x3))) = s(length(Cons(x1,x3)))}

H2 = {length(append1(x1,x2)) = s(length(x1))}
```

```
Simplification of:
    s(s(length(x3))) = s(length(Cons(x1,x3))) by R[H2 U E2]:
    s(s(length(x3))) = s(s(length(x3)))

E3 = {s(s(length(x3))) = s(s(length(x3)))}

H3 = {length(append1(x1,x2)) = s(length(x1))}

Delete  s(s(length(x3))) = s(s(length(x3)))

The initial conjectures are inductive consequences of R
```

♦

Example 5.5 (Other examples) *Consider the following axioms which specify* $+$, $*$, dbl, half, even, odd, $even_m$ *and* odd_m *on natural numbers. Note that* $even_m$ *is defined by a mutual recursion with* odd_m.

$$
\begin{aligned}
x + 0 &\rightarrow x \\
x + s(y) &\rightarrow s(x + y) \\
x * 0 &\rightarrow 0 \\
x * s(y) &\rightarrow (x * y) + x \\
dbl(0) &\rightarrow 0 \\
dbl(s(n)) &\rightarrow s(s(dbl(n))) \\
half(0) &\rightarrow 0 \\
half(s(0)) &\rightarrow 0 \\
half(s(s(n))) &\rightarrow s(half(n)) \\
even_m(0) &\rightarrow True \\
odd_m(0) &\rightarrow False \\
even_m(s(0)) &\rightarrow False \\
odd_m(s(0)) &\rightarrow True \\
even_m(s(n)) &\rightarrow odd_m(n) \\
odd_m(s(n)) &\rightarrow even_m(n)
\end{aligned}
$$

No	Theorems	Lemmas
1	$x*y=0 \Rightarrow x=0 \vee y=0$	
2	$x+y=y+x$	
3	$s(x)+x=s(x+x)$	$x+y=y+x$
4	$dbl(x)=x+x$	$x+y=y+x$
5	$len(x <> z :: y)=s(len(x <> y))$	
6	$len(x <> y)=len(y <> x)$	$len(x <> z :: y)=s(len(x <> y))$
7	$len(x <> y)=len(x)+len(y)$	$x+y=y+x$
8	$len(x <> x)=dbl(len(x))$	$len(x <> z :: y)=s(len(x <> y))$
9	$even_m(x+x) = True$	$x+y = y+x$
10	$even_m(x)=True \Rightarrow half(x)+half(x)=x$	$x+y=y+x$
11	$odd_m(s(x+x)) = True$	$x+y=y+x$
12	$odd_m(x+x) = False$	$x+y=y+x$
13	$odd_m(x+y) = True \vee odd_m(y) = True \vee odd_m(x) = False$	$x+y=y+x$
14	$odd_m(x+y) = True \vee odd_m(y) = False \vee odd_m(x) = True$	$x+y=y+x$
15	$odd_m(x+y) = False \vee odd_m(y) = False \vee odd_m(x) = False$	$x+y=y+x$
16	$odd_m(x+y) = False \vee odd_m(y) = True \vee odd_m(x) = True$	$x+y=y+x$
17	$even(x+y) = True \vee even(y) = False \vee even(x) = False$	$x+y=y+x$
18	$even(x+y) = True \vee even(y) = True \vee even(x) = True$	$x+y=y+x$
19	$even(x+y) = False \vee even(y) = False \vee even(x) = True$	$x+y=y+x$
20	$even(x+y) = False \vee even(y) = True \vee even(x) = False$	$x+y=y+x$
21	$len(rot(x,z <> [y]))=s(len(rot(x,z)))$	
22	$len(rev(x))=len(x)$	$len(x <> [y])=s(len(x))$
23	$len(x <> [y])=s(len(x))$	
24	$rev(x <> [y])=y :: rev(x)$	
25	$rev(rev(x) <> [y])=y :: x$	$rev(x <> [y])=y :: rev(x)$
26	$len(rev(x <> y))=len(x)+len(y)$	$len(x <> [y])=s(len(x))$ $s(x)+y=x+s(y)$
27	$len(qrev(x,z :: y))=s(len(qrev(x,y)))$	
28	$qrev(x,y)=rev(x) <> y$	$(x <> [y]) <> z=x <> y :: z$
29	$len(qrev(x,y))=len(x)+len(y)$	$s(x)+y=x+s(y)$
30	$qrev(x <> [y],z)=y :: qrev(x,z)$	
31	$nth(s(i),nth(j,y :: x))=nth(i,nth(j,x))$	

Table 1 Some examples with SPIKE.

Consider also the following axioms which specify the functions len, append, nth, rot, rev *and* qrev *on lists over natural numbers.*

$$
\begin{aligned}
len(Nil) &\rightarrow 0 \\
len(cons(h,t)) &\rightarrow s(len(t)) \\
append(Nil,a) &\rightarrow a \\
append(cons(h,a),t) &\rightarrow cons(h,append(a,t)) \\
nth(0,l) &\rightarrow l \\
nth(x,Nil) &\rightarrow Nil \\
nth(s(x),cons(h,t)) &\rightarrow nth(x,t) \\
rot(0,l) &\rightarrow l \\
rot(n,Nil) &\rightarrow Nil \\
rot(s(n),cons(h,t)) &\rightarrow rot(n,append(t,cons(h,Nil))) \\
rev(Nil) &\rightarrow Nil \\
rev(cons(h,t)) &\rightarrow append(rev(t),cons(h,Nil)) \\
qrev(Nil,l) &\rightarrow l \\
qrev(cons(h,t),l) &\rightarrow qrev(t,cons(h,l))
\end{aligned}
$$

Now, a few examples are given in Table 1. For brevity, :: *is written for infix* cons, <> *for infix* append, [] *for the empty list* nil, *and* $[x]$ *for the list* $cons(x,nil)$. *Note that the lemmas are derived automatically using the* divergence critic *developed by Toby Walsh [Walsh, 1994].* ♦

6 Conclusion

We have proposed a general scheme for test set induction procedure and described a simple technique to prove the correctness of this procedure. Our technique is easier and more general than the one given in [Reddy, 1990; Bouhoula and Rusinowitch, 1994]. Therefore, it can be easily extended to new inference rules. We argue that test set induction is the best technique for mechanising induction, that is to derive induction proofs with minimal interaction from the user. The main arguments in favour of our method are:

Well-founded ordering: The well-founded ordering on which induction is based is exactly the termination ordering used to orient the axioms into rules. Therefore, the various

mechanical tools that were designed to prove termination of rewrite systems are readily available for suggesting good induction orderings.

Induction schemas: Schemas are defined first by a function, which, given a conjecture, selects the positions of variables where the induction will be applied (induction variables), and second by a special set of terms called a *cover set* or a *test set* by which these variables must be instantiated. We have proposed a new notion of induction positions and an algorithm to compute them. This computation is carried out only once for a given specification and permits us to determine whether a variable position of a conjecture is an induction variable or not. Then we guarantee the efficiency of the method because it is not necessary to consult the axioms several times in order to select the induction variables of a conjecture. We have also proposed a procedure computing test sets even if the constructors are not free.

Generation of lemmas: The lemmas are generated by replacing the induction variables of a conjecture by test sets or cover sets and applying inductive rewriting or case analysis.

Mutual induction: Conjectures are processed in a non-hierarchical list. New sublemmas to be proved are simply added to this list. Therefore, mutual induction is automatically handled by our technique. This point is also crucial for handling mutually recursive definitions. Recently, we have noted that our approach has some advantages concerning this problematic aspect of explicit induction techniques [Bouhoula, 1994a].

Case analysis: With test set induction, case analysis can easily be simulated by term rewriting. Divergence problems are avoided by applying conditional rules.

Refutation: We can refute false conjectures when the axioms are presented by a ground convergent conditional rewrite system. A conjecture is rejected whenever an inconsistency appears. We have proposed a new notion of provable inconsistency which allows us to refute more false conjectures than with previous approaches. Our strategy is also refutationally complete, in that it refutes any conjecture which is not valid in the initial model provided that the axioms are ground convergent and the functions are completely defined over free constructors (not restricted to boolean specifications).

Our test set induction procedure is implemented in the prover SPIKE. SPIKE has proved several interesting problems with a minimum of interaction with the user [Bouhoula, 1994b]. It has been extended by a powerful *divergence critic* [Walsh, 1994] which relies

on the difference matching algorithm of [Basin and Walsh, 1993]. This critic has proved very successful at identifying divergence and proposing appropriate lemmas and generalisations for a large number of theorems; it enables the system SPIKE to prove many theorems from the definitions alone where other systems like Nqthm or Clam fail [Walsh, 1994].

References

[Aubin, 1979] R. Aubin. Mechanizing structural induction. In *Theoretical Computer Science*, volume 9, pages 329–362, 1979.

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

[Basin and Walsh, 1993] D. Basin and T. Walsh. Difference unification. In Ruzena Bajcsy, editor, *Proceedings 13th International Joint Conference on Artificial Intelligence, Chambéry (France)*, volume 1, pages 116–122. Morgan Kaufmann, August 1993.

[Bouhoula and Rusinowitch, 1993] Adel Bouhoula and Michaël Rusinowitch. Automatic case analysis in proof by induction. In Ruzena Bajcsy, editor, *Proceedings 13th International Joint Conference on Artificial Intelligence, Chambéry (France)*, volume 1, pages 88–94. Morgan Kaufmann, August 1993.

[Bouhoula and Rusinowitch, 1994] Adel Bouhoula and Michaël Rusinowitch. Implicit induction in conditional theories, 1994. Accepted for publication by *Journal of Automated Reasoning*, to appear. Also available as INRIA report 2045, 1993.

[Bouhoula *et al.*, 1995] Adel Bouhoula, Emmanuel. Kounalis, and Michaël. Rusinowitch. Automated mathematical induction. 1995. Accepted for publication by *Journal of Logic and Computation*, to appear. Also available as INRIA report 1636, 1992.

[Bouhoula, 1994a] Adel Bouhoula. The challenge of mutual recursion and mutual simplification in implicit induction. Invited talk at the International Workshop on the Automation of Proof by Mathematical Induction, June 1994.

[Bouhoula, 1994b] Adel Bouhoula. *Preuves Automatiques par Récurrence dans les Théories Conditionnelles.* PhD thesis, Université de Nancy I, Mars 1994.

[Bouhoula, 1994c] Adel Bouhoula. Spike: a system for sufficient completeness and parameterized inductive proof. In A. Bundy, editor, *Proceedings 12th International Conference*

on Automated Deduction, Nancy (France), volume 814 of *Lecture Notes in Artificial Intelligence*, pages 836–840. Springer-Verlag, June 1994.

[Bouhoula, 1994d] Adel Bouhoula. Sufficient completeness and parameterized proofs by induction. In *Proceedings Fourth International Conference on Algebraic and Logic Programming, Madrid (Spain)*, volume 850 of *Lecture Notes in Computer Science*, pages 23–40. Springer-Verlag, September 1994.

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

[Boyer and Moore, 1988] R.S Boyer and J. S. Moore. *A Computationnal Logic Handbook.* Academic Press inc., Boston, 1988.

[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 *Lecture Notes in Artificial Intelligence*, pages 132–146. Springer-Verlag, July 1989.

[Bundy *et al.*, 1991] A. Bundy, van Harmelen F., J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, 1991. In press. Earlier version available from Edinburgh as Research Paper No 413.

[Burstall, 1969] R. M. Burstall. Proving properties of programs by structural induction. *Computer Journal*, 12:41–48, 1969.

[Chadha and Plaisted, 1992] R. Chadha and D. A. Plaisted. Mechanizing mathematical induction. Technical report, 1992. Presented at the Second International Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale, Florida.

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

[Dershowitz, 1987] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1 & 2):69–116, 1987.

[Dowek *et al.*, 1991] G. Dowek, A. Felty, H. Herbelin, G. Huet, C. Paulin-Mohring, and B. Werner. The Coq Proof Assistant. User's guide, INRIA-CNRS-ENS, 1991.

[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 *Lecture Notes in Computer Science*, pages 105–115. Springer-Verlag, 1986.

[Gramlich, 1989] B. Gramlich. UNICOM: a refined completion based inductive theorem-prover. In M. E. Stickel, editor, *10th International Conference on Automated Deduction*, volume 449 of *Lecture Notes in Artificial Intelligence*, pages 655–656. Springer-Verlag, July 1989.

[Hofbauer and Huber, 1994] D. Hofbauer and M. Huber. Linearizing term rewriting systems using test set. *Journal of Symbolic Computation*, 17(1):91–129, 1994.

[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. Preliminary version in Proceedings 21st Symposium on Foundations of Computer Science, IEEE, 1980.

[Huet, 1991] G. Huet. The gilbreath trick: a case study in axiomatisation and proof development in coq proof assistant. Technical Report 1511, INRIA, 1991.

[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 and Choquer, 1986] S. Kaplan and M. Choquer. On the decidability of quasi-reducibility. *Bulletin of European Association for Theoretical Computer Science*, 28:32–34, February 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 *et al.*, 1986] D. Kapur, P. Narendran, and H. Zhang. Proof by induction using test sets. In *Proceedings 8th International Conference on Automated Deduction, Oxford (UK)*, volume 230 of *Lecture Notes in Computer Science*, pages 99–117. Springer-Verlag, 1986.

[Kapur *et al.*, 1990] D. Kapur, P. Narendan, and F. Otto. On ground confluence of term rewriting systems. *Information and Computation*, May 1990.

[Kounalis and Rusinowitch, 1990] E. Kounalis and M. Rusinowitch. A mechanization of conditional reasoning. In *First International Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale, Florida*, January 1990.

[Kounalis, 1990] E. Kounalis. Testing for inductive (co)-reducibility. In A. Arnold, editor, *Proceedings 15th CAAP, Copenhagen (Denmark)*, volume 431 of *Lecture Notes in Computer Science*, pages 221–238. Springer-Verlag, May 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. ACM, 1980.

[Padawitz, 1988] P. Padawitz. *Computing in Horn Clause Theories.* Springer-Verlag, 1988.

[Pierre, 1990] L. Pierre. *Représentation fonctionnelle et preuve automatisée de circuits digitaux.* PhD thesis, Thèse de l'université de Provence, Aix-Marseille, 1990.

[Plaisted, 1985] D. Plaisted. Semantic confluence and completion method. *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 *Lecture Notes in Computer Science*, pages 162–177. Springer-Verlag, 1990.

[Walsh, 1994] T. Walsh. A divergence critic. In A. Bundy, editor, *Proceedings 12th International Conference on Automated Deduction, Nancy (France)*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 14–28. Springer-Verlag, June/July 1994.

[Walther, 1993] C. Walther. Combining induction axioms by machine. In Ruzena Bajcsy, editor, *Proceedings 13th International Joint Conference on Artificial Intelligence, Chambéry (France)*, volume 1, pages 95–100. Morgan Kaufmann, August 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 *Lecture Notes in Computer Science*, 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 lès 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 1 (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)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)

ISSN 0249 - 6399