Automatic Case Analysis in Proof by Induction

Adel Bouhoula and Michael Rusinowitch
CR1N & INRIA-Lorraine
BP 239, 54506 Vandoeuvre-les-Nancy, France
email: {bouhoula,rusi}@loria.fr

Abstract

We propose a new procedure for proof by induction in conditional theories where case analysis is simulated by term rewriting. This technique reduces considerably the number of variables of a conjecture to be considered for applying induction schemes (inductive positions). Our procedure is presented as a set of inference rules whose correctness has been formally proved. Moreover, when the axioms are ground convergent and the defined functions are completely defined over free constructors, it is possible to apply the system for refuting conjectures. The procedure is even refutationally complete for conditional equations with boolean preconditions (under the same hypotheses). The method is entirely implemented in the prover SPIKE. This system has proved interesting examples in a completely automatic way, that is, without interaction with the user and without ad-hoc heuristics. It has also proved the challenging Gilbreath card trick, with only 5 easy lemmas.

1 Introduction

Formal methods are more and more frequently adopted by industry for hardware and software verification. They require efficient automatic tools to relieve designers and programmers of the related proof obligations. Mathematical induction is essential as a technique for building formal proofs in this context. Its power is expressed by the successes of Nqthm [Boyer and Moore, 1979] that has been for many years the only significant automated theorem proving system for induction. However Nqthm requires a lot of interaction with the user. For instance many lemmas need to be given to Nqthm as milestones even for simple proof tasks.

Another direction for automating induction was proposed in the early eighties, the *inductionless induction* technique [Musser, 1980; Huet and Hullot, 1982] whose principle is to simulate induction by term rewriting. This method is refutational and does not require human interaction. While very limited at the beginning, its domain of application has widened considerably, thanks to the contributions of [Jouannaud and Kounalis, 1986] who relaxed the conditions on *constructor symbols* and of [Fribourg, 1986] who showed that only linear derivations were needed. More recently the method has been completely freed from the completion framework [Kounalis and Rusinowitch, 1990a; Reddy, 1990]. It has now become possible to apply it to conditional equational theories [Kounalis and Rusinowitch, 1990b; Bouhoula *et* al., 1992a]. Inductionless induction in our new setting reduces to *firstly* instanciating conjectures by induction schemes called *test sets* and *secondly* simplifying them by axioms, other conjectures or induction hypotheses. Every iteration generates new lemmas that are processed in the same way as the initial conjectures. The method does not require any hierarchy between the lemmas. They are all stored in a list and using conjectures for mutual simplification simulates si*multaneous induction.* The system SPIKE has been developed [Bouhoula *et* al., 1992b] on this principle and incorporates many optimizations such as powerful simplification techniques. To our knowledge, this system is the only one that can prove and disprove inductive theorems in conditional theories without any interaction.

However computer experiments have convinced us of the necessity of introducing a proper rule to perform case reasoning. Case analysis is a fundamental reasoning technique. A typical instance of it is the cut rule that consists in splitting a goal formula *A* along another formula *C* for generating two subgoals $C \Rightarrow A$ and $\neg C \Rightarrow A$. The main difficulty, already recognized by logicians a long time ago, relies in the choice of the cut formula C. A natural solution when dealing with theories axiomatized by Horn clauses (or conditional equations) is to use the negative literals of the axioms as cut formulas. This approach is frequently used in the context of conditional rewrite system when a conditional rule $C \Rightarrow l \rightarrow r$ may reduce a goal *A* to subgoals $C \Rightarrow A[r/l]$ (i.e. term / is replaced by r in *A)* and $\neg C \Rightarrow A$.

The problem is now that one of the subgoals is not smaller than the initial goal and a lot of control is needed to avoid divergence of the process since a similar case analysis can be applied again to $\neg C \Rightarrow A$. This has motivated us to introduce a new case analysis rule that allows one to split a goal *A* into subgoals $C_i \Rightarrow A[l_i/r_i]$ and $\vee_i C_i$. Since in the context of conditional rewrite system all subgoals are strictly smaller than the initial

goal the search space is much more controlled. Related approaches were proposed independently by [Bronsard and Reddy, 1990J and [Bevers, 1993]. However, since their inference systems are unable to handle non Horn clauses, they cannot prove the $\vee_i C_i$ formulas otherwise than by external means. On the contrary, our proof technique applies to non Horn clauses as well. The disjunction $\vee_i C_i$ is added to the other conjectures and does not require particular treatment. Therefore our setting is very homogeneous and permits one to extend our basic inference rules by various optimizations without loosing correctness and completeness. In particular we have a notion of inductive positions defining the subset of variables of a conjecture that can be instanciated by induction schemes and we have proved that these positions are the only ones needed for completeness. The restriction of induction to these positions reduces drastically the search space. The importance of such restrictions was recognized a long time ago by [Boyer and Moore, 1979].

The paper is organized as follows. In Section 2 we introduce the basic definitions about term rewriting. In Section 3 we define the notions of inductive theory and inductive rewriting, which is a fundamental tool for proving inductive theorems. We define in section 4 inductive positions and test sets. Section 5 presents our technique of simulating case reasoning by rewriting. This technique reduces considerably the number of inductive positions to be considered. The strategy can be embedded in a correct set of inference rules described in Section 6. When the axioms are ground convergent and the defined functions are completely defined then it is possible to apply the system for refuting conjectures (subsection 6.3). In Section 7, the strategy is even proved refutationally complete for conditional equations with boolean preconditions (under previous hypotheses). Computer experiments with SPIKE are discussed in Section 8.

2 Basic concepts

We assume that the reader is familiar with the basic notions of rewrite systems. We introduce the essential terminology below and refer to [Dershowitz and Jouannaud, 1990J for more detailed presentations.

2.1 Terms and substitutions

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. We assume that we have a partition of F in two subsets, the first one, C, contains the *constructor symbols* and the second, D, is the set of *defined symbols*.

Let X be a family of free 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.

For any term t, $occ(t) \subseteq N^*$ denotes its set of positions and the expression t/u denotes the *subterm of t at a position u*. We write $t[s]_u$ (resp. $t[s]$) to indicate that s is a subterm of t at position u (resp. at some position). The root position is written ϵ.

A Σ – *substitution* assigns Σ – *terms* of appropriate sorts to variables. Composition of substitution σ and η is written by $\sigma\eta$. The Σ – *term* $t\eta$ obtained by applying a substitution η to t is called an *instance* of t. If η is a ground substitution (i.e. $\eta(x)$ is ground for every x), we say that $t\eta$ is a *ground instance* of t.

2.2 Conditional Equations and Clauses

Let $\Sigma = (S, F)$ be a signature. A Σ-*equation* is a pair $e = e'$ where e, $e' \in T(F, X)$ are terms of the same sort. A *conditional* Σ-equation is either a Σ-equation or an expression of one of the following forms: $e_1 \wedge \ldots \wedge e_n \Rightarrow e$, or $e_1 \wedge \ldots \wedge e_n \Rightarrow$, or $\Rightarrow$ where $e, e_1, \ldots e_n$ are Σ-equations. Given a conditional Σ-equation , $e_1, \ldots e_n$ are the *conditions* and e is the *conclusion*. A Σ-clause is an expression of the form $\neg e_1 \vee \neg e_2 \vee \ldots \vee \neg e_n \vee e'_1 \vee \ldots \vee e'_m$. When Σ is clear from the context we omit the prefix Σ. A clause is *positive* if $\neg$ does not occur in it.

In this paper axioms are built from conditional equations and goals to be proved are clauses, i.e. disjunction of equational literals, since $=$ is the only predicate [1]. The symbol $\equiv$ is used for syntactic equality between two objects.

2.3 Rewrite Relations

2.3.1 Preliminaries

Given a binary relation $\rightarrow$, $\rightarrow^*$ denotes its reflexive and transitive closure. Let a and b be two terms, we say that $a \downarrow b$, if there exists c such that $a \rightarrow^* c$ and $b \rightarrow^* c$. A relation $\rightarrow$ is noetherian if there is no infinite sequence $t_1 \rightarrow t_2 \rightarrow \ldots$. In the following we suppose given a *reduction ordering* $\succ$ on the set of terms, that is, a transitive irreflexive relation that is noetherian, monotonic ($s \succ t$ implies $w[s] \succ w[t]$) and stable ($s \succ t$ implies $s\sigma \succ t\sigma$). We also assume that the ordering $\succ$ can be extended when adding new constants to the signature. The multiset extension of an ordering $\succ$ will be denoted by $\gg$.

A conditional equation $a_1 = b_1 \wedge \ldots a_n = b_n \Rightarrow s = t$ will be written as $a_1 = b_1 \wedge \ldots a_n = b_n \Rightarrow s \rightarrow t$ if $\{s\sigma\} \gg \{t\sigma, a_1\sigma, b_1\sigma, \ldots, a_n\sigma, b_n\sigma\}$ for all ground substitutions σ; in that case we say that $a_1 = b_1 \wedge \ldots a_n = b_n \Rightarrow s \rightarrow t$ is a *conditional rule*. The term s is the *left-hand side* of the rule. A set of conditional rules is a *rewrite system*.

2.3.2 Conditional Rewriting

The idea of rewriting is to impose a direction when using equations in proofs. This direction is indicated by an arrow when it is independent from the instantiation: $l \rightarrow r$ means that we can replace l by r in any context. When an instance of a conditional equation is orientable and has a valid conditional part it can be applied as a rule. The conditions are checked recursively. Termination is ensured by requiring the conditions to be smaller (w.r.t. the reduction ordering $\succ$) than the conclusion.

[1] we identify a conditional equation and its corresponding representation as a Horn clause

Definition 2.1 (Conditional Rewriting) *Let R be a set of conditional equations. Let a be a term and u a position in a. We write: $a[s\sigma]_u \rightarrow_R a[t\sigma]_u$ if there is a substitution σ and a conditional equation $\wedge_{i=1}^n a_i = b_i \Rightarrow s = t$ in R such that:*

1. $s\sigma \succ t\sigma$.
2. $\forall i \in [1..n]\ a_i\sigma \downarrow_R b_i\sigma$.
3. $\{a[s\sigma]_u\} \gg \{a_1\sigma,\ b_1\sigma,\ \ldots,\ a_n\sigma,\ b_n\sigma\}$.

A term t is reducible w.r.t. to $\rightarrow_R$ if there is a term t' such that $t \rightarrow_R t'$. Otherwise we say t is *R-irreducible*. The system R will be qualified as *ground convergent* if $\forall a, b \in T(F)$ $(R \models a = b$ *implies* $a \downarrow_R b)$.

Note that when R is a rewrite system the relation $\rightarrow_R$ is similar to the notion of decreasing rewriting of Dershowitz, Okada and Sivakumar [Dershowitz *et al.*, 1988].

2.3.3 Sufficient completeness

When for all possible arguments the result of a defined operator can be expressed with constructors only we say that this operator is completely defined w.r.t. the constructors. This requirement is very natural when building specifications in a structured way. Here is a more formal definition:

Definition 2.2 *Let C be a set of constructors, let R be a rewrite system and D be a set of defined operator. The operator $f \in D$ is completely defined w.r.t. C iff for all $t_1, \ldots, t_n$ in $T(C)$, there exists t in $T(C)$ such that $f(t_1, \ldots, t_n) \rightarrow_R^* t$.*

Although this property is in general undecidable, SPIKE offers facilities to check and complete definitions.

3 Induction

3.1 Inductive theory

Given a set of conditional equations Ax, the theory of Ax, which is the one we are interested in, is the class of sentences that are true in the minimal Herbrand model of Ax. Every element in the domain of a Herbrand model is denoted by a ground term built on the signature of Ax. But since ground terms can be well-ordered, induction is available as a natural technique to prove sentences in this model. We call *inductive consequence* of Ax any sentence C that is valid in the minimal Herbrand model of Ax and we denote this by $Ax \models_{ind} C$. In the following we assume that Ax can be oriented as a rewrite system R (w.r.t. $\succ$).

3.2 Inductive rewriting

To simplify goals we extend the conditional rewriting relation 2.1, so that we can check the conditions of a rule to be applied to a clause C with inductive hypothesis, other conjectures and the premisses of C, considered as an implication formula.

Let us first introduce a few notations. Let $C \equiv \neg(a_1 = b_1) \vee \ldots \vee \neg(a_n = b_n) \vee (c_1 = d_1) \vee \ldots \vee (c_m = d_m)$. Then we denote by $prem(C)$ the set $\{a_i = b_i\}_{i=1,n}$. The expression $(a = b)^\epsilon$ denotes the literal $a = b$ if $\epsilon = +$ and the literal $\neg(a = b)$ if $\epsilon = -$. The skolemized clause $\overline{C}$ of C is the clause obtained by substituting every variable of C by a new constant.

Definition 3.1 (Inductive rewriting) *Let R and W be two sets 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: $a \rightarrow_{R[W],r} a'$ if $a \succ a'$ and:*
either $\overline{a} \rightarrow_{prem(\overline{r})} \overline{a'}$,
or *there exists a position u in a, a substitution σ and a conditional equation $\wedge_{i=1}^n a_i = b_i \Rightarrow s = t$ in R such that:*

1. $a \equiv a[s\sigma]_u$ *and* $a' \equiv a[t\sigma]_u$.
2. $\{a[s\sigma]_u\} \gg \{a_1\sigma,\ b_1\sigma,\ \ldots,\ a_n\sigma,\ b_n\sigma\}$.
3. $\forall i \in [1..n]$ $\exists c_i', d_i'$ *such that* $a_i\sigma \rightarrow_{R \cup W\ [\ ],r}^* c_i'$ *and* $b_i\sigma \rightarrow_{R \cup W\ [\ ],r}^* d_i'$ *and* $\overline{c_i'} =_{prem(\overline{r})} \overline{d_i'}$.

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

The set W in the definition is intended to contain induction hypotheses in the proof system described below.

Inductive rewriting can be viewed as a powerful generalization of both the rewriting relation defined in [Kounalis and Rusinowitch, 1990b] and contextual rewriting [Zhang, 1993].

4 Selection of Induction Schemes

To perform a proof by induction, it is necessary to provide some induction schemes. In our framework these schemes are defined first by a function which, given a conjecture, selects the positions of variables where induction will be applied and second by a special set of terms called a test set. In general the selection of good inductive positions leads to drastic improvements. We shall not discuss test sets here but just recall their main properties.

Let us consider first the problem of choosing the positions where variables need to be instanciated by induction schemes. In order to define the set of these variables, we introduce $VL(s)$, the set of linear variables of a term s and $Def(f)$, the set of terms with root symbol f.

Definition 4.1 *Given a rewrite system R on $T(F, X)$ the set of inductive positions for a function symbol f is: $Occ\_ind(R, f) = \{u/$ there exists $p \Rightarrow g \rightarrow d \in R$ such that $g \in Def(f)$, $u \in Occ(g) \setminus \{\epsilon\}$ and $g/u \notin VL(g)\}$.*

Given a term s, an *induction variable* of s is a variable that occurs at a position $u.v$ of s such that v is an inductive position of the root of t/u (if s is a variable then it is considered as an induction variable). Given a term s and a set of terms TS, σ is a *TS-substitution* for s, if it maps any induction variable of s to an element of TS of the same sort. When no subterm of t matches a left-hand side of the rewrite system R we say that t is *strongly R-irreducible*.

Proposition 4.1 *If R is a rewrite system, then a* test set $S(R)$ for R *is a finite set of R-irreducible terms that has the following properties:*

a. for any R-irreducible ground term s there exists a term t in S(R) and a ground substitution a such that $t\sigma = s$;

b. let s be a term and let a be a S(R)-substitution for s; if sa is strongly R-irreducible then there is a ground instance sap such that sap is strongly R-irreducible.

The first property allows us to prove theorems by induction on the domain of irreducible terms rather than on the whole set of terms. Sets of terms with the property a. are usually called cover sets in the literature [Reddy, 1990; Zhang et a/., 1988]. However they cannot be used to refute theorems. The second property b. of test sets is fundamental for this purpose.

It is possible to compute test sets for equational theories (see [Kounalis, 1990; Huber, 1991]). Unfortunately no algorithm exists for the general case of conditional theories. However, in [Kounalis and Rusinowitch, 1990b] a method is described for computing test sets in conditional theories defined over & free set of constructors.

The role of test sets for refutation is shown by the following definition that gives a falsity criteria for positive clauses:

Definition 4.2 *Suppose that we are given a rewrite system R and a test set $S(R)$. Then a clause $C \equiv g_1 = d_1 \vee \ldots \vee g_n = d_n$ is quasi-inconsistent with respect to R if there is a $S(R)$-substitution σ of C such that for all $1 \leq j \leq n$, $g_j\sigma \not\equiv d_j\sigma$ and the maximal elements of $\{g_j\sigma, d_j\sigma\}$ w.r.t. $\succ$ are strongly R-irreducible.*

We recall the next theorem from [Kounalis and Rusinowitch, 1990a]. In its proof, the condition b of proposition 4.1 plays a crucial role.

Theorem 4.1 Let R be a ground convergent rewrite system. If a positive clause C is quasi-inconsistent then C is not an inductive consequence of R.

5 Automatic Case Analysis

We shall introduce now the Case rewriting relation that allows one to reduce goals with conditional rules without attempting to check their preconditions. The preconditions are appended to the goal as a context. Case rewriting can be viewed as an implementation of case analysis that is well adapted to the given conditional axioms. We have found this rule absolutely necessary for proving non trivial conjectures with our automatic system. Moreover it is the basis of a refutationally complete system for boolean systems. In this section we first discuss the problems with former case rewriting rules. Then we propose a new rule that is easier to automate and has given much better results on experiments. A first version of case rewriting was proposed in [Kounalis and Rusinowitch, 1990b]. Given a term t and a rule $p \Rightarrow g \rightarrow d$ such that g matches a subterm of t with substitution a, this rewriting is expressed by the following inference rule :

$$l[g\sigma]_u \vee r \vdash l[d\sigma]_u \vee \neg p\sigma \vee r,\ l[g\sigma]_u \vee p\sigma \vee r$$

However an important control problem with this technique lies in the choice of the rule to apply during the proof by induction. This can be illustrated by the next example.

Example 5.1 *The following rules define odd and even for nonnegative integers. A test set here is $\{0, s(0), s(s(x)), T, F\}$.*

$$even(0) \rightarrow T \quad (1)$$
$$even(s(0)) \rightarrow F \quad (2)$$
$$even(s(s(x))) \rightarrow even(x) \quad (3)$$
$$even(x) = T \Rightarrow odd(x) \rightarrow F \quad (4)$$
$$even(x) = F \Rightarrow odd(x) \rightarrow T \quad (5)$$
$$T = F \Rightarrow \quad (6)$$

Let us prove:

$$even(s(x)) = T \vee odd(x) = F \quad (7)$$

by the case rewriting rule above. Clause 7 can be split according to axiom 4 in the two following clauses:

$$even(x) = T \Rightarrow even(s(x)) = T \vee F = F \quad (8)$$
$$\neg(even(x) = T) \Rightarrow even(s(x)) = T \vee odd(x) = F \quad (9)$$

8 is a tautology and 9 is equivalent to:

$$even(x) = T \vee even(s(x)) = T \vee odd(x) = F \quad (10)$$

Now splitting clause 10 with axiom 5 yields:

$$even(x) = F \Rightarrow even(x) = T \vee even(s(x)) = T \vee T = F \quad (11)$$
$$\neg(even(x) = F) \Rightarrow even(x) = T \vee even(s(x)) = T \vee odd(x) = F \quad (12)$$

11 simplifies to:

$$even(x) = F \Rightarrow even(x) = T \vee even(s(x)) = T \quad (13)$$

that can be proved an inductive consequence of R. 12 is equivalent to:

$$even(x) = F \vee even(x) = T \vee even(s(x)) = T \vee odd(x) = F \quad (14)$$

Note that the same case analysis can be applied infinitely often to 14. A possible way to avoid divergence is to limit application of the case analysis rule. Case rewriting in [Kounalis and Rusinowitch, 1990b] is controlled by conditions for avoiding infinite applications of the same rule. However, when adding these technical conditions the proof of 14 diverges.

These problems have motivated us to introduce a new case rewriting technique that rewrites a term t simultaneously to several terms $t_1, \ldots, t_n$ each reduction being respectively valid in different contexts $c_1, \ldots, c_n$. In other words, given a term t, we consider all the way to rewrite it w.r.t. to axioms and positions. We must then prove that the disjunction DP of the conditions of the applied rules is inductively valid. Note that DP is usually a non Horn clause. Our approach to inductive proofs is non hierarchical: we can prove DP by simply adding it to the set of conjectures to be further processed.

Definition 5.1 (Case rewriting) *Let R be a rewrite system and $C \equiv (a = b)^e \vee r$ be a clause. We define G as the set $\{< a[d\sigma]_u, P\sigma >$; there exists $P \Rightarrow g \rightarrow d$ in R, a position u in a such that $a/u = g\sigma$, $g\sigma$ is R-irreducible and does not contain an inductive variable$\}$. Then Case_rewriting $((a = b)^e, r)$ is the following set of clauses:*

$$\{\neg P \vee (a' = b)^e \vee r;\ < a', P > \in G\} \cup \{\bigvee_{<a',P> \in G} P\}$$

Example 5.2 (example 5.1 continued)
With our method, the proof of 7 is as follows: we apply case rewriting to get :

$$even(x) = T \Rightarrow even(s(x)) = T \vee F = F \quad (15)$$
$$even(x) = F \Rightarrow even(s(x)) = T \vee T = F \quad (16)$$

We must prove:

$$even(x) = T \vee even(x) = F \quad (17)$$

15 is a tautology, 16 is simplified by R into:

$$even(x) = F \Rightarrow even(s(x)) = T \quad (18)$$

The instances of 18 by elements 0 and $s(0)$ from the test set yield clauses that are simplified by R and subsumed by an axiom. The instance of 18 by $s(s(y))$ gives a clause that is simplified by R and subsumed by 18 which has become an induction hypothesis. In the same way 17 can easily be proved.

Example 5.3 *Consider the system* $R = \{p(x,0,z) = T,\ p(x,s(y),z) = p(x,y,z),\ p(y,x,z) = T \Rightarrow f(x,y,z) = 0\}$. *To prove* $f(x,y,z) = 0$ *with the method of [Kounalis and Rusinowitch, 1990a], we instanciate* x,y *and* z *by* 0 *and* $s(x)$ *(from* $S(R)$*) in all possible ways. We obtain* 8 *equations and the proof of some of them diverges. With the method we present here, thanks to case rewriting, we do not need to consider all these inductive positions. We have:* $Occ\_ind(p) = \{2\}$ *and* $Occ\_ind(f) = \emptyset$. *To prove* $f(x,y,z) = 0$, *we apply case rewriting to get* $p(y,x,z) = T \Rightarrow 0 = 0$ *and* $p(y,x,z) = T$. *The first clause is a tautology and the second one is proved by instanciating* x *by* 0 *and* $s(x)$.

Other authors have applied case rewriting techniques for inductive theorem proving. Among them [Bronsard and Reddy, 1990] and [Bevers, 1993] propose an approach related to ours but their methods cannot be considered as automatic since they cannot check the applicability of case rewriting rules due to the fact that their provers are restricted to Horn clauses.

To conclude, we think that our new case rewriting rule avoids many drawbacks of the previously defined ones. It allows to prove a larger class of theorems. Last but not least it does not require human interaction or call to an external theorem prover.

6 A Proof Procedure for Conditional Theories

6.1 Inferences rules

We present the procedure for proof by induction as a set of inference rules to be applied fairly to the goals. Let R be a rewrite system. The procedure modifies incrementally two sets of clauses E and H, where E contains the conjectures to be checked and H contains clauses, previously in E, that have been reduced and can therefore be used as inductive hypotheses. This procedure is refutational in essence, and performs implicit induction w.r.t. to $\succ$. Its correctness is obtained by very simple arguments about the existence of a minimal counterexample. We think that our correctness proof is much simpler than the related ones [Reddy, 1990]. As a consequence it was easy for us to add many optimizations to the procedure

generate: $(E \cup \{C_E\}, H) \vdash_I (E \cup (\cup_\sigma E_\sigma), H \cup \{C_E\})$
if $C_E \equiv (a = b)^e \vee r$ and for every $S(R)$-substitution σ of C_E:
either $C_E\sigma$ is a tautology and $E_\sigma = \emptyset$
or $a\sigma \rightarrow_{R[H \cup E];, e\sigma} a'$ and $E_\sigma = \{(a' = b\sigma)^e \vee r\sigma\}$
otherwise $E_\sigma = case\_rewriting((a = b)^e\sigma, r\sigma)$

case simplify: $(E \cup \{(a=b)^e \vee r\}, H) \vdash_I (E \cup E', H)$
if $E' = case\_rewriting((a = b)^e, r)$

simplify: $(E \cup \{(a=b)^e \vee r\}, H) \vdash_I (E \cup \{(a'=b)^e \vee r\}, H)$
if $a \rightarrow_{R[H \cup E];, r} a'$ or $a[s]_u \rightarrow_{H \cup E[R];, r} a' \equiv a[t]_u$ and $u \neq \epsilon$

complement: $(E \cup \{\neg(a\sigma = b\sigma) \vee r\}, H) \vdash_I (E \cup \{(a\sigma = b'\sigma) \vee r\}, H)$
if $\neg(b = b') \in R$, $a = b \vee a = b' \in E \cup H$ and $b\sigma \succeq b'\sigma$.

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

fail: $(E, H) \vdash_I \Box$
if no other rule applies to (E, H)

Figure 1: Inference System I

and show that they do not affect correctness. The inference system for induction I contains the rules given in figure 1.The *generate* rule allows to derive lemmas and initiates induction steps. The *case simplify* rule simplifies a conjecture with conditional rules and adds to the result the contexts where the respective reductions are valid. The *simplify* rule reduces a clause C with axioms from R, induction hypotheses from H and other conjectures (therefore we can simulate simultaneous induction). The premisses of C considered as a conditional axiom can also help to check that the preconditions of a rule being applied to C are valid. The *complement* rule transforms negative clauses to positive clauses that are easier to refute. The role of *deletion* is obvious. The *fail* rule is applied to (E, H) if no other rule can be applied to (E, H).

An I-derivation is a sequence of states:

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

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

6.2 Correctness of the procedure

The correctness of I is obtained by defining a well-founded ordering on clauses 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 $(E_0, H_0) \vdash_I (E_1, H_1) \vdash_I \ldots$ is *fair* if either it fails or the set of persisting clauses $(\cup_i \cap_{j \geq i} E_j)$ is empty. Then we reason by contradiction: if a non valid clause is generated in a non failed 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. Hence we have:

Theorem 6.1 (correctness) *Let R be a rewrite system and let $(E_0, \emptyset) \vdash_I (E_1, H_1) \vdash_I \ldots$ be a fair I-derivation. If it does not fail then $R \models_{ind} E_0$.*

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

6.3 Refutation of conjectures

In this subsection R is a ground convergent rewrite system such that all its defined symbols are completely defined. From now on, we assume that every left-hand side of a conditional rule has a defined symbol. Let us call J the set of inference rules obtained by adding to I the rule:

disproof: $(E \cup \{c_E\}, H) \vdash_J$ *Disproof*

if c_E is positive and **generate** cannot be applied to $(\{c_E\}, H)$.

The inference system J allows to refute conjectures: If disproof is applied at some step k then a quasi-inconsistent clause is detected and therefore, from theorem 4.1, we can conclude that some conjecture in E_k is false. Then by induction on the number of steps (k), this implies that some conjecture from E_0 is false too. Hence we have:

Theorem 6.2 *Let R be a ground convergent rewrite system such that every defined symbol is completely defined and let $(E_0, \emptyset) \vdash_J (E_1, H_1) \vdash_J \ldots$ be a J-derivation. If there exists j such that* disproof *applies to (E_j, H_j) then $R \not\models E_0$.*

7 A refutationally complete procedure for theories with boolean preconditions

In this section we shall consider axioms that are conditional rules with boolean preconditions. More precisely we assume there exists a sort *bool* with two nullary free constructors $\{T, F\}$. Every rule in R is of type: $\wedge_{i=1}^{n} p_i = p_i' \Rightarrow s \to t$ where for all i in $[1 \ldots n]$, $p_i' \in \{T, F\}$. Such a system R is called a *boolean rewrite system*. For $\alpha \in \{T, F\}$ we denote by α^- the complementary *bool* symbol of α. Conjectures will be *boolean clauses*, that is clauses whose negative literals are of type $\neg(p = p')$ where $p' \in \{T, F\}$.

We also assume that any function symbol p with boolean values is completely defined. In other words, the following is inductively valid:

$$p(\vec{x}) = True \vee p(\vec{x}) = False$$

We can then define a new inference system K from I by reformulating the *complement* as follows:

complement: $(E \cup \{\neg(\alpha = \alpha) \vee r\}, H) \vdash_K (E \cup \{(\alpha = \alpha^-) \vee r\}, H)$

if $\alpha \in \{T, F\}$

and replacing the *fail* rule by:

disproof: $(E, H) \vdash_K$ *Disproof*

if no other rule applies to (E, H)

A K-derivation *fails* if it ends with *disproof*. Note first that the only rule that permits to introduce negative clauses is *case-rewriting*. Since the axioms have boolean preconditions, all the clauses generated in a A' derivation (such that *Eo* only contains boolean clauses) are boolean. So the new inference system *K* can be proved refutationally complete for boolean clauses too.

Theorem 7.1 *Let R be a ground convergent boolean rewrite system such that every defined symbol is completely defined and let $(E_0, \emptyset) \vdash_K (E_1, H_1) \vdash_K \ldots$ be a fair K-derivation such that E_0 only contains boolean clauses. Then $R \not\models_{ind} E_0$ iff the derivation fails.*

8 Computer Experiments

Our prototype SPIKE (written in Caml Light) is designed to prove the validity of a set of clauses in a conditional theory. The first step in a proof session is to check if all defined functions are completely defined. If this step is successful then we can use a more efficient version of the case rewriting rule. The second step is to check the ground convergence of the set of axioms. If the first two steps are successful then we can refute false conjectures. The third step is to compute test sets and inductive positions. After these preliminary tasks (detailed in [Bouhoula *et* al., 1992a]), the proof starts.

Example 8.1 *Consider the specification of lists of natural numbers with an "insert" operation and a "sorted" predicate on lists that is true iff a list is ordered. SPIKE can prove the conjecture $sorted(insert(x, l)) = sorted(l)$ in a completely automatic way. Note that RRL [Zhang* et al.*, 1988] is unable to succeed with this example unless the user suggests some well-chosen lemmas.*

Example 8.2 *An interactive proof of Gilbreath Cards Trick was first given by G. Huet [Huet, 199l] using the COQ proof assistant. B. Boyer has used NQTHM to derive another proof. A similar but much faster proof was obtained by H. Zhang with RRL. These two "automatic" proofs require many lemmas, some of them being non-obvious. For instance, B. Boyer introduces a predicate "silly" that is "only defined to force a certain weird induction" (here we quote B. Boyer). The same predicate appears in Zhang's experiment. On the other hand, our proof is based on 5 lemmas easy to understand. These lemmas have been suggested by a first unsuccessful proof attempt with SPIKE. The source of failure was identified by the impossibility of reducing a family of patterns. Hence we have introduced the adequate lemmas for simplifying them. This was enough to derive a proof*

The proof has taken more real time than Boyer's (and Zhang's). However, it is difficult to compare these approaches from the efficiency point of view, since we have spent much less time to get the right lemmas. On the other hand, the differences between programming languages lead also to some discrepancy in the performance.

The following array compare users inputs in the proof of Gilbreath Card Trick for NQTHM, RRL and SPIKE:

	NQRHM	RRL	SPIKE
! Explicit definitions	6	8	9
Implicit definitions	2	1	0
Induction scheme definition	1	1	0
Lemmas	17	23	5

9 Conclusion

We have proposed a new procedure for proof by induction in conditional theories. Our procedure relies on the implicit induction paradigm and puts the stress on simplification and case analysis. As our previous procedure [Bouhoula et al, 1992a], it allows simplification of conjectures by conjectures and has been extended to handle non-orientable equations. It can also refute non valid conjectures. A meaningful contribution of this paper is that our strategy is refutationally complete for a class of rewrite systems that can specify numerous interesting examples. This class contains the boolean ground convergent rewrite systems with completely defined functions over free constructors. In other words with our procedure every false conjecture will be disproved in finite time. However, our method remains valid even when the functions are not completely defined. Note that our correctness and completeness proofs do not require an elaborated notion of fairness. It is compatible too with many simplification rules that were not discussed here for lack of space. An extension to theories that are presented by non Horn clauses is also easy.

The method is entirely implemented in the prover SPIKE. This system has proved non trivial examples in a completely automatic way, that is, without interaction with the user and without ad-hoc heuristics. It has also proved the challenging Gilbreath card trick, with only 5 lemmas.

We plan to enhance the system with recent techniques developed for interactive provers such as the ones in [Bundy et ai, 1989; Chadha and Plaisted, 1992].

Acknowledgement: We thank Helene Kirchner for her comments.

References

[Bevers, 1993] E. Bevers. A utomated Reasonning in Conditional Algebric Specifications: Termination and Proof by Consistency. PhD thesis, Katholieke Universiteit Leuven, Belgium, 1993.

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

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

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

[Bronsard and Reddy, 1990] F. Bronsard and S. Reddy. Conditional rewriting in focus. In S. Kaplan and M. Okada, editors, Proceedings of the 2nd Workshop on Conditional and Typed Rewriting Systems, volume 516 of LNCS. Springer-Verlag, 1990.

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

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

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

[Dershowitz et ai, 1988] N. Dershowitz, M. Okada, and G. Sivakumar. Canonical conditional rewrite systems. In Proceedings 9th International Conference on Automated Deduction, Argonne (III., USA), volume 310 of LNCS. Springer-Verlag, May 1988.

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

[Huber, 1991] M. Huber. Test-set approaches for ground reducibility in term rewriting systems, characterizations and new applications. Master's thesis, Technische Universitat Berlin, 1991.

[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.

[Huet, 1991] G. Huet. The Gilbreath trick: a case study in axiomatisation and proof development in the 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.

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

[Kounalis, 1990] E. Kounalis. Testing for inductive (co)-reducibility. In A. Arnold, editor, Proceedings 15th CAAP, Copenhagen (Denmark), volume 431 of LNCS, pages 221 — 238. Springer-Verlag, May 1990.

[Musser, 1980] D. R. Musser. On proving inductive properties of abstract data types. In Proceedings 1th ACM Symp. on Principles of Programming Languages, pages 154 162. Association for Computing Machinery, 1980.

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

[Zhang et ai, 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 (III, USA), volume 310 of LNCS, pages 162-181. Springer-Verlag, 1988.

[Zhang, 1993] II. Zhang Implementing Contextual Rewriting Proceedings of the 3rd CTRS Workshop, volume 656 of LNCS. Springer-Verlag, 1993.