



Automated mathematical induction

Adel Bouhoula, E. Kounalis, Michaël Rusinowitch

► To cite this version:

Adel Bouhoula, E. Kounalis, Michaël Rusinowitch. Automated mathematical induction. [Research Report] RR-2187, INRIA. 1994, pp.42. inria-00074484

HAL Id: inria-00074484

<https://inria.hal.science/inria-00074484>

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Automated Mathematical Induction

Adel BOUHOULA
Emmanuel KOUNALIS - Michaël RUSINOWITCH

N° 2187
Février 1994

PROGRAMME 2

Calcul symbolique,
programmation
et génie logiciel

*Rapport
de recherche*

1994

Automated Mathematical Induction *

Adel Bouhoula [†] Emmanuel Kounalis [‡] Michaël Rusinowitch [†]

[†] CRIN & INRIA Lorraine, BP239, 54506 Vandœuvre-lès-Nancy, France

[‡] Laboratoire d'Informatique I3S, 250 Avenue A. Einstein, 06560 Valbonne, France

Abstract

Proofs by induction are important in many computer science and artificial intelligence applications, in particular, in program verification and specification systems. We present a new method to prove (and disprove) automatically inductive properties. Given a set of axioms, a well-suited induction scheme is constructed automatically. We call such an induction scheme a test set. Then, for proving a property, we just instantiate it with terms from the test set and apply pure algebraic simplification to the result. This method needs no completion and explicit induction. However it retains their positive features, namely, the completeness of the former and the robustness of the latter. It has been implemented in the theorem-prover SPIKE ¹.

keywords: Theorem proving, Term rewriting systems, Implicit induction, Conditional theories.

Preuves automatiques par récurrence

Résumé

Nous proposons une nouvelle méthode de preuve (et de réfutation) automatique dans le modèle initial d'une théorie conditionnelle. Etant donné un ensemble d'axiomes, un schéma de récurrence, appelé *ensemble test* est construit automatiquement. Une propriété est prouvée en l'instanciant par les éléments de l'*ensemble test* et en lui appliquant des technique de simplification. Cette méthode nécessite ni complétion ni récurrence explicite. Nous montrons sa correction en général et sa complétude dans le cas équationnel. Notre méthode est implantée dans le prouveur SPIKE.

mots clés: Preuve automatique, Système de réécriture, Récurrence implicite, Théories conditionnelles.

*This is a new version of Technical Report 1636, INRIA, 1992.

¹Preliminary versions of the results in this paper have been presented at the *First International Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale, Florida, January 1990* and in the *Bulletin of European Association for Theoretical Computer Science*, 41:216-226, June 1990.

Contents

1	Introduction	3
1.1	Motivation	3
1.2	Highlights of the paper and related works	4
1.3	Layout of the paper	5
2	Overview of our approach on examples	6
3	Preliminaries	8
3.1	Conditional theories	8
3.1.1	Terms and substitutions	8
3.1.2	Conditional equations and clauses	9
3.1.3	Inductive theory	10
3.2	Rewrite Relations	10
3.2.1	Conditional rewriting	11
3.2.2	Case rewriting	12
3.2.3	Supported rewriting	12
3.2.4	Relaxed rewriting	12
4	How to prove and disprove inductive theorems	13
4.1	Test sets	13
4.2	Inductive proofs by simplification	15
4.3	Disproving inductive conjectures	17
4.4	A general procedure for proof by induction	20
4.4.1	Inference rules for inductive proofs	20
4.4.2	Refutation of conjectures	23
5	How to get the convergence property	24
5.1	The saturation technique	24
5.2	Hierarchical axiomatization techniques	25
6	How to get test sets	26
7	A detailed proof session	31
8	Conclusion	35

1 Introduction

1.1 Motivation

Inductive reasoning is simply a method of performing inferences in domains where there exists a well-founded relation on the objects. It is fundamental when proving properties of numbers, data-structures, or programs axiomatized by a set of equations and or conditional axioms. The use of conditional axioms as definitions is well known in Computer Science, since programs in logical or functional style and data-structures can be expressed in this framework. As opposed to deductive theorems, inductive theorems are usually valid only in some particular models of the axioms. For instance, Herbrand models or initial models fit nicely with the semantics of data-type specifications, logic and functional programming.

As a classical example, consider the data-structure of nonnegative integers built up using the 0 and S function symbols. Every element of this structure can be represented by a variable-free (ground) term that involves 0 and S only. Suppose now we define the addition operation $+$ by the following axioms:

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

Clearly, adding two integers, using the above equations, yields a nonnegative integer. For instance, $S(S(S(0))) + S(S(0))$ equates to $S(S(S(S(S(0)))))$ by deductive reasoning: apply just twice the second axiom and once the first. Consider now the property of associativity of $+$:

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

This is a typical example of an identity whose proof requires some kind of induction. As everybody knows from his experience, it might be difficult, not only to find an appropriate well-founded relation to support inductive inferences, but also to guess suitable induction hypothesis. Two main approaches have been proposed to overcome these difficulties. The first applies explicit induction arguments on the structure of terms [Aub79, Bur69, BM79, BHHW86, GG89, ZKK88]. The second one involves a proof by consistency: this is the *inductionless induction* method [Mus80, HH82, KM87, JK86, Fri86, KNZ86, Bac88, Küc89, BK89, Gra89]. To prove the associativity of $+$ by explicit induction we can use the following scheme:

Basic Case: $(x + y) + 0 = x + (y + 0)$

Induction Step: $(x + y) + Su = x + (y + Su)$

The proof of Basic Case is trivial, and the proof of Induction Step follows immediately from the defining axioms and the induction hypothesis $(x + y) + u = x + (y + u)$. On the other hand, to prove the same property by inductionless induction we first try to compile the axioms into a convergent (i.e., terminating and confluent) set of rewrite rules. In the above example it is sufficient to orient the axioms from left to right. Then

the associativity is added as a rule oriented from right to left and a completion-like procedure is started in order to transform the whole set of rules into a convergent one by superposing left-hand sides. Here, all superpositions lead to trivial identities and therefore the associativity of $+$ is proved.

However, both methods have many limitations, either on the theorems that could be proved or on the underlying theory. For instance, explicit induction techniques need a hierarchical structure for managing the subgoals in order to avoid ill-founded induction steps. This point is crucial for handling mutually recursive definitions. It was only recently that some advances have been made concerning this problematic aspect of explicit induction. Also explicit induction can only refute trivially unsatisfiable problems [Pro92]. Our method allows us to detect more general false conjectures. Moreover we have refutational completeness results for non conditional theories.

Note also that guiding a proof by explicit induction requires some skill for finding the right axioms or hypothesis to apply. On the other side, the inductionless induction technique often fails where explicit induction succeeds. If the associativity of $+$ were oriented from left to right in the example this method fails to prove it, since the completion procedure loops in this case. Moreover, there does not exist any realistic inductionless induction procedure for conditional theories. However the main advantage of inductionless induction is that it does not require to build an hierarchy of lemmas to be proved. All the conjectures and intermediate results are considered at the same level (and proved in a same round) without any notion of hierarchy.

In this paper we present an alternative proof system for automatizing inductive reasoning in theories defined by conditional axioms. We show how to prove (and disprove) equations and, more generally, clauses in Herbrand models. This proof system combines the power of explicit induction and inductionless induction.

1.2 Highlights of the paper and related works

The presented method relies on the notion of *test sets* (which, in essence, is a finite description of the initial model) and uses only pure algebraic simplification. The key-idea of the simplification strategy is to use axioms, previously proved conjectures, and instances of the conjecture itself as soon as they are smaller than the currently considered proposition with respect to a well-founded relation. This last point captures the notion of Induction Hypothesis in the proof by induction paradigm. The main observation is that when the axioms are oriented as a terminating set of rules, they provide a natural well-founded ordering which can be used to support induction.

The main arguments in favour of our method are:

- it works even when some functions are incompletely defined and when there are relations between constructors which is not the case for the Boyer-Moore system;
- for its correctness, it does not require the given theory to be turned into a convergent set of axioms;
- it provides automatically induction schemes through algorithms for computing test sets;

- it allows one to refute false conjectures when the axioms are presented by a ground convergent conditional rewrite system; it is also refutationally complete in the following sense : any equation that is not valid in the initial model will be disproved, provided that the axioms can be turned into a ground convergent unconditional rewrite system;
- it does not require any hierarchy for handling the induction hypothesis;
- it is not restricted to equational theories but also applies to conditional theories;
- it is not restricted to the proof (and disproof) of equations but also applies to non-orientable equations and general clauses.

Hofbauer and Kutsche [HK88] were the first to notice that inductionless induction techniques still work when the ground confluence property of the input set of equations is relaxed. However their procedure is still influenced by the completion framework since it is based on computing critical pairs and testing conjectures for ground reducibility. In the procedure we propose here, and whose first version appeared in january 1990 in [KR90a], we rather combine these two steps through the use of test sets. Our approach is developed for general *conditional* axiomatizations and do not require either the ground convergence property for correctness. Moreover, due to the property of test sets, it can be easily proved to be refutationally complete when the theory is presented by a ground convergent set of equational rules. Reddy presented a related method for the *nonconditional* case in july 1990 [Red90]. He used a notion of *covering sets* developed by [ZKK88], analogous to test sets, for computing induction schemes. However, no procedure is known for deriving such covering sets. This is not surprising since for any equational theory, a set of variables is always a covering set (and such a set is particularly useless for induction). We do not understand how the adepts of *covering sets* assert to perform induction *automatically* without providing a constructive definition of induction schemata.

On the other hand, we give here, following [KR90b, KR90c], a procedure to obtain test sets even in the conditional case. Moreover, the notion of covering set cannot be used to disprove false conjectures. Our proof of refutational completeness is self-contained and can be extended to many strategies. For instance, we allow simplification of conjectures by other conjectures, which is a fundamental feature for efficiency as shown by computer experiments.

1.3 Layout of the paper

The structure of the paper is as follows. In Section 2 we present our inference system on a simple example. In Section 3 we introduce the essential notions used throughout this paper. In particular, we define some useful rewriting relations. In Section 4 we provide the basic theorems on proving and disproving inductive conjectures. In Section 5 we give a general inference system to perform induction and show its correctness. We also prove its refutational completeness for convergent equational theories. In Section 6 we introduce some methods to get a completely operational proof system.

In particular, we show how the convergence property (required for the refutational completeness, but not for correctness) can be obtained either by a Knuth-Bendix like procedure or by semantic techniques specific to hierarchical axiomatizations. Whereas the computation of test sets is generally undecidable, in this section we also propose a method to obtain test sets in conditional theories over a free set of constructors. The Section 7 describes a proof session with our SPIKE software.

2 Overview of our approach on examples

Before discussing technical details of the method that we propose for mechanizing proofs of inductive theorems, we first describe our inference system on a simple example, namely, positive integers with cut-off and *gcd* functions and the *less* predicate. The arrow \rightarrow just indicates how to apply a (conditional) equation for simplification:

$$x - 0 \rightarrow x \quad (1)$$

$$0 - x \rightarrow 0 \quad (2)$$

$$\text{succ}(x) - \text{succ}(y) \rightarrow x - y \quad (3)$$

$$0 < \text{succ}(x) \rightarrow \text{true} \quad (4)$$

$$x < 0 \rightarrow \text{false} \quad (5)$$

$$x < y = \text{true} \Rightarrow \text{succ}(x) < \text{succ}(y) \rightarrow \text{true} \quad (6)$$

$$x < y = \text{false} \Rightarrow \text{succ}(x) < \text{succ}(y) \rightarrow \text{false} \quad (7)$$

$$x < y = \text{true} \Rightarrow \text{gcd}(\text{succ}(x), \text{succ}(y)) \rightarrow \text{gcd}(\text{succ}(x), y - x) \quad (8)$$

$$x < y = \text{false} \Rightarrow \text{gcd}(\text{succ}(x), \text{succ}(y)) \rightarrow \text{gcd}(x - y, \text{succ}(y)) \quad (9)$$

$$\text{gcd}(x, 0) \rightarrow x \quad (10)$$

$$\text{gcd}(0, x) \rightarrow x \quad (11)$$

Consider the conjectures:

$$x - x = 0 \quad (12)$$

$$x < x = \text{false} \quad (13)$$

$$x < \text{succ}(x) = \text{true} \quad (14)$$

$$x < y = \text{true} \vee x < y = \text{false} \quad (15)$$

$$x < y = \text{false} \vee y < z = \text{false} \vee x < z = \text{true} \quad (16)$$

$$\text{gcd}(x, x) = x \quad (17)$$

$$x < \text{succ}(x) = \text{false} \quad (18)$$

$$x < y = \text{false} \vee y < x = \text{true} \quad (19)$$

Except the two last ones all these propositions are valid in the standard arithmetic. Note that 13,15 and 16 state that $<$ is a total ordering on integers. Let us prove them by induction. With our method the first step consists of computing a test set (see definition 4.1). Using techniques of section 6, we get the following test set:

$\{0, succ(x), true, false\}$. The next step consists in replacing variables of the conjecture by the elements of the test set and checking these instances using pure simplification.

The simplification strategy may use axioms, previously proved conjectures, and instances of the conjecture itself as long as they are smaller (w.r.t. a noetherian relation that contains the rewriting relation) than the currently considered proposition. This last point captures the notion of Induction Hypothesis in the proof by induction paradigm (see theorem 4.1, 4.2).

For the equation 12 two instances need to be checked: $0 - 0 = 0$ and $succ(x) - succ(x) = 0$. The first one reduces immediately to a trivial identity. For the second one consider the reduction (notice the use of 12 as an induction hypothesis):

$$succ(x) - succ(x) \mapsto_3 x - x \mapsto_{12} 0$$

For 13 the only non-trivial instance is $succ(x) < succ(x) = false$

$$succ(x) < succ(x) \mapsto_{7[13]} false$$

For the last derivation, we have used an induction hypothesis to satisfy the condition of 7 (cf., theorem 4.1). For 14 the same argument can be employed. For 15 there are four instances by test set. The only non-trivial case is

$$succ(x) < succ(y) = true \vee succ(x) < succ(y) = false$$

Using case rewriting (cf., definition 3.3), we can split this formula into the conjunction of 20 and 21:

$$\neg x < y = true \vee true = true \vee succ(x) < succ(y) = false \quad (20)$$

$$x < y = true \vee succ(x) < succ(y) = true \vee succ(x) < succ(y) = false \quad (21)$$

20 is trivial and 21 is split again in:

$$x < y = true \vee \neg x < y = false \vee false = false \vee succ(x) < succ(y) = true \quad (22)$$

$$\begin{aligned} x < y = true \vee x < y = false \vee \\ succ(x) < succ(y) = false \vee succ(x) < succ(y) = true \end{aligned} \quad (23)$$

22 is trivial again and 23 is a subclause of the initial conjecture: this is the induction step (cf., theorem 4.2). The 16 and 17 are proved exactly in the same way. Consider now 18. To disprove it, we are going to use the convergence property of the initial system. Note that theorem 5.1 applies for proving the ground convergence of the given set of axioms. The instances to be considered are : $0 < succ(0) = false$, $succ(x) < succ(succ(x)) = false$. The first one reduces to $true = false$ whose members are irreducible and different. By theorem 4.3 the conjecture is false. For 19 consider the following instance $0 < succ(x) = false \vee succ(x) < 0 = true$: It can be reduced to $true = false \vee false = true$ and therefore 19 is not valid.

3 Preliminaries

We assume that the reader is familiar with the basic notions of Horn theories and rewrite systems. We introduce the essential terminology below and refer to Huet-Oppen [HO80], and Padawitz [Pad88b] for more detailed presentations. The material in this section is divided into two parts: we first present the basics about conditional equations and then we overview the fundamental rewrite concepts.

3.1 Conditional theories

3.1.1 Terms and substitutions

A signature Σ is a pair (S, F) where S is a set of *sorts* and F is a finite set of function symbols such that F is equipped with a mapping *type*: $F \rightarrow S^* \times S$. For any $g \in F$ the value *type*(g) is the *type* of g , *sort*(Σ) will denote S , *fun*(Σ) will denote F and $g : \alpha \rightarrow s$ will denote a function $g \in F$ whose *type* is α, s . From now on we assume that a given signature Σ is *sensible*, i.e., it admits at least one ground term of each sort.

Every signature Σ defines a set of formulas that can be built from function symbols taken from Σ and free variables taken from a denumerable set X of variables. Let X be a family $\{X_s\}_{s \in S}$ of sets of free variables indexed by S that is disjoint with Σ . For every sort $s \in \Sigma$ let $T(\Sigma, X)_s$ be the set of Σ -terms of sort s constructed using function symbols in F of type α, s as root symbols and variables in X . Σ -terms will be denoted by t, s, l, r, \dots $Var(t)$ stands for the set of all variables appearing in t . If t is a Σ -term and $x \in Var(t)$ then $\#(x, t)$ denotes the number of occurrences of the variable x in t . A term t is *linear* iff $\#(x, t) = 1$ for all variables in $Var(t)$. If $Var(t)$ is the empty set then t is a *ground* term. By $T(\Sigma)_s$ we will denote the set of all ground terms constructed using function symbols in F of type α, s . A sort s is *nullary* if by definition $T(\Sigma)_s$ is a finite set. Note that this property can be easily decided.

Furthermore, let N^* be the set of sequences of positive integers, ϵ the empty sequence in N^* and \cdot the concatenation operation on sequences. We call the elements of N^* *positions* and denote them $u, v, w, p, q \dots$ We now define the *prefix ordering* \leq in N^* by $u \leq v$ iff there exists w such that $v = u.w$; in this case we define $v/u = w$. We write $u < v$ iff $u \leq v$ and $u \neq v$. Positions u and v are said to be *disjoint*, denoted $u|v$, iff neither $u \leq v$ nor $v \leq u$.

For any term t $dom(t) \subseteq N^*$ denotes its set of positions and the expression t/u denotes the *subterm of t at a position u* . Also 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 *variable position* if $t(u) = x \in X$ and $\#(x, t) = 1$, a *non-linear variable position* if $t(u) = x \in X$ and $\#(x, t) > 1$. We use $sdom(t)$ to denote the set of strict positions in t . We write $t[s]_u$ to indicate that s is a subterm of t at position u . We denote $s[t]$ the term s whose subterm is t .

A Σ - *substitution* $\eta = \{x_1 \leftarrow t_1, x_2 \leftarrow t_2, \dots, x_k \leftarrow t_k\}$, where $x_i \neq t_i$, assigns Σ - *terms* of appropriate sorts to variables. It is applied to a Σ - *term* t , by simultaneous replacement of all occurrences of each x_i in t by t_i . The Σ - *terms*

t_1, t_2, \dots, t_k are the *values* of η . Greek letters σ, η, \dots denote substitutions. Composition of substitution σ and η is denoted by $\sigma\eta$. The Σ -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 . Syntactic equality between terms is denoted by \equiv . A term t *unifies* with a term s if there exists a substitution σ such that $t\sigma \equiv s\sigma$.

3.1.2 Conditional equations and clauses

Let $\Sigma = (S, F)$ be a signature. A Σ -equation is a pair $e =_s e'$ where $e, e' \in T(\Sigma, X)_s$ are terms of the same sort s . A *conditional* Σ -equation is either a Σ -equation or an expression of one of the following forms: $e_1 \wedge \dots \wedge e_n \Rightarrow e$, or $e_1 \wedge \dots \wedge e_n \Rightarrow$, or \Rightarrow where e, e_1, \dots, e_n are Σ -equations. Given a conditional Σ -equation, e_1, \dots, e_n are said to be the *conditions* and e is the *conclusion*. A Σ -formula (or a Σ -clause) is built from Σ -equations using the symbols \neg and \vee . In other words, a Σ -formula is an expression of the form $\neg e_1 \vee \neg e_2 \vee \dots \vee \neg e_n \vee e'_1 \vee \dots \vee e'_m$. When Σ is clear from the context we omit the prefix Σ in Σ - $\{expression\}$.

In this paper axiomatizations are built out from conditional equations and goals to be proved are clauses, i.e. disjunction of equational literals, since $=$ is the only predicate. We shall identify a conditional equation and its corresponding representation as a clause.

3.1.3 Inductive theory

Given a set of conditional equations Ax of the signature Σ , we recall that a Herbrand model of Ax is a model of Ax whose domain is the set of ground terms (axioms for equality are implicitly assumed to be valid, too). A formula \mathcal{F} is a *deductive theorem of Ax* if it is valid in any model of Ax . This will be denoted by $Ax \models \mathcal{F}$. Deductive theorems can be proved by refutation, by deriving a contradiction from $\neg \mathcal{F} \wedge Ax$. Usually $\neg \mathcal{F}$ is transformed into a universal sentence \mathcal{U} by introducing *skolem functions*. Hence the signature Σ has to be extended. Most theorem-proving techniques rely on Herbrand's theorem, which implies that $\mathcal{U} \wedge Ax$ is unsatisfiable iff it has no Herbrand model (w.r.t. the extended signature). The notion of inductive theory can be related to special kinds of models: Herbrand models, initial models, or constructor models [Pad88a, Pad88c, Pad88b, Zha88]. We study here the initial and the Herbrand model approaches:

Definition 3.1 *Let Ax be a set of conditional equations of the signature Σ . A clause e is an inductive theorem of Ax iff for any ground substitution σ , $e\sigma$ is valid in Ax . This will be denoted by $Ax \models_{ind} e$.*

For clauses validity in all Herbrand models differs, in general, from validity in the initial model. However, these two notions of validity coincide for unconditional equations:

Proposition 3.1 ([Pad88b]) *A clause e is an inductive theorem iff it is valid in any Herbrand model of Ax . An unconditional equation e is an inductive theorem iff it is valid in the initial model of Ax .*

3.2 Rewrite Relations

Given a binary relation \rightarrow , \rightarrow^* (resp. \rightarrow^{-1}) denotes its reflexive and transitive closure (resp. its inverse). Given two binary relations R, S , let $R \circ S$ denote their composition. The relation $\rightarrow^* \circ (\rightarrow^*)^{-1}$ is written in short \downarrow . A relation \rightarrow is noetherian if there is no infinite sequence $t_1 \rightarrow t_2 \rightarrow \dots$. In the following we suppose given a *reduction ordering* \succ on the set of terms, that is, a transitive and 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 write $s \triangleleft t$ if s is a strict subterm of t and $t \succ s$. If neither $t \triangleleft s$ nor $t \succ s$ nor $t \equiv s$ then we say that s and t are incomparable and we denote this by $t \nprec s$.

A reduction ordering can be extended to literals by comparing the multisets of their members using the multiset extension of \succ (see [DM79]). Formulas are compared using the multiset extension of this last ordering to the multiset of their atomic subformulas. Since there is no ambiguity, all these extensions will be denoted by \succ , too.

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

3.2.1 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 some 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 by a recursive call to the theorem-prover. Termination is ensured by requiring the conditions to be smaller (w.r.t. the reduction ordering \succ) than the conclusion. Various conditional rewrite relations have been studied in the literature ([KR89, DOS88, Vor89]).

Definition 3.2 *Let H be a set of conditional equations and let A be a term. We write:*

$$A[s\sigma] \mapsto_H A[t\sigma]$$

if there exists a substitution σ and a conditional equation $a_1 = b_1 \wedge \dots \wedge a_n = b_n \Rightarrow s = t$ in H such that :

1. $s\sigma \succ t\sigma$;
2. $\forall i \in \{1 \dots n\} \ a_i\sigma \downarrow_H b_i\sigma$;

$$3. \{A[s\sigma]\} \succ \{a_1\sigma, b_1\sigma, \dots, a_n\sigma, b_n\sigma\}$$

A term A is reducible w.r.t. \mapsto_H if there is a term B such that $A \mapsto_H B$. Otherwise we say that A is *R-irreducible* we denote that by $t \not\mapsto_H$.

The system H will be qualified as *convergent* if

$$\forall a, b \in T(F) \quad H \models a = b \quad \Rightarrow \quad a \Downarrow_H b$$

Note that when H is a set of conditional rules then the relation \mapsto_R is similar to the notion of decreasing rewriting of Dershowitz, Okada and Sivakumar [DOS88].

The relation \mapsto_H will be extended to sets of clauses in a natural way: by definition, $S \cup \{c\} \mapsto_H S \cup \{d\}$ whenever $c \mapsto_H d$.

3.2.2 Case rewriting

Case reasoning is a technique, which is the basis of many theorem proving strategies. It is a most important rule in the context of inductive theorem proving where case splitting arises naturally from an induction hypothesis. We propose here a notion of case rewriting, which is well-suited to inductive reasoning.

Definition 3.3 (Case rewriting) Let H be a set of conditional equations and let $c \Rightarrow s = t$ be a conditional equation in H (where c is a positive literal). Let $A[s\sigma]_n$ be a clause (where σ is a substitution) and let S be a set of clauses. The case rewriting rule can be stated as follows

$$S \cup \{A[s\sigma]_n\} \mapsto_H S \cup \{(c\sigma \vee A[s\sigma]_n), (\neg c\sigma \vee A[t\sigma]_n)\}$$

This definition can be generalized to the case where c is a conjunction of positive literals in a straightforward way. Let us denote $\mapsto_H \cup \mapsto_H$ by \hookrightarrow_H . The following proposition is the basis for proving (or disproving) clausal theorems.

Proposition 3.2 The case rewriting rule is sound (the derived set of clauses is logically equivalent to the initial set provided S contains H).

The proof is trivial. Note that the relation \hookrightarrow_H is not noetherian. Therefore some *external* control will be necessary when applying this rewriting technique to simplify goals.

3.2.3 Supported rewriting

We introduce now a new rewrite relation, which will be useful for expressing that an inductive hypothesis can help the proof of the premises of a conditional rule.

Definition 3.4 (Supported rewriting) Let H be a set of conditional equations, W be a set of equations and A be a term. We define the supported rewriting relation $\mapsto_{H[W]}$ by:

$$A[s\sigma] \mapsto_{H[W]} A[t\sigma]$$

if there exists a substitution σ and a conditional equation $a_1 = b_1 \wedge \dots \wedge a_n = b_n \Rightarrow s = t$ in H such that:

1. $s\sigma \succ t\sigma$
2. $\forall i \in \{1 \dots n\} \quad a_i\sigma \Downarrow_{H \cup W} b_i\sigma$
3. $\{A[s\sigma]\} \succ \{a_1\sigma, b_1\sigma, \dots, a_n\sigma, b_n\sigma\}$

If W is empty or H is a set of unconditional equations we write \mapsto_H instead of $\mapsto_{H[W]}$.

3.2.4 Relaxed rewriting

The following rewrite relation will be useful to handle non-orientable equations.

Definition 3.5 (Relaxed rewriting) *Let H be a set of unconditional equations. We define the relaxed rewriting relation by: $g[u\theta] \rightsquigarrow_H g[v\theta]$ if there is $(u = v) \in H$ and there is a substitution θ such that $u\theta \neq v\theta$.*

4 How to prove and disprove inductive theorems

In this section, we propose general methods to prove or disprove automatically that sets of clauses are inductive consequences of theories axiomatized by sets of conditional rules. This technique allows us to replace inductive reasoning by pure algebraic simplification. This mechanization of inductive proofs is based on the notion of *test set*, which, in essence, provides a finite description of the initial model of a given conditional theory. This section is divided into three parts. We first discuss and define the key notion of *test set*. Then we show how to use it for proving theorems. Finally we show how test sets allow to disprove theorems.

4.1 Test sets

Let us first give a motivation for using test-sets rather than standard structural schemes such as *covering sets*. If we consider a theory with the only axiom $s(s(0)) = 0$, then the proposition $s(s(x)) = x$ is valid in the initial model, not in all models, but does not really need induction to be proved since the initial model reduces to two elements: $0, s(0)$. It is sufficient to check the theorem for these two values, which is a trivial task. We are looking for a method that would produce as induction schemes these terms $0, s(0)$ rather than the less informative terms $0, s(x)$. To ensure that these situations can be captured we must exhibit the ground irreducible terms up to some depth; this explains the technical depth conditions that single test sets out cover sets. Furthermore, an important point concerning efficiency is that the cardinality of a test set is smaller in general than the cardinality of a covering set since we require that every element of the test set admits at least one irreducible ground instance. This partly motivates the second requirement in definition 4.1.

A rewrite rule $c \Rightarrow s \rightarrow r$ is *left-linear* if s 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*. A term is *strongly R -irreducible* if none of its non-variable subterms is an instance of a left-hand side of R .

Further, 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 $\text{depth}(t)$. The *strict depth* of t , written as $\text{sdepth}(t)$, is the maximum of the depths of the strict positions in t . The *depth of a rewrite system* R , denoted $\text{depth}(R)$, is defined as the maximum of the depths of the left-hand sides of R . Similarly, the *strict depth* of R , written as $\text{sdepth}(R)$, is the maximum of the depths of the strict positions in the left-hand sides of R .

If R is a rewrite system, then R can be partitioned into left-linear rules R_{ll} and non-left-linear rules R_{nl} , i.e., $R = R_{ll} \cup R_{nl}$. $\text{depth}(R_{ll})$ denotes the maximum of the depths of the left-linear rules R_{ll} of R and $\text{depth}(R_{nl})$ denotes the maximum of the depths of the non-left-linear rules R_{nl} , then the number $D(R)$ is equal to $\text{depth}(R)$ if $\text{depth}(R_{nl}) < \text{depth}(R_{ll})$ and $\text{sdepth}(R) < \text{depth}(R_{ll})$, otherwise $D(R)$ is equal to $\text{depth}(R)+1$. $D(R)$ is said to be a *bound* for R .

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

completeness: *For any R -irreducible ground term s there exists a term t in $S(R)$ and a ground substitution σ such that $t\sigma = s$;*

transnormality: *For any non-ground term t in $S(R)$ and for any position u in t for which t/u is a non-ground term and $|u| = \text{depth}(R)$, there exist infinitely many strongly-irreducible ground instances t_0, t_1, \dots of t such that $t_0/u \neq t_1/u, \dots$;*

coveredness: *Any non-ground term in $S(R)$ has variables only at depth greater or equal than $D(R)$.*

Let us show that this notion of test set is what is really needed for automating induction. First, the *completeness* property allows us to prove theorems by induction on the domain of irreducible terms rather than on the whole set of terms (see theorem 4.1 and 4.2). Second, the *transnormality* and *coveredness* properties are crucial for the refutation of inductive conjectures (see theorem 4.3). In order to derive useful lemmas we must be able to apply the rewrite rules to some instances of the conjecture. For this reason, these instances have to be deep enough: this is also ensured by the *coveredness* property.

For left-linear rules the *transnormality* property may be weakened: it is sufficient to consider non-ground terms that admit at least one strongly irreducible instance. Moreover, $D(R)$ can be taken equal to $\text{depth}(R) - 1$ in left-linear theories defined over a free set of constructors. This is important for efficiency purpose.

It is possible to compute test sets for equational theories in a relatively efficient way (see [Kou90, Hub91]). Unfortunately no algorithm exists for the general case of conditional theories. However, in the last section, we will give a method to compute test sets in conditional theories defined over a free set of constructors.

Example 4.1

a) If $F = \{a, f, g\}$ and $R = \{f(x, x) \rightarrow x, f(x, g(x)) \rightarrow g(x), f(g(x), x) \rightarrow x, g(g(a))$

$\rightarrow a\}$, then the set $\{a, g(a)\}$ is a test set for R .

b) Let us come back to the introductory example. Let H be the set of axioms $1, 2, \dots, 11$. As we pointed out, $S(H) = \{0, \text{succ}(x), \text{true}, \text{false}\}$ may be considered a test set. Note that the set of ground instances of the members of the test set contains all ground irreducible terms w.r.t. the relation \mapsto_H . Moreover, the three properties of the definition are verified.

Only some particular variables in a conjecture need to be instantiated by induction schemes. There are called *induction variables* and we shall prove that they are the only one needed for completeness. Intuitively, these variables occurs at positions where a certain amount of instantiation is needed before applying a function definition (i.e. a rewrite step) above this position.

Definition 4.2 Let R be a set of conditional rules and let C be a clause. A variable $x \in C$ is an induction variable if it occurs in a subterm s of C such that i) s is unifiable with the left-hand side l of a rule $c \Rightarrow l \rightarrow r$ of R , ii) the position of x in s is a strict or a non-linear variable position of l or the position of a variable of l that also occurs in c .

Terms in test sets are used to build *test-substitutions* for induction variables.

Definition 4.3 If $S(R)$ is a test set for a set R of conditional rules and C is a clause, then a test instance of C with respect to $S(R)$ is an instance of C obtained by substituting renamed terms of $S(R)$ for the induction variables of C of the corresponding sort. A test-substitution is a substitution that maps any variable from its domain to a renaming of an element of $S(R)$ of the same sort.

Example 4.2 Let R be $\{x + 0 = x, x + \text{succ}(y) = \text{succ}(x + y)\}$. Consider for C , the associativity property $(x+y)+z = x+(y+z)$. The induction variables are y and z . The test instances of C with respect to $\{0, \text{succ}(x')\}$ (true and false are omitted here since we only consider well-sorted equations) $(x + 0) + 0 = x + (0 + 0)$, $(x + 0) + \text{succ}(z') = x + (0 + \text{succ}(z'))$, $(x + \text{succ}(y')) + 0 = x + (\text{succ}(y') + 0)$, $(x + \text{succ}(y')) + \text{succ}(z') = x + (\text{succ}(y') + \text{succ}(z'))$.

4.2 Inductive proofs by simplification

Our notion of induction uses a noetherian ordering on ground terms that contains the conditional rewriting relation. We can use as an inductive hypothesis any instance of the theorem we want to prove, as soon as this instance is smaller (w.r.t. \succ) than the one that is currently considered. We propose here a rewriting relation which is sound, with respect to the use of induction hypotheses. On one hand, an inductive hypothesis can be used to verify the conditions of a conditional rule. This use of an inductive hypothesis is safe since the conditions of a conditional rule are smaller than the conclusion with respect to the noetherian ordering. On the other hand an inductive hypothesis can be applied after simplification of the goal by an axiom rule. This use of inductive hypotheses is also safe since we can show that in this situation

they are smaller than the currently examined goal. The following theorem formalizes the previous discussion and shows how to prove equations in the class of isomorphic initial models of conditional theories.

Theorem 4.1 *Let R be a set of conditional rules, $S(R)$ be a test set, and $u = v$ be an equation. We define \Leftrightarrow as the reflexive closure of the relation:*

$$(\vdash_{R[u=v]}) \circ (\vdash_{R \cup \{u=v\}})^*$$

If for all test-substitutions ν there is a term w such that $u\nu \Leftrightarrow w$ and $v\nu \Leftrightarrow w$ then $u = v$ is an inductive theorem for R .

proof: Suppose that $u = v$ is not an inductive theorem of R . Let τ be a ground substitution such that $u\tau = v\tau$ is the smallest equation w.r.t. \prec such that

$$R \not\models u\tau = v\tau$$

We can suppose that τ is R -irreducible, otherwise we could exhibit a smaller counter-example. Hence, there is a test substitution ϕ and a ground substitution θ such that $\tau = \phi\theta$. From the hypothesis, there is a term w such that $u\phi \Leftrightarrow w$ and $v\phi \Leftrightarrow w$. By instantiation, we have :

$$u\tau \Leftrightarrow w\theta \text{ and } v\tau \Leftrightarrow w\theta$$

It remains to show that every term replacement occurring in the proof above is valid (in the initial model) either because it is a logical consequence of R or by induction hypothesis. This will give a contradiction with the choice of τ .

1. suppose the instance $u\rho = v\rho$ of $u = v$ is used in some unconditional step. Since it is not the first step, the replacement is applied to a term which is smaller than the initial term $u\phi\theta$ (or $v\phi\theta$) since this last one has been rewritten at least once. We have again $\{u\phi\theta, v\phi\theta\} \succ \{u\rho, v\rho\}$:
2. if $u\rho = v\rho$ is used in some conditional step. Since conditions are evaluated only when they are smaller than the term to rewrite, every instance of an equation which is used in such a proof is smaller than the original goal $\{u\phi\theta, v\phi\theta\}$.

□

It is straightforward to generalize the previous method to prove clauses. In the following, we call a *tautology* a clause that contains either two complementary literals or an instance of $x = x$.

Theorem 4.2 *Let R be a set of conditional rules, $S(R)$ be a test set, and C be a clause. If for all test-substitutions ν we have $\{C\nu\} \hookrightarrow_{R^*} \{p_1, p_2, \dots, p_n\}$, and every clause p_j is either a tautology or is subsumed by an axiom or contains an instance of C that is strictly smaller w.r.t. \succ than $C\nu$, then C is an inductive theorem of R .*

The proof is straightforward.

Example 4.3 *Let us prove the transitivity of $<$ (see axioms of the introductory example).*

$$x < y = \text{false} \vee y < z = \text{false} \vee x < z = \text{true}$$

The only non-trivial test instance among eight of them is C :

$$\text{succ}(x) < \text{succ}(y) = \text{false} \vee \text{succ}(y) < \text{succ}(z) = \text{false} \vee \text{succ}(x) < \text{succ}(z) = \text{true}$$

After three steps of case-rewriting we get only one non-tautological clause, namely:

$$x < y = \text{false} \vee y < z = \text{false} \vee x < z = \text{true} \vee \text{succ}(x) < \text{succ}(y) = \text{false} \\ \vee \text{succ}(y) < \text{succ}(z) = \text{false} \vee \text{succ}(x) < \text{succ}(z) = \text{true}$$

This clause contains a subclause that is a strictly smaller instance of the property C to be proved. Hence by theorem 5.2, the proof of transitivity is achieved.

4.3 Disproving inductive conjectures

The notion of a test set is particularly useful for refuting inductive properties. The next definition provides the criteria to reject conjectures that are not inductive theorems. The idea is that when a test instance of a clause has no subterm that matches a left-hand side of an axiom, then we can derive a ground instance of that clause with the same property. This is ensured by the transnormality and coveredness properties of the test sets. The ground instance provides us with a counterexample to the conjectured theorem.

Definition 4.4 *Suppose that we are given a set of conditional rules R and a test set $S(R)$. Then a clause $\neg e_1 \vee \dots \vee \neg e_m \vee g_1 = d_1 \vee \dots \vee g_n = d_n$ is **quasi-inconsistent** with respect to R if there is a test instance $C\sigma$ such that for all $i \leq m$ $e_i\sigma$ is an inductive theorem and for all $j \leq n$ at least one of the following is satisfied:*

- i) $g_j\sigma \not\equiv d_j\sigma$, and both $g_j\sigma$ and $d_j\sigma$ are strongly irreducible by R .*
- ii) $g_j\sigma \succ d_j\sigma$, and $g_j\sigma$ is strongly irreducible by R .*
- iii) $g_j\sigma \prec d_j\sigma$, and $d_j\sigma$ is strongly irreducible by R .*

The next result shows that, when the set of axioms is convergent, a quasi-inconsistent clause cannot be inductively valid. This is proved by building a well-chosen ground instance of the clause, that is false in some Herbrand model of the axioms. In particular, if the clause is an equation then it is not valid in the initial model.

Theorem 4.3 *Let R be a convergent set of conditional rules and $S(R)$ be a test set for R . If C is quasi-inconsistent w.r.t. R then C is not an inductive theorem of R .*

Example 4.4 Consider the following conditional axioms for integers with $+$, odd , and $even$.

$$x + 0 \rightarrow x \quad (24)$$

$$x + s(y) \rightarrow s(x + y) \quad (25)$$

$$even(0) \rightarrow true \quad (26)$$

$$even(s(0)) \rightarrow false \quad (27)$$

$$even(s(s(x))) \rightarrow even(x) \quad (28)$$

$$even(x) = true \Rightarrow odd(x) \rightarrow false \quad (29)$$

$$even(x) = false \Rightarrow odd(x) \rightarrow true \quad (30)$$

Here the test set is $\{0, s(0), s(s(z)), true, false\}$. Note that the axioms satisfy the convergence property. Consider the conjecture $even(x) = true \vee odd(x) = false$. It is quasi-inconsistent as shown by the following instance $even(s(0)) = true \vee odd(s(0)) = false$.

proof of theorem 4.3:

Let L be the set of left-hand sides of the rules in R . When we say that a term is an instance of L we mean that it is an instance of an element of L . Let C be of the form $\neg e_1 \vee \neg e_2 \vee \dots \vee \neg e_m \vee g_1 = d_1 \vee \dots \vee g_n = d_n$, a clause which is quasi-inconsistent with respect to R . Then there is a test instance $C\sigma$ of C , such that $e_{l''}\sigma$ is an inductive theorem for all $l'' \leq m'$ and for all $l' \leq n$ at least one of the following facts hold:

- i) $g_{l'}\sigma \not\equiv d_{l'}\sigma$ and no subterm of $g_{l'}\sigma$, $d_{l'}\sigma$ is an instance of L .
- ii) $g_{l'}\sigma \succ d_{l'}\sigma$ and no subterm of $g_{l'}\sigma$ is an instance of L .
- iii) $g_{l'}\sigma \prec d_{l'}\sigma$ and no subterm of $d_{l'}\sigma$ is an instance of L .

We shall prove the theorem when iii) holds. The other cases are similar. In order to show that C is not an inductive theorem of R , it is sufficient to show that $(g_1 = d_1 \vee \dots \vee g_n = d_n) \sigma$ is not an inductive theorem of R . Indeed, all ground instances of $(\neg e_1 \vee \neg e_2 \vee \dots \vee \neg e_m) \sigma$ are false in R , by hypothesis. Let us show that in the case where for all $l' \leq n$ $g_{l'}\sigma \not\equiv d_{l'}\sigma$ and no subterm of $g_{l'}\sigma$, $d_{l'}\sigma$ is an instance of L , then C is not an inductive theorem of R .

Let $Var(g_1 = d_1 \vee \dots \vee g_n = d_n) = \{x_1, \dots, x_k\}$ and let $A\sigma$ be an instance of $A = g_1 = d_1 \vee \dots \vee g_n = d_n$ such that $\sigma = \{x_1 \leftarrow t_1, \dots, x_k \leftarrow t_k\}$, where $t_i \in S(H)$ for all $i \leq k$. We shall show that there exists at least a ground instance $A\sigma\eta$ of $A\sigma$ no subterm of which is an instance of L . This will be enough to ensure that $(g_1 = d_1 \vee \dots \vee g_n = d_n) \sigma$ is not an inductive theorem of R . There are two cases depending on the groundness of t_k :

Case 1: Suppose t_i is ground for all $i \leq k$. Then $A\sigma$ is a ground clause. Thus, there is no i such that $g_i\sigma$ and $d_i\sigma$ rewrite to the same term using R . Hence

$(g_1 = d_1 \vee \dots \vee g_n = d_n) \sigma$ is false using the ground convergence of R .

Case 2: Suppose there exists $i \leq k$ such t_i is non-ground. Suppose that the variables x_1, \dots, x_k occur at positions v_1, \dots, v_k , and let m be the maximal number of non-ground subterms of $t_{k'}$, which are rooted at positions of depth equal to $\text{depth}(H)$. Let $\sigma\eta$ be a strongly H-irreducible ground substitution instance of σ such that $|\text{depth}(A\sigma\eta/v_i w_{ij}) - (A\sigma\eta/v_{i'} w_{i'j'})| > \text{depth}(A\sigma)$ for all $l \leq i, i' \leq k$, and all $l \leq j' \neq j \leq m$. Note that such a substitution instance exists by using the transnormality property of the test sets. Let us show that $A\sigma\eta$ contains no subterm which is an instance of L . Were a subterm of $A\sigma\eta$ an instance of L , there would exist a strict position u in $A\sigma\eta$ (because $\sigma\eta$ is strongly R-irreducible) and a term $s \in L$ such that $A\sigma\eta/u$ is an instance of s . There are two cases depending on the linearity of s :

Case 2.1: Suppose s is linear. Let first v be a non-variable position in s . Necessarily uv is a non variable position in $A\sigma$ by using the coveredness property of test sets. Since $A\sigma\eta/u$ is an instance of s , we have $A\sigma(uv) = A\sigma\eta(uv) = s(v)$. Now suppose that $VP(s)$ is the set of variable positions of s , and σ'' the substitution such that for any $s(w) = x \in X$, then $x\sigma'' = A\sigma\eta/uw$. Note that σ'' is well defined, because s is linear, so x cannot have several occurrences. It follows that $A\sigma/u = \sigma''$, thus $A\sigma$ contains a subterm which is an instance of $s \in L$, contradiction.

Case 2.2: Suppose s is non-linear. Let v a non-variable position in s . Necessarily uv is a non variable position in $t\sigma$ by using the coveredness property. Since $A\sigma\eta/u$ is an instance of s , we have $A\sigma(uv) = A\sigma\eta(uv) = s(v)$. Let $(u_1, u_2) \in VP(s)$, such that $s(u_1) = s(u_2) = x \in X$. Then $A\sigma\eta/uu_1 = A\sigma\eta/uu_2$. Since $A\sigma$ contains no subterm which is an instance of L , there must exist such a pair uu_1, uu_2 of position in $A\sigma$ such that $A\sigma/uu_1 \neq A\sigma/uu_2$. The remainder of the proof of this case is by subcases depending on the groundness of subterms $A\sigma/uu_1$ and $A\sigma/uu_2$:

Case 2.2.a: Suppose $A\sigma/uu_1$ and $A\sigma/uu_2$ are ground. Then $A\sigma/uu_1 = A\sigma\eta/uu_1$ and $A\sigma/uu_2 = A\sigma\eta/uu_2$. Since $A\sigma\eta/uu_1 = A\sigma\eta/uu_2$, we have $A\sigma/uu_1 = A\sigma/uu_2$, contradiction.

Case 2.2.b: Suppose $A\sigma/uu_1$ is ground and $A\sigma/uu_2$ is non-ground. Then $A\sigma/uu_1 = A\sigma\eta/uu_1$. On the other hand, since $u \in \text{sdom}(A\sigma)$ and $|u_2| \leq \text{depth}(H)$, there must exist a position $v_i w_{ij}$ in $A\sigma$ such $A\sigma/v_i w_{ij}$ is non-ground, $uu_2 < v_i w_{ij}$, and $|w_{ij}| = \text{depth}(H)$, for some $i \leq k$ and $j \leq m$. Thus, $\text{depth}(A\sigma\eta/uu_2) > \text{depth}(A\sigma\eta/v_i w_{ij})$. Further, $\text{depth}(A\sigma\eta/v_i w_{ij}) > \text{depth}(A\sigma)$ by using the definition of substitution $\sigma\eta$. Now, $\text{depth}(A\sigma) > \text{depth}(A\sigma/uu_1) = \text{depth}(A\sigma\eta/uu_1)$. Thus, $\text{depth}(A\sigma\eta/uu_2) > \text{depth}(A\sigma\eta/uu_1)$, contradiction.

Case 2.2.c: Suppose $A\sigma/uu_1$ and $A\sigma/uu_2$ are non-ground. Since $A\sigma/uu_1 \neq A\sigma/uu_2$, there must exist a position w such that $A\sigma/uu_1 w \neq A\sigma/uu_2 w$. There are again three subcases depending on the groundness of subterms $A\sigma/uu_1 w$ and $A\sigma/uu_2 w$:

- (i) Suppose $A\sigma/uu_1w$ and $A\sigma/uu_2w$ are ground: the proof is identical to case 22a.
- (ii) Suppose $A\sigma/uu_1w$ is ground and $A\sigma/uu_2w$ is non-ground: the proof is identical to case 22b.
- (ii) Suppose $A\sigma/uu_1w$ and $A\sigma/uu_2w$ are non-ground: Since $A\sigma/uu_1w \neq A\sigma/uu_2w$, there must exist $l \leq i$, $i' \leq k$, such that one of the following cases hold:
 - (1) $uu_1w = v_iw_{ij}$ and $uu_2w = v_{i'}w_{i'j'}$,
 - (2) $uu_1w = v_iw_{ij}$ and $uu_2w > v_{i'}w_{i'j'}$,
 - (3) $uu_1w = v_iw_{ij}$ and $uu_2w < v_{i'}w_{i'j'}$,
 - (4) $uu_1w > v_iw_{ij}$ and $uu_2w = v_{i'}w_{i'j'}$,
 - (5) $uu_1w > v_iw_{ij}$ and $uu_2w < v_{i'}w_{i'j'}$,
 - (6) $uu_1w > v_iw_{ij}$ and $uu_2w > v_{i'}w_{i'j'}$,
 - (7) $uu_1w < v_iw_{ij}$ and $uu_2w = v_{i'}w_{i'j'}$,
 - (8) $uu_1w < v_iw_{ij}$ and $uu_2w < v_{i'}w_{i'j'}$,
 - (9) $uu_1w < v_iw_{ij}$ and $uu_2w > v_{i'}w_{i'j'}$,

(1) Suppose $uu_1w = v_iw_{ij}$ and $uu_2w = v_{i'}w_{i'j'}$. Then $\text{depth}(A\sigma\eta/uu_1w) > \text{depth}(A\sigma\eta/v_iw_{ij}) > \text{depth}(A\sigma\eta/v_iW_{i'j'})$ by using definition of substitution $\sigma\eta$. However, $\text{depth}(A\sigma\eta/v_iW_{i'j'}) = \text{depth}(A\sigma\eta/uu_2w)$ by case hypothesis, and thus $\text{depth}(A\sigma\eta/uu_1w) > \text{depth}(A\sigma\eta/uu_2w)$, contradiction.

The proof of subcases (2), (3), (4), (5), (7) and (9) is identical to subcase (1).

(6) Suppose $uu_1w > v_iw_{ij}$ and $uu_2w > v_{i'}w_{i'j'}$. Since $u \in \text{sdm}(t\sigma)$ and $|u_1| \leq \text{depth}(R)$, there must be $uu_1 < v_iw_{ij}$. Let $w = pq$, where p is such that $uu_1p = v_iw_{ij}$. Since $A\sigma/uu_1w \neq A\sigma/uu_2w$, we have $A\sigma/uu_1p \neq A\sigma/uu_2p$, and $uu_1p = v_iw_{ij}$ and either $uu_2p = v_{i'}w_{i'j'}$, or $uu_2p < v_{i'}w_{i'j'}$, or $uu_2p > v_{i'}w_{i'j'}$. The proof of those of subcases is identical to (1), (2), (3).

(8) Now Suppose $uu_1w < v_iw_{ij}$ and $uu_2w < v_{i'}w_{i'j'}$. Assume also that $v_{i'}w_{i'j'}$ is along a maximal path of $A\sigma\eta/uu_2$. Then $\text{depth}(A\sigma\eta/uu_1w) > \text{depth}(A\sigma\eta/v_iw_{ij}) > \text{depth}(A\sigma) + \text{depth}(A\sigma\eta/v_iW_{i'j'})$ by using the definition of substitution η . However, $\text{depth}(A\sigma) + \text{depth}(A\sigma\eta/v_iW_{i'j'}) > \text{depth}(A\sigma\eta/uu_2w)$, contradiction.

Therefore, no subterm of $A\sigma\eta$ is an instance of L and $g_{n'}\sigma\eta \neq d_{n'}\sigma\eta$ for all $n' \leq n$. Therefore $R \not\models g_{n'}\sigma\eta = d_{n'}\sigma\eta$ since R is convergent. Thus, C is not an inductive theorem of R . This complete the proof of the theorem. \square

4.4 A general procedure for proof by induction

In the previous subsection we have given a technique for proving inductive theorems in one step. However, most of the time several rounds are needed before getting a proof. For instance, several successive instantiations by test sets may be necessary.

The easiest way to present this process is to use the formalism of inference rules as in Bachmair [Bac91] and Reddy [Red90]. The main advantage of this approach is that there is no hierarchy between the intermediate lemmas to be proved, and therefore no difficulty for the management of inductive hypotheses: *every intermediate lemma to be proved will be put in the same set as the initial conjectures and no priority will be (a priori) attached to them.* In that sense our procedure can be viewed as being close to inductionless induction.

4.4.1 Inference rules for inductive proofs

Let R be a system of conditional rules. Our proof by induction procedure (which is detailed in [Bou91]) modifies incrementally two sets of equations:

1. E , the set of equations to be proved.
2. H , the set of equations that have been *reduced* to equations of E , and therefore can be used as induction hypotheses.

Let us give now the set of inference rules I :

The rule *Generate* allows us to initialize inductive steps by replacing conjectures by *implicitly smaller conjectures*. The rule *Simplify₁* simplifies a conjecture with conditional axioms from R , using conjectures from $H \cup E$ when checking the convergence of preconditions. The rules *Simplify₂*, *Simplify₃* and *Simplify₄* apply to a conjecture using either induction hypotheses or other conjectures. Note that *Simplify₄* allows one to handle non orientable equations. The rule *Delete* helps getting rid of tautologies. The rule *Fail* applies when *Generate* cannot be applied.

An *I-derivation* is a sequence of states:

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

An *I-derivation* is *failed* if it ends with an application of the *Fail* rule.

Example 4.5 Let $R = \{p(0) = \text{True}, p(x) = \text{True} \Rightarrow p(s(x)) = \text{True}\}$. The initial state is $(\{p(x) = \text{True}\}, \emptyset)$. We have:

$$(\{p(x) = \text{True}\}, \emptyset) \vdash_{\text{Generate}} (\{\text{True} = \text{True}\}, \{p(x) = \text{True}\}) \vdash_{\text{Delete}} (\emptyset, \{p(x) = \text{True}\})$$

The *Generate* step is justified by noting that $p(s(x)) \mapsto_{R[\{p(x) = \text{True}\}]} \text{True}$ since

$$R \cup \{p(x) = \text{True}\} \models (p(x) = \text{True})$$

Definition 4.5 An *I-derivation* $(E_0, \emptyset) \vdash_I (E_1, H_1) \vdash_I \dots$ is *fair* if either it is failed or the set of persisting equations $(\cup_i \cap_{j \geq i} E_j)$ is empty.

Lemma 4.1 Let $(E_0, H_0) \vdash_I (E_1, H_1) \vdash_I \dots$ be a fair *I-derivation* such that $R \not\models_{\text{ind}} E_0$. Then there exists k such that the rule *Fail* is applied to (E_k, H_k) .

Generate: $(E \cup \{e = e'\}, H) \vdash_I (E \cup (\cup_{\sigma} E_{\sigma}), H \cup \{e = e'\})$

For any test-instance $e\sigma = e'\sigma$, there is b such that:

either $e \not\prec e'$ and $(e\sigma \mapsto_{R[H \cup E \cup \{e=e'\}]} b)$ and $E_{\sigma} = \{b = e'\sigma\}$
or $e \star e'$ and $(e'\sigma \mapsto_{R[H \cup E \cup \{e=e'\}]} b)$ and $E_{\sigma} = \{e\sigma = b\}$
or $e\sigma \equiv e'\sigma$ and $E_{\sigma} = \emptyset$.

Simplify₁: $(E \cup \{a = b\}, H) \vdash_I (E \cup \{a' = b\}, H)$

if $a \mapsto_{R[H \cup E \cup \{a=b\}]} a'$

Simplify₂: $(E \cup \{a = b\}, H \cup \{g = h\}) \vdash_I (E \cup \{a' = b\}, H \cup \{g = h\})$

if $a[g\sigma]_u \mapsto_H a' \equiv a[h\sigma]_u$ and $(g\sigma \triangleleft a$ or $g \succ h)$.

Simplify₃: $(E \cup \{c = d\} \cup \{a = b\}, H) \vdash_I (E \cup \{c = d\} \cup \{a' = b\}, H)$

if $a[c\sigma]_u \mapsto_E a' \equiv a[d\sigma]_u$ and $c\sigma \triangleleft a$.

Simplify₄: $(E \cup \{a = b\}, H) \vdash_I (E \cup \{a' = b\}, H)$

if $a \rightsquigarrow_H a'$, $a \star b$ and $a' \preceq b$.

Delete: $(E \cup \{a = a\}, H) \vdash_I (E, H)$

fail: $(E \cup \{e = e'\}, H) \vdash_I \square$

if there is a test-instance $e\sigma = e'\sigma$ such that $e\sigma \neq e'\sigma$ and:

$(e \succ e' \wedge e\sigma \not\downarrow_{R[H \cup E \cup \{e=e'\}]})$ or
 $(e \star e' \wedge e\sigma \not\downarrow_{R[H \cup E \cup \{e=e'\}]} \wedge e'\sigma \not\downarrow_{R[H \cup E \cup \{e=e'\}]})$

Figure 1: Inference System I

This lemma is proved by considering the smallest invalid ground equation that is an instance of an element of $\cup_i E_i$. See appendix.

Theorem 4.4 (Soundness) *Let $(E_0, \emptyset) \vdash_I (E_1, H_1) \vdash_I \dots$ be a fair derivation. If this derivation is non failed then $R \models_{ind} E_0$.*

Note that E_0 is valid even when the derivation is infinite (and *Fail* never applies).

Corollary 4.1 *Given a set of equations E and a set of conditional equations R , if there is an I -derivation from the state (E, \emptyset) to some state (\emptyset, H) , where H is any set of equations, then any element of E is an inductive theorem of R .*

Let us emphasize that we deal here with conditional theories, and that we allow simplification of conjectures by conjectures, unlike Reddy (see Simplify3).

The failure to prove some theorem with the above procedure may be due to the absence of some inference rule for case reasoning. We propose to enhance the procedure by allowing case reasoning through *case rewriting*. More generally, note also that at any step i , one of the current conjectures in E_i can be proved by any other induction proof technique. This is expressed by the generic rule:

$$\text{Other } \frac{(E \cup \{C\}, H)}{(E, H \cup \{C\})} \text{ if } R \models_{\text{ind}} C$$

We can easily prove that the system remains correct when we add this inference rule to the previous ones.

Example 4.6 Consider example 4.4 and let us prove first the commutativity of $+$. We just consider the non-trivial instance by a test-substitution:

$$s(s(x)) + s(s(y)) = s(s(y)) + s(s(x)) \quad (31)$$

After simplification, we have to consider the goal:

$$s(s(s(s(x)) + y)) = s(s(s(s(y)) + x)) \quad (32)$$

The rule *simplify₄* applies to $s(s(s(s(x)) + y))$ and yields $s(s(y + s(s(x))))$ since $s(s(y + s(s(x)))) \prec s(s(s(s(y)) + x))$. Then we get:

$$s(s(y + s(s(x)))) = s(s(s(s(y)) + x)) \quad (33)$$

Applying *Generate* and *simplify₂* using the hypotheses $s(s(y + s(s(x)))) = s(s(s(s(y)) + x))$ finishes the job.

Having proved the commutativity of $+$, we can prove : $\text{even}(x + x) = \text{true}$. To prove it, we proceed as follow : $\text{even}((s(s(x)) + (s(s(x))))$ reduces successively to $\text{even}(s(s(s(s(x + x))))$ and then to $\text{even}(x + x)$. Now $\text{even}(x + x)$ is simplified to true by using the induction hypothesis.

Let us prove now $\text{odd}(x + s(x)) = \text{true}$. The non trivial case is $\text{odd}(s(s(x)) + s(s(s(x))))$ which simplifies to true by supported rewriting.

Therefore, $x + y = y + x$, $\text{even}(x + x) = \text{true}$ and $\text{odd}(x + s(x)) = \text{true}$ are valid in the initial model of 24-30.

4.4.2 Refutation of conjectures

Conditional systems

Let us consider the set of inference rules J obtained by extending I with the following new inference rule:

$$\text{Disproof } \frac{(E \cup \{C\}, H)}{\text{disproof}} \text{ if } C \text{ is quasi - inconsistent.}$$

If the given theory is a (ground) convergent set of conditional rules *Disproof* allows one to detect many false conjectures. This is an easy consequence of the theorem 4.3 and this is formally expressed by the following corollary:

Corollary 4.2 *Let R be a convergent set of conditional equations and let $(E_0, \emptyset) \vdash_J (E_1, H_1) \vdash_J \dots$ be a J -derivation. If there exists k such that *Disproof* is applied to (E_k, H_k) then $R \not\models_{ind} E_k$.*

When discovering an inconsistency at some step (E_k, H_k) we can conclude that the input set E_0 is not valid. This is stated in the next lemma:

Lemma 4.2 *Let $(E_0, \emptyset) \vdash_J (E_1, H_1) \vdash_J \dots$ be a J -derivation where neither *Fail* nor *Disproof* are applied. If for all $j < k$ $R \models_{ind} E_j$ then $R \models_{ind} E_k$.*

proof: If $(E_{k-1}, H_{k-1}) \vdash_J (E_k, H_k)$ by a simplification rule, then the equations which are used for simplification occur in some E_j ($j < k$) and therefore are valid in R by hypothesis. Hence, E_k is valid too in R . If $(E_{k-1}, H_{k-1}) \vdash_J (E_k, H_k)$ by *Generate* on $e = e'$, every auxiliary equation which is used for rewriting an instance of $e = e'$ by a test-substitution is either in R or $E_{j'}$ ($j' < k$) and hence E_k is valid in R . \square

Theorem 4.5 *Assume that R is convergent and that $(E_0, \emptyset) \vdash_J (E_1, H_1) \vdash_J \dots$ is a J -derivation. If there exists k such that *Disproof* is applied to (E_k, H_k) , then $R \not\models_{ind} E_0$.*

1proof: Let $(E_0, \emptyset) \vdash_J (E_1, H_1) \vdash_J \dots$ be a J -derivation. Assume that there exists k such that *Disproof* is applied to (E_k, H_k) . It is clear that *Disproof* has not been applied before step k . By corollary 4.2 $R \not\models_{ind} E_k$ and by lemma 4.2. $R \not\models_{ind} E_0$. \square

Note also that the soundness theorem 4.5 remains valid for the inference system J if we replace, in the statement, “*Fail*” by “*Fail* or *Disproof*”.

A refutationally complete system for equations

Let us consider the particular case of a convergent equational system R . We take the same inference rules as above, except that we replace $R[W]$ by R . We also notice that, in this situation, the rules *Fail* and *Disproof* are identical. Therefore we can prove that our inference system I ($= J$) is refutationally complete. This means that the method allows us to detect any false conjecture by the *Fail* rule.

Theorem 4.6 *Let $(E_0, \emptyset) \vdash_I (E_1, H_1) \vdash_I \dots$ be a fair derivation. Then $R \not\models_{ind} E_0$ if and only if the derivation is failed.*

proof: Let $(E_0, \emptyset) \vdash_I (E_1, H_1) \vdash_I \dots$ be a fair derivation. Assume that $R \not\models_{ind} E_0$. By theorem 4.4. then the derivation is failed. Conversely assume that there exists k such that *Fail* is applied to (E_k, H_k) . By corollary 4.3. $R \not\models_{ind} E_k$ and by lemma 4.2. $R \not\models_{ind} E_0$. \square

5 How to get the convergence property

Convergent systems of equations have the property that two terms are equal if and only if they simplify to identical ones. In this section we present several methods to obtain the convergence property, which is crucial in our framework for disproving conjectures.

5.1 The saturation technique

This technique is detailed in [KR87] and we only give an overview of it here. The Knuth and Bendix procedure [KB70] has been designed to derive convergent systems from equational presentations. The *saturation* technique is a natural extension of Knuth and Bendix algorithm to conditional theories and has first been introduced in [KR87, Rus87]. This technique is based on a set of inference rules which is refutationally complete for conditional equations. The main inference rule is the *superposition* which is a refinement of *paramodulation* (see [RW69, Rus88, HR91]): given an ordering on terms and atoms, the only inferences which are allowed are those obtained by paramodulating maximal members of conclusions into maximal members of conclusions (where *maximal* refers to the given ordering). There are also a certain number of standard rules for simplification, tautology deletion and subsumption. When a fair application of the rules to a set of clauses S generates only clauses that do not persist, i.e. that can be deleted (at some step) then we say that S is saturated. Applying the set of inference rules to conditional equations, the theorem of Knuth and Bendix can be generalized: if an application of the rules allows us to derive from S' a saturated set S then \mapsto_S is convergent.

5.2 Hierarchical axiomatization techniques

Hierarchical axiomatizations are natural tools for building structured specifications. They are obtained by incremental extensions of base theories with new function definitions.

Let $\Sigma B \equiv (S, FB)$ be a subsignature of $\Sigma \equiv (S, F)$. In other words, Σ differs from ΣB only in new function symbols. The elements of ΣB are often called *constructors* and those of $\Sigma - \Sigma B$ are called *defined* function symbols. Terms, substitutions, equations, conditional equations, and formulas over ΣB are called *constructor Σ - terms*, *constructor Σ - substitutions*, *constructor Σ - equations*, *constructor Σ - conditional equations*, and *constructor Σ - formulas*, respectively.

For hierarchical axiomatizations, ground confluence can be obtained by semantic methods. The next theorem is the one underlying D.Plaisted's work [Pla85]:

Theorem 5.1 *Let H' be a set of conditional rules over a signature $\Sigma B \cup (\Sigma - \Sigma B)$ and let $H \subset H'$ be a convergent set of rules over ΣB . Assume that the initial model of H' is a conservative extension of the initial model of H . If for every ground term $f(t_1, \dots, t_n)$, where $f \in (\Sigma - \Sigma B)$, there exists a constructor Σ - term t' such that $f(t_1, \dots, t_n) \mapsto_{H'}^* t'$ then H' is ground convergent.*

proof: Let t be a term in $T(F)$. Suppose that $t \mapsto_{H'}^* t_1$ and $t \mapsto_{H'}^* t_2$. From the hypothesis, there exists two terms t'_1 and t'_2 in $T(\Sigma B)$, such that $t_1 \mapsto_{H'}^* t'_1$ and $t_2 \mapsto_{H'}^* t'_2$. Since H' and H share the same initial model, $t'_1 = t'_2$ is also a theorem in H . By the convergence property of H , we can find a term s such that $t'_1 \mapsto_H^* s$ and $t'_2 \mapsto_H^* s$. This achieves the proof. \square

In the case of a hierarchical conditional theory H' , our procedure for computing test sets (see next section) may be used to demonstrate convergence provided that the initial model of H' is a conservative extension of the initial model of H . For instance, the introductory example is convergent since the axioms introduced to define successively $-$, $<$, gcd do not modify the initial model (Peano arithmetic) and the rewrite rules eliminate these symbols from any ground term. This is the same as asking for the sufficient-completeness property w.r.t. a rewriting relation [GH78].

Further, verification of inductive properties often involves the proof of some lemmas. Adding these lemmas to the initial axiomatization does not destroy the convergence, as stated in the following result:

Theorem 5.2 *If H is ground convergent and C is a conditional equation that is an inductive theorem of H . Then $\mapsto_{H \cup \{C\}}$ is ground convergent.*

proof: Let t, t_1, t_2 be ground terms such that: $t \mapsto_{H \cup \{C\}}^* t_1$ and $t \mapsto_{H \cup \{C\}}^* t_2$. Since C is an inductive theorem, all the ground instances of C that have been used in the proofs above are valid in H . Hence, $t_1 = t_2$ is valid in H , and the convergence property of H allows one to conclude the proof. \square

For instance, in the introductory example 1-11 and 12,13,14,17 are ground convergent (with an appropriately chosen reduction ordering).

6 How to get test sets

As we have already pointed out, the construction of test sets for conditional theories is undecidable. This lies in that such a computation requires some kind of induction. We propose here a method of computing test sets for conditional theories in which the set of function symbols of their signature can be partitionned into a set ΣB of constructors and a set $\Sigma - \Sigma B$ of defined functions. Therefore, we will assume that every left-hand side of a conditional rule has a symbol from $\Sigma - \Sigma B$. This corresponds to the well known requirements for a *principle of definition* to hold in an equational theory (see [HH82]). The key concept of the present method of computing test sets for conditional theories is the notion of *Pattern trees*.

Definition 6.1 *Consider a linear term $r = f(t_1, t_2, \dots, t_k)$ where $f \in \Sigma - \Sigma B$ and $t_i \in T(\Sigma B, X)_{s_i}$ for all $i \leq k$. Given a variable x of sort s in r , the sons of r w.r.t. x are all possible different terms (modulo variable renaming) obtained by replacing x in r by all terms of sort s of the form $g(x_1, \dots, x_n)$, where g is a function symbol of sort s of arity n in ΣB and x_1, \dots, x_n are fresh distinct variables of appropriate sorts.*

Example 6.1 *If $F = \{a, f, g\}$ $FB = \{a, g\}$ where $a : \rightarrow s$, $f : s \times s \rightarrow s$, $g : s \rightarrow s$ and $r = f(g(x), y)$, then the terms $f(g(a), y)$ and $f(g(g(x_1)), y)$ are the sons of r : they are obtained by replacing the variable x in r by the terms a , $g(x_1)$.*

Definition 6.1 is the basis for the concept of pattern trees:

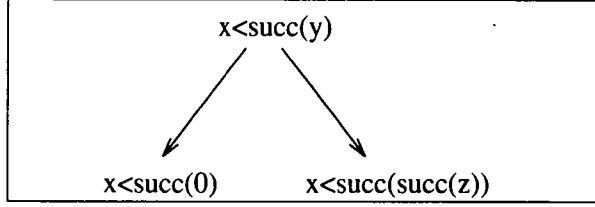


Figure 2: Pattern tree of $x < \text{succ}(y)$

Definition 6.2 A pattern tree \mathbf{T} of a linear term t is a tree, the root of which is labeled with t and the outgoing branches from each non-leaf node are labeled with the sons of the label at that node, w.r.t. some variable.

Let us illustrate definition 6.2 on a simple example:

Example 6.2 If $FB = \{0, \text{succ}, \text{true}, \text{false}\}$, $F = FB \cup \{<\}$ where $0 : \rightarrow \text{int}$, $< : \text{int} \times \text{int} \rightarrow \text{bool}$, $\text{true} : \rightarrow \text{bool}$, $\text{false} : \rightarrow \text{bool}$, and $\text{succ} : \text{int} \rightarrow \text{int}$, then the tree given in figure 2 is a pattern tree of the term $x < \text{succ}(y)$.

Pattern trees enjoy some fundamental properties that are described in the next proposition:

Proposition 6.1 Given a finite pattern tree of t , i) the set of ground instances of t is equal to the set of ground instances of all leaf labels, ii) the sets of ground instances of two different leaf labels are disjoint, iii) for any ground substitution η there exist a unique leaf r and a unique ground substitution σ such that $t\eta = r\sigma$.

The construction of a test set for a given conditional theory H consists in computing suitable pattern trees of the terms $f(x_1, x_2, \dots, x_k)$ where $f \in \Sigma - \Sigma B$ and $x_i \in X_s$, for all $i \leq k$. In general, when we want to construct such a tree by expansion from its root, there are several questions that come naturally to mind: given a pattern tree \mathbf{T} , then

- a) What nodes have to be expanded?
- b) What variables in them have to be replaced?
- c) When the construction of \mathbf{T} has to halt?

To answer these questions, we will define special kinds of terms that we want to be the leaf labels of the pattern trees we are interested in. These terms possess a well-defined structure that, to a certain degree, mirrors the structure of the left-hand sides of the rules of the conditional theory under consideration.

Definition 6.3 A term t is said to be H -extensible at position u if

- a) u is a variable position in t of sort s , and

b) either s is nullary or u is a strict or a non-linear position of a left-hand side of a rule in H .

A term t is said to be H -extensible if it is H -extensible at some position u . Otherwise, t is said to be H -covering.

Definition 6.4 Given a set H of conditional rules, a term t is said to be pseudo-reducible by H if there exists a non empty sequence of conditional rules $C_1 \Rightarrow t_1 \rightarrow r_1$, $C_2 \Rightarrow t_2 \rightarrow r_2$, ..., $C_n \Rightarrow t_n \rightarrow r_n$ in H and a sequence of positions u_1, u_2, \dots, u_n in t such that $t/u_1 = t_1\sigma_1$, $t/u_2 = t_2\sigma_2$, ..., $t/u_n = t_n\sigma_n$ and $C_1\sigma_1 \vee C_2\sigma_2 \vee \dots \vee C_n\sigma_n$ is an inductive theorem of H . Otherwise, t is said to be pseudo-irreducible by H .

Thus, if a term t is pseudo-reducible by H , then all its ground instances are reducible by H . But proving that a node label is pseudo-reducible by a given conditional theory H amounts to proving some inductive theorems. To avoid any vicious circle, either we can use a different method to prove these particular properties or we can use our method itself with a weaker notion of test set than the one we are currently computing. For instance, the set of all terms that are of depth not greater than $D(H)$ and with variables only at depth $D(H)$ suffices for these weaker test sets.

All of the necessary machinery is now at hand to resolve the questions a, b and c stated above. Let us introduce the different types of node labels we are dealing with by using the class of covering terms.

Definition 6.5 Given a conditional theory H , let \mathbf{T} be a pattern tree of $f(x_1, \dots, x_n)$. A node label t in \mathbf{T} is said to be of

type 1 if t is pseudo-reducible by H ,

type 2 if t is pseudo-irreducible by H and is H -covering,

type 3 if t is an H -irreducible ground term.

A node label t in \mathbf{T} is said to be type- x -free if t is neither of type 1, nor of type 2, nor of type 3. A pattern tree \mathbf{T} of $f(x_1, \dots, x_n)$ is said to be complete if each node label t in \mathbf{T} of type 1, 2, 3 is a leaf label.

In the following discussion, we denote node labels of type 1 using boldface letters.

Example 6.3 If $FB = \{0, \text{succ}, \text{true}, \text{false}\}$, $F = FB \cup \{<\}$ where $0 : \rightarrow \text{int}$, $< : \text{int} \times \text{int} \rightarrow \text{bool}$, $\text{true} : \rightarrow \text{bool}$, $\text{false} : \rightarrow \text{bool}$, and $\text{succ} : \text{int} \rightarrow \text{int}$, and H is the following conditional theory.

$$0 < \text{succ}(x) \rightarrow \text{true} \quad (34)$$

$$x < 0 \rightarrow \text{false} \quad (35)$$

$$x < y = \text{true} \Rightarrow \text{succ}(x) < \text{succ}(y) \rightarrow \text{true} \quad (36)$$

$$x < y = \text{false} \Rightarrow \text{succ}(x) < \text{succ}(y) \rightarrow \text{false} \quad (37)$$

Note that $0 < \text{succ}(y_1)$ and $x < 0$ are reducible and that $\text{succ}(x_1) < \text{succ}(y_1)$ is pseudo-reducible since $x_1 < y_1 = \text{true} \vee x_1 < y_1 = \text{false}$ is an inductive theorem. Therefore, the pattern tree of $x < y$ is complete (see figure 3).

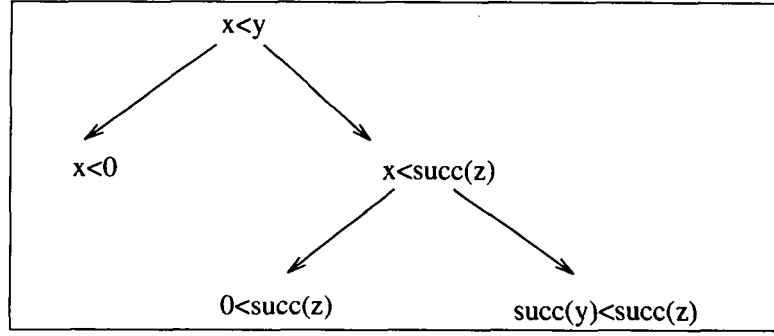


Figure 3: Pattern tree of $x < y$

In general, to compute a complete pattern tree T of the term $f(x_1, \dots, x_n)$, $f \in F$, the following procedure may be used.

A procedure for computing complete pattern trees:

The procedure takes as input a conditional theory H over a signature $\Sigma = (S, F)$. Initially, T is reduced to a node labeled with the term $f(x_1, \dots, x_n)$.

REPEAT AS LONG AS TYPE-X-FREE LEAF LABELS ARE IN T . IF NONE REMAINS TERMINATE SUCCESSFULLY: THE PATTERN TREE OF $f(x_1, \dots, x_n)$ IS COMPLETE.

- (1) SELECT A TYPE-X-FREE LEAF LABEL R IN T .
- (2) SELECT A VARIABLE x IN R AT A POSITION u WHERE R IS H -EXTENSIBLE.
- (3) EXPAND R AT x TO COVER ALL POSSIBILITIES FOR x (I.E., COMPUTE THE SONS OF R AT x).

When the procedure terminates for all $f \in \Sigma - \Sigma B$ with all leaf labels of type 1, then a test-set for H can be easily computed by examining the obtained pattern trees:

Theorem 6.1 *We assume given a set H of conditional rules. If for all $f \in \Sigma - \Sigma B$ there exists a complete pattern tree of $f(x_1, \dots, x_n)$ all leaf labels of which are of type 1, then a test set $S(H)$ can be computed. This test set contains:*

1. *all arguments of leaf labels (of all trees) that are minimal with respect to the subsumption ordering;*
2. *all constructor terms with variables only at depth $D(H)$ no instance of which is an argument of a leaf label.*

proof: Clearly, $S(H)$ is finite. Let us show that $S(H)$ satisfies the properties of Definition 4.1:

- Since all leaf labels are of type 1, every ground term of the form $f(t_1, \dots, t_n)$, where $f \in \Sigma - \Sigma B$ and $t_i \in T(\Sigma B)_s$ for all i , is H -reducible.

Therefore every ground irreducible term is a constructor Σ -term. By the properties of pattern trees, the completeness follows.

- Coveredness is obtained by construction.
- By construction of test sets the only terms in $S(H)$ are constructor terms. By hypothesis any left-hand sides in H contains a non constructor symbol. This means that all ground instances of terms in $S(H)$ are strongly irreducible. By coveredness each term in $S(H)$ has variables occurring only at depth $\text{Depth}(H)$. Therefore all variables may be substituted by infinitely many different constructor terms (we assume that there exists at least one constructor symbol which is not a constant). Hence $S(H)$ has the transnormality property. \square

For example, the leaf labels of the previous example are $x < 0$, $0 < \text{succ}(z_1)$, and $\text{succ}(x_1) < \text{succ}(y_1)$. The arguments of these leaf labels are: x , 0 , 0 , $\text{succ}(z_1)$, $\text{succ}(x_1)$, and $\text{succ}(y_1)$. Thus a test set for H is $\{0, \text{succ}(x_1), \text{true}, \text{false}\}$. Note that true and false are constructor terms and do not occur as argument of leaf labels.

The method also gives a way to check that any ground term of a hierarchical axiomatization can reduce to a constructor term and, as a consequence, by theorem 5.1, to prove the convergence of the system.

Theorem 6.2 *Let H be a set of conditional rules over $\Sigma = \Sigma B \cup (\Sigma - \Sigma B)$ such that each left-hand side contains a symbol from $\Sigma - \Sigma B$. Assume that the initial model of H is a conservative extension of $T(\Sigma B)$. If every leaf of T , a complete pattern tree w.r.t. H , is of type 1 then every term in $T(\Sigma)$ is reducible to a term in $T(\Sigma B)$.*

proof: Let $s = f(t_1, \dots, t_n)$ be a ground term such that $f \in \Sigma - \Sigma B$ and for all i , $t_i \in T(\Sigma B)$. Necessarily, there exists a leaf t of T and a ground substitution σ such that $t\sigma = s$. Since t is of type 1, there exists a rule $\wedge(a_i = b_i) \Rightarrow l \rightarrow r$ such that s contains a ground instance $l\mu$ of l and for all i , $H \models (a_i = b_i)\mu$. By induction hypothesis we may assume that there are terms $A_i, B_i \in T(\Sigma B)$ such that $a_i\mu \mapsto_H A_i$ and $b_i\mu \mapsto_H B_i$. Since the theory is a conservative extension of the initial model $T(\Sigma B)$, no relation can be derived between different constructor terms. Therefore $A_i \equiv B_i$, for all i . Hence for all i there exists c_i such that $a_i\mu \mapsto_H c_i$ and $b_i\mu \mapsto_H c_i$. This shows that s is reducible by H to some smaller term to which we can apply the induction hypothesis. As a consequence, the normal form of s belongs to $T(\Sigma B)$. \square

Example 6.4 *Consider the following theory H :*

$$\text{dif}(x, x) \rightarrow ff \quad (38)$$

$$\text{dif}(0, s(x)) \rightarrow tt \quad (39)$$

$$\text{dif}(s(x), 0) \rightarrow tt \quad (40)$$

$$\text{dif}(s(x), s(y)) \rightarrow \text{dif}(x, y) \quad (41)$$

$$\text{remove}(x, \text{nil}) \rightarrow \text{nil} \quad (42)$$

$$\text{dif}(x, y) = tt \Rightarrow \text{remove}(x, \text{cons}(y, l)) \rightarrow \text{cons}(x, \text{remove}(y, l)) \quad (43)$$

$$\text{dif}(x, y) = ff \Rightarrow \text{remove}(x, \text{cons}(y, l)) \rightarrow \text{remove}(y, l) \quad (44)$$

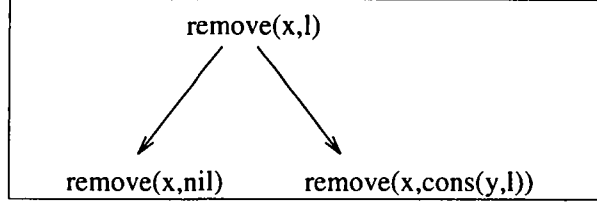


Figure 4: Pattern tree of $remove(x, y)$

then the pattern tree of $remove(x, y)$ is complete and each leaf label is of type 1 (see figure 4). since $dif(x, y) = tt \vee dif(x, y) = ff$ is an inductive theorem. Therefore

- a. the test set of H is $S(H) = \{tt, ff, 0, s(x), cons(x, l), nil\}$.
- b. the system is ground convergent by theorem 5.1.

Note that the previous construction may be extended to the non-free constructors case provided that they are specified by a set of unconditional equations. However, definition 6.1, 6.2 and 6.3 must be changed accordingly. In the following example the constructors s and p verify some relations.

$$0 < 0 \rightarrow ff \quad (45)$$

$$0 < s(0) \rightarrow tt \quad (46)$$

$$s(x) < y \rightarrow x < p(y) \quad (47)$$

$$p(x) < y \rightarrow x < s(y) \quad (48)$$

$$s(p(x)) \rightarrow x \quad (49)$$

$$p(s(x)) \rightarrow x \quad (50)$$

$$x < y = tt \Rightarrow x < s(y) \rightarrow tt \quad (51)$$

$$y < x = ff \Rightarrow y < p(x) \rightarrow ff \quad (52)$$

The test set here is $\{0, p(0), p(p(x)), s(0), s(s(x)), tt, ff\}$. To see that it is sufficient to verify that all the following terms are pseudo-reducible:

$0 < 0, 0 < p(0), 0 < p(p(x)), 0 < s(0), 0 < s(s(x)), p(0) < 0, p(0) < p(0), p(0) < p(p(x)), p(0) < s(0), p(0) < s(s(x)), p(p(x)) < 0, p(p(x)) < p(0), p(p(x)) < p(p(y)), p(p(x)) < s(0), p(p(x)) < s(s(y)), s(0) < 0, s(0) < p(0), s(0) < p(p(x)), s(0) < s(0), s(0) < s(s(x)), s(s(x)) < 0, s(s(x)) < p(0), s(s(x)) < p(p(y)), s(s(x)) < s(0), vs(s(x)) < s(s(y))$.

7 A detailed proof session

Our implementation of test set induction is based on four main data structures: R , a rewrite system for a conditional theory, built with the constructor discipline. Note that this restriction is not required for purely equational theories. C , a set of conjectures to be proved. H , a set of inductive hypotheses. S , a test set of R . We can

describe the procedure *Prove_by_induction* in this way:

Prove_by_induction (C, H, R)

1. Compute a test set S of R .
2. – Simplify each conjecture in C :
 - ★ by the axioms of R .
 - ★ by the other conjectures.
 - ★ by the inductive hypotheses H
 – Eliminate trivial identities.
3. if $C = \emptyset$
 - then all the initial conjectures are inductive consequences of R .
 - else
 - a. Select a conjecture c in C .
 - b. Apply Generate or Case rewriting to derive new inductive hypotheses by instantiation of c by test-substitutions at inductive positions.
 - c. if step b. did not succeed
 - then
 - if R is convergent and c is a quasi-inconsistent equation.
 - then C is not an inductive consequence of R .
 - else failure.
 - else go to 2.

Instead of being hierarchical, our induction proof handles all the induction hypotheses at the same level. This provides us with a fully automated procedure, where much more simplifications are permitted than with explicit hierarchical induction. Moreover, it permits to prove several properties in the same round, each of them being helpful to prove the others.

Below, we show partial transcripts of sessions with SPIKE on the example 4.6, to give more intuition about the abilities of the system. A more detailed account is given in [Bou91].

The procedure is initialized with R , the axioms, and C_0 , the initial conjectures. There are no inductive hypothesis in H_0 .

```

R = { x1+0 = x1 ;
      x1+S(x2) = S(x1+x2)
      even(0) = True ;
      even(S(0)) = False ;
      even(S(S(x1))) = even(x1) ;
      even(x1)=True => odd(x1) = False ;
      even(x1)=False => odd(x1) = True }

C0 = {even(S(x1+x1)) = False ,
      even(x1+x1) = True ,
      odd(x1+S(x1)) = True ,
      odd(x1+x1) = False ,
      x1+x2 = x2+x1 ,

```

```

even(x1) = True v even(x1) = False ,
odd(x1) = True v odd(x1) = False}

H0 = {}

Simplification of odd(x1+S(x1)) = True by
R[H0 U C0]:
True = True

Simplification of odd(x1+x1) = False by
R[H0 U C0]:
False = False

Delete True = True

Delete False = False

...

```

We simplify $odd(x1 + S(x1)) = True$ to the trivial $True = True$. This operation is justified since the premise of the rule $even(x1) = False \Rightarrow odd(x1) = True$ is verified when we assume the other conjecture $even(S(x1 + x1)) = False$. We do the same thing for $odd(x1 + x1) = False$. So we obtain two trivial identities. The fragment of this example illustrates the efficiency of our system thanks to the mutual simplification of the conjectures.

```

Application of generate on x1+x2 = x2+x1
=> The induction variables: x2

C10 = {x1 = 0+x1 ,
       S(x1) = S(0)+x1 ,
       S(S(x1+x2)) = S(S(x2))+x1 ,
       ...}

H10 = {even(S(x1+x1)) = False ,
       even(x1+x1) = True ,
       x1+x2 = x2+x1}

Simplification of S(S(x1+x2)) = S(S(x2))+x1 by H10:
S(S(x2+x1)) = S(S(x2))+x1

C11 = {S(S(x2+x1)) = S(S(x2))+x1 ,
       odd(x1) = True or odd(x1) = False ,
       even(x1) = True or even(x1) = False}

H11 = {even(S(x1+x1)) = False ,
       even(x1+x1) = True ,
       x1+x2 = x2+x1}

...

```

Here, we simplify $S(S(x1 + x2)) = S(S(x2)) + x1$ into $S(S(x2 + x1)) = S(S(x2)) + x1$ using the induction hypothesis $x1 + x2 = x2 + x1$ which is not orientable. This confirms our claim that we can handle non-orientable equations.

```

Application of case rewriting on:
odd(x1) = True or odd(x1) = False:

1) odd(x1) = False v True = True v ~ (even(x1) = False).
2) odd(x1) = True v odd(x1) = False v even(x1) = False.

C12 = {even(x1) = True or even(x1) = False ,

```

```

odd(x1) = False v True = True v ~ (even(x1) = False) ,
odd(x1) = True v odd(x1) = False v even(x1) = False}

H12 = {even(S(x1+x1)) = False ,
       even(x1+x1) = True ,
       x1+x2 = x2+x1 ,
       x1 = 0+x1 ,
       S(x1) = S(0)+x1 ,
       S(S(x1+x2)) = S(S(x1))+x2}

Delete odd(x1) = False v True = True v ~ (even(x1) = False)
...

```

The procedure has just performed a case analysis through case rewriting using the axiom $even(x1) = False \Rightarrow odd(x1) = True$.

Application of generate on:

```

       even(x1) = True or even(x1) = False

C14 = {True = True v True = False ,
       False = True v False = False ,
       even(S(S(x1))) = True v even(S(S(x1))) = False ,
       ...}

H14 = {even(S(x1+x1)) = False ,
       even(x1+x1) = True ,
       x1+x2 = x2+x1 ,
       x1 = 0+x1 ,
       S(x1) = S(0)+x1 ,
       S(S(x1+x2)) = S(S(x1))+x2 ,
       even(x1) = True v even(x1) = False}

```

Delete True = True v True = False

Delete False = True v False = False

Simplification of $even(S(S(x1))) = True \vee even(S(S(x1))) = False$ by R:
 $even(x1) = True \vee even(x1) = False$

Delete $even(x1) = True \vee even(x1) = False$.
 which contains an instance of:
 $even(x1) = True \vee even(x1) = False$.
 smaller than:
 $even(S(S(x1))) = True \vee even(S(S(x1))) = False$.

The generate rule has supplied H with a new induction hypothesis, and C with new conjectured lemmas. Most of them are trivial. A real induction step allows one to eliminate the conjecture: $even(S(S(x1))) = True \vee even(S(S(x1))) = False$.

C15 = {}

```

H15 = {even(S(x1+x1)) = False ,
       even(x1+x1) = True ,
       x1+x2 = x2+x1 ,
       x1 = 0+x1 ,
       S(x1) = S(0)+x1 ,
       S(S(x1+x2)) = S(S(x1))+x2 ,
       even(x1) = True v even(x1) = False}

```

All the initial conjectures are inductive consequences of R.

Have a nice day !

() : unit

/tmp_mnt/users/eureca/bouhoula/SPIKE/example.ml loaded

() : unit

Since the set of conjectures is now empty the proof is achieved. The lemmas $\{x1 = 0 + x1, S(x1) = S(0) + x1, S(S(x1 + x2)) = S(S(x1)) + x2\}$ were generated automatically during the session.

8 Conclusion

We have presented new methods for inductive reasoning. These methods try to exploit as much as possible the power of rewriting. Rewriting systems are a natural framework for inductive reasoning, since they provide well-suited noetherian relations. Proofs in the initial model usually require checking infinite sets of ground equations. The concept of test set allows one to reduce this set to a finite one. Moreover, when the axioms are convergent, test sets give a complete strategy to disprove theorems by producing counterexamples. Moreover, this method generalizes to the case where there are axioms in the theory with negation of equations in the conditions. The theorem-prover SPIKE based on our technique has solved a number of interesting problems. It is being currently extended to incorporate generalization mechanisms and tactics (see [Bun83, BvHSI89, Hut89]), which are necessary to solve many usual problems. Moreover, we are working on better algorithms for the computation of test sets.

Acknowledgements: We sincerely thank Sergey Vorobyov and Konstantin Vershinin for their careful reading of a first draft of this work.

References

- [Aub79] R. Aubin. Mechanizing structural induction. In *Theoretical Computer Science*, volume 9, pages 329–362, 1979.
- [Bac88] L. Bachmair. Proof by consistency in equational theories. In *Proceedings 3rd IEEE Symposium on Logic in Computer Science, Edinburgh (UK)*, pages 228–233, 1988.
- [Bac91] L. Bachmair. *Canonical equational proofs*. Computer Science Logic, Progress in Theoretical Computer Science. Birkhäuser Verlag AG, 1991.
- [BHHW86] S. Biundo, B. Hummel, D. Hutter, and C. Walther. The Karlsruhe induction theorem proving system. In J. Siekmann, editor, *Proceedings 8th International Conference on Automated Deduction, Oxford (UK)*, volume 230 of *Lecture Notes in Computer Science*, pages 672–674. Springer-Verlag, 1986.
- [BK89] R. Bündgen and W. Küchlin. Computing ground reducibility and inductively complete positions. In N. Dershowitz, editor, *Proceedings 3rd*

Conference on Rewriting Techniques and Applications, Chapel Hill (North Carolina, USA), volume 355 of *Lecture Notes in Computer Science*, pages 59–75. Springer-Verlag, April 1989.

- [BM79] R.S. Boyer and J.S. Moore. *A Computational Logic*. Academic Press, New York, 1979.
- [Bou91] A. Bouhoula. Preuve automatique par paramodulation, réécriture et induction. Rapport CRIN 91-R-204, September 1991.
- [Bun83] A. Bundy. *The Computer Modelling of Mathematical Reasoning*. Academic Press, New York, 1983.
- [Bur69] R.M. Burstall. Proving properties of programs by structural induction. *Computer Journal*, 12:41–48, 1969.
- [BvHSI89] 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.
- [Der87] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1 & 2):69–116, 1987.
- [DM79] N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Communications of the Association for Computing Machinery*, 22(8):465–476, 1979.
- [DOS88] N. Dershowitz, M. Okada, and G. Sivakumar. Canonical conditional rewrite systems. In *Proceedings 9th International Conference on Automated Deduction, Argonne (Illinois, USA)*, volume 310 of *Lecture Notes in Computer Science*. Springer-Verlag, May 1988.
- [Fri86] 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.
- [GG89] S.J. Garland and John V. Guttag. An overview of LP, the Larch Prover. In N. Dershowitz, editor, *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (North Carolina, USA)*, volume 355 of *Lecture Notes in Computer Science*, pages 137–151. Springer-Verlag, April 1989.
- [GH78] John V. Guttag and James J. Horning. The algebraic specification of abstract data types. *Acta Informatica*, 10:27–52, 1978.

- [Gra89] 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.
- [HH82] 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.
- [HK88] D. Hofbauer and R.D. Kutsche. Proving inductive theorems based on term rewriting systems. In J. Grabowski, P. Lescanne, and W. Wechler, editors, *Proceedings 1st International Workshop on Algebraic and Logic Programming*, pages 180–190. Akademie Verlag, 1988.
- [HO80] G. Huet and D. Oppen. Equations and rewrite rules: A survey. In R.V. Book, editor, *Formal Language Theory: Perspectives and Open Problems*, pages 349–405. Academic Press, New York, 1980.
- [HR91] J. Hsiang and M. Rusinowitch. Proving refutational completeness of theorem proving strategies: The transfinite semantic tree method. *Journal of the Association for Computing Machinery*, 38(3):559–587, July 1991.
- [Hub91] M. Huber. Test-set approaches for ground reducibility in term rewriting systems, characterizations and new applications. Master’s thesis, Technische Universität Berlin, 1991.
- [Hut89] D. Hutter. Guiding inductive proofs. In M.E. Stickel, editor, *10th International Conference on Automated Deduction*, volume 449 of *Lecture Notes in Artificial Intelligence*, pages 147–161. Springer-Verlag, July 1989.
- [JK86] 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 (Massachusetts, USA)*, pages 358–366, 1986.
- [KB70] Donald E. Knuth and P.B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, Oxford, 1970.
- [KM87] D. Kapur and D.R. Musser. Proof by consistency. *Artificial Intelligence*, 31(2):125–157, February 1987.
- [KNZ86] 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.

- [Kou90] 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.
- [KR87] E. Kounalis and M. Rusinowitch. On word problem in Horn logic. In J.-P. Jouannaud and S. Kaplan, editors, *Proceedings 1st International Workshop on Conditional Term Rewriting Systems, Orsay (France)*, volume 308 of *Lecture Notes in Computer Science*, pages 144–160. Springer-Verlag, July 1987. See also the extended version published in *Journal of Symbolic Computation*, 11(1 & 2), 1991.
- [KR89] S. Kaplan and J.-L. Rémy. Completion algorithms for conditional rewriting systems. In H. Aït-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures, Volume 2: Rewriting Techniques*, pages 141–170. Academic Press, 1989.
- [KR90a] E. Kounalis and M. Rusinowitch. A mechanization of conditional reasoning. In *First International Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale, Florida*, January 1990.
- [KR90b] E. Kounalis and M. Rusinowitch. Mechanizing inductive reasoning. *Bulletin of European Association for Theoretical Computer Science*, 41:216–226, June 1990.
- [KR90c] 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.
- [Küc89] W. Küchlin. Inductive completion by ground proof transformation. In H. Aït-Kaci and M. Nivat, editors, *Colloquium on the Resolution of Equations in Algebraic Structures, Volume 2: Rewriting Techniques*, pages 211–244. Academic Press, 1989.
- [Mus80] D.R. Musser. On proving inductive properties of abstract data types. In *Proceedings 7th ACM Symp. on Principles of Programming Languages*, pages 154–162. Association for Computing Machinery, 1980.
- [Pad88a] P. Padawitz. Can inductive proofs be automated. *Bulletin of European Association for Theoretical Computer Science*, 35:163–170, 1988.
- [Pad88b] P. Padawitz. *Computing in Horn Clause Theories*. Springer-Verlag, 1988.
- [Pad88c] P. Padawitz. Inductive proofs of constructor horn clauses. Technical Report MIP-8810, Universität Passau (Germany), 1988.
- [Pla85] D. Plaisted. Semantic confluence tests and completion methods. *Information and Control*, 65:182–215, 1985.

- [Pro92] M. Protzen. Disproving Conjectures. *11th Conference on Automated Deduction*, LNCS 607, pages 340–354, Saratoga Springs, NY, USA, June 1992.
- [Red90] 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.
- [Rus87] M. Rusinowitch. *Démonstration automatique par des techniques de réécriture*. Thèse de Doctorat d’Etat, Université de Nancy I, 1987. Also published by InterEditions, Collection Science Informatique, directed by G. Huet, 1989.
- [Rus88] M. Rusinowitch. Theorem-proving with resolution and superposition: an extension of Knuth and Bendix procedure to a complete set of inference rules. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, 1988. See also the extended version published in *Journal of Symbolic Computation*, number 1&2, 1991.
- [RW69] G.A. Robinson and L.T. Wos. Paramodulation and first-order theorem proving. In B. Meltzer and D. Mitchie, editors, *Machine Intelligence 4*, pages 135–150. Edinburgh University Press, 1969.
- [Vor89] S.G. Vorobyov. Conditional rewrite rule systems with built-in arithmetic and induction. In N. Dershowitz, editor, *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (North Carolina, USA)*, volume 355 of *Lecture Notes in Computer Science*, pages 492–512. Springer-Verlag, April 1989.
- [Zha88] H. Zhang. *Reduction, Superposition and Induction: Automated Reasoning in an Equational Logic*. PhD thesis, Rensselaer Polytechnic Institute, Department of Computer Science, Troy, NY, 1988.
- [ZKK88] 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 (Illinois, USA)*, volume 310 of *Lecture Notes in Computer Science*, pages 162–181. Springer-Verlag, 1988.

Appendix

proof of lemma 4.1

To prove this lemma, we introduce a notion of complexity of a proof step:

Definition 8.1 Let $P \equiv s\sigma \leftrightarrow_{s=t} t\sigma$ be an equational proof step. We define the complexity of P by:

$$C(P) = \begin{cases} (\{s\sigma\}, \{t\sigma\}) & \text{if } s \succ t \\ (\{t\sigma\}, \{s\sigma\}) & \text{if } s \prec t \\ (\{s\sigma, t\sigma\}, -) & \text{otherwise} \end{cases}$$

Let $P \equiv s\sigma \leftrightarrow_{s=t} t\sigma$ be an equational proof step, we say that $s = t$ justifies P . We say that P is valid in R if $R \models s\sigma = t\sigma$. We consider the set K of proof-steps which are justified by equations in $(\cup_i E_i)$ and we show that the rule *Fail* applies to an equation $s = t$ of $(\cup_i E_i)$ which justifies a minimal proof-step of K which is not valid in R . It is sufficient to prove that $s = t$ cannot be simplified, and that *generate* cannot apply to $s = t$. Therefore *Fail* is applied since the equation cannot persist in the derivation due to the fairness hypotheses.

Let $P \equiv s\sigma \leftrightarrow_{s=t} t\sigma$ be this proof-step and assume that $s \not\prec t$. We can assume that σ is irreducible by R . By hypothesis we have $R \not\models s\sigma = t\sigma$. Assume that $s = t \in E_j$ and $(E_j, H_j) \vdash_I (E_{j+1}, H_{j+1})$ by application of *simplify* or *generate* on $s = t$. We discuss now the situation according to the rule which is applied and we derive a contradiction in every case. In this proof, in order to simplify notations, we write E for E_j and H for H_j .

Simplify₁

We distinguish two cases:

Case 1: $s\sigma \mapsto_{R[H \cup E]} s'\sigma$ and $s \succ s'$. Assume that $e = f \in (H \cup E)$ has been used for proving the preconditions (with substitution θ). Consider $G \equiv e\theta \leftrightarrow_{e=f} f\theta$. By definition 3.4 $\{e\theta, f\theta\} \prec \{s\sigma\}$, hence $C(G) < (\{s\sigma\}, -)$ and therefore $C(G) < C(P)$. On the other hand $(H \cup E) \subset \cup_i E_i$, therefore G is valid in R and finally $R \models s'\sigma = t\sigma$. Let $Q \equiv s'\sigma \leftrightarrow_{s'=t} t\sigma$. We verify that $C(Q) < C(P)$:

- **If $s \succ t$ then** $C(P) = (\{s\sigma\}, \{t\sigma\}) < C(Q)$ since $\{s'\sigma, t\sigma\} \prec \{s\sigma\}$.
- **Otherwise:** $C(P) = (\{s\sigma, t\sigma\}, -)$
 - * **If $s' \succ t$ then** $C(Q) = (\{s'\sigma\}, \{t\sigma\}) < C(P)$ since $s'\sigma \prec s\sigma$.
 - * **If $s' \prec t$ then** $C(Q) = (\{t\sigma\}, \{s'\sigma\}) < C(P)$ since $\{t\sigma\} \prec \{s\sigma, t\sigma\}$.
 - * **Otherwise:** $C(Q) = (\{s'\sigma, t\sigma\}, -) < C(P)$ since $\{s'\sigma, t\sigma\} \prec \{s\sigma, t\sigma\}$.

Hence, Q is not valid in R and strictly smaller than $s\sigma \leftrightarrow_{s=t} t\sigma$. This contradicts the choice of $s = t$ (since $s' = t \in \cup_i E_i$).

Case 2: $t\sigma \mapsto_{R[H \cup E]} t'\sigma$ and $t \succ t'$. For the same reason as before we have: $R \not\models s\sigma = t'\sigma$. Let $Q \equiv s\sigma \leftrightarrow_{s=t'} t'\sigma$. We verify that $C(Q) < C(P)$:

- **If $s \succ t$ then $C(P) = (\{s\sigma\}, \{t\sigma\})$.**
In this case we have $s \succ t \succ t'$, hence $C(Q) = (\{s\sigma\}, \{t'\sigma\}) < C(P)$ since $t'\sigma \prec t\sigma$.
- **Otherwise: $C(P) = (\{s\sigma, t\sigma\}, -)$**
 - * **If $s \succ t'$ then $C(Q) = (\{s\sigma\}, \{t'\sigma\}) < C(P)$ since $\{s\sigma\} \prec \{s\sigma, t\sigma\}$.**
 - * **If $s \prec t'$ then $C(Q) = (\{t'\sigma\}, \{s\sigma\}) < C(P)$ since $t'\sigma \prec t\sigma$.**
 - * **Otherwise: $C(Q) = (\{s\sigma, t'\sigma\}, -) < C(P)$ since $\{s\sigma, t'\sigma\} \prec \{s\sigma, t\sigma\}$.**

Hence, Q is not valid in R and strictly smaller than $s\sigma \leftrightarrow_{s=t} t\sigma$, contradiction.

Simplify₂

We distinguish two cases:

Case 1: $s \mapsto_H s'$ and $s \succ s'$. Let $g = h$ be the equation of H which has been used to simplify s . Hence there exists τ such that $s \equiv s[g\tau]$, $s' \equiv s[h\tau]$ and $g \succ h$.

- **If $g = h$ is valid in R then $R \not\models s'\sigma = t\sigma$ and on the other hand $C(s'\sigma \leftrightarrow_{s'=t} t\sigma) < C(P)$.** This contradicts the choice of $(s = t)$ since it was assumed that $(s = t)$ justifies a minimal proof step of K which is not valid in R .
- **Otherwise: $R \not\models (g\tau)\sigma = (h\tau)\sigma$.**
 - **If $g\tau \triangleleft s$ then $C((g\tau)\sigma \leftrightarrow_{g=h} (h\tau)\sigma) < P$ since $\{(g\tau)\sigma, (h\tau)\sigma\} \prec \{s\sigma\}$, contradiction.**
 - **Otherwise: $g \succ h$.** Hence there exists k such that $\{g_i = h\sigma_i\}_{i=1\dots n} \subset E_k$ with $\{\sigma_i\}_{i=1\dots n}$ test-substitution for $(g = h)$ and $g\sigma_i \mapsto_{R[H \cup E]} g_i$. On the other hand let $\sigma' = (\tau\sigma)$, where σ' is a ground substitution. Assume that it is irreducible. Then there exists a test-substitution σ_{i_0} and a ground substitution θ such that $\sigma' = \sigma_{i_0}\theta$. Then $g_{i_0}\theta \leftrightarrow_{g_{i_0}=h\sigma_{i_0}} (h\sigma_{i_0})\theta$ with $g_{i_0} = h\sigma_{i_0} \in (\cup_i E_i)$ since $(g\sigma_{i_0})\theta \leftrightarrow_{g=h} (h\sigma_{i_0})\theta$ and $(g\sigma_{i_0})\theta \mapsto_{R[H \cup E]} g_{i_0}\theta$. Note that all the equations used in the proof of preconditions are valid (since they justify smaller proof-steps). Therefore, $R \not\models g_{i_0}\theta = (h\sigma_{i_0})\theta$. On the other hand, $C(g_{i_0}\theta \leftrightarrow_{g_{i_0}=h\sigma_{i_0}} (h\sigma_{i_0})\theta) < C(P)$ since $\{g_{i_0}\theta, (h\sigma_{i_0})\theta\} \prec \{s\sigma\}$ since $g_{i_0}\theta \prec (g\sigma_{i_0})\theta \preceq s\sigma$ and $(h\sigma_{i_0})\theta \prec (g\tau)\sigma \preceq s\sigma$. This contradicts the choice of $s = t$.

Case 2: $t \mapsto_H t'$. Same reasoning as Case 1.

Simplify₃

We distinguish two cases:

Case 1: $s \mapsto_E s'$ and $s \succ s'$. Let $g = h$ be the equation of E which is used to simplify s . Hence there exists τ such that $s \equiv s[g\tau]$, $g\tau \triangleleft s$, $s' \equiv s[h\tau]$ and $g\tau \succ h\tau$.

- **If $g = h$ is valid in R then $R \not\models s'\sigma = t\sigma$.** On the other hand $C(s'\sigma \leftrightarrow_{s'=t} t\sigma) < C(P)$. (same proof as for *Simplify₁*), contradiction.
- **Otherwise:** $R \not\models (g\tau)\sigma = (h\tau)\sigma$. Let $Q \equiv (g\tau)\sigma \leftrightarrow_{g=h} (h\tau)\sigma$, we verify that $C(Q) < C(P)$:
 - **If $g \succ h$ then $C(Q) = (\{(g\tau)\sigma\}, \{(h\tau)\sigma\}) < C(P)$ since $(g\tau)\sigma \prec s\sigma$.**
 - **If $g \prec h$ then absurd since $g\tau \succ h\tau$.**
 - **Otherwise:** $C(Q) = (\{(g\tau)\sigma, (h\tau)\sigma\}, -) < C(P)$ since $\{(g\tau)\sigma, (h\tau)\sigma\} \prec \{s\sigma\}$ due to the fact that $(g\tau)\sigma \prec s\sigma$ and $(h\tau)\sigma \prec (g\tau)\sigma \prec s\sigma$.

contradiction.

Case 2. $t \mapsto_E t'$. Same reasoning as Case 1.

Simplify₄

We distinguish two cases:

Case 1. $s \rightsquigarrow_H s'$, $s \star s'$, $s' \preceq t$ and $s \star t$. $C(P) = (\{s\sigma, t\sigma\}, -)$ since $s \star t$. On the other hand let $g = h$ be the equation of H which is used to simplify s . Hence there exists τ such that $s \equiv s[g\tau]$, $s' \equiv s[h\tau]$ and $g\tau \star h\tau$:

- If $s' \prec t$ then** we have $C((g\tau)\sigma \leftrightarrow_{g=h} (h\tau)\sigma) < C(P)$ since $\{(g\tau)\sigma, (h\tau)\sigma\} \prec \{s\sigma, t\sigma\}$ hence $R \models (g\tau)\sigma = (h\tau)\sigma$ and so: $R \not\models s'\sigma = t\sigma$ with $s' = t \in (\cup_i E_i)$. Let $Q \equiv s'\sigma \leftrightarrow_{s'=t} t\sigma$. We have $s' \prec t$ hence $C(Q) = (\{t\sigma\}, \{s'\sigma\}) < C(P)$, contradiction.
- Otherwise:** $s' = t$, in this case we have $R \not\models (g\tau)\sigma = (h\tau)\sigma$. Let $\sigma' = (\tau\sigma)$, where σ' is a ground substitution. Assume that it is irreducible, hence there exists σ_{i_0} a test-substitution and a ground substitution θ such that $\sigma' = \sigma_{i_0}\theta$. We distinguish two cases:

i. If $(g\sigma_{i_0})\theta \mapsto_{R[H \cup E]} g_{i_0}\theta$. Then $g_{i_0}\theta \leftrightarrow_{g_{i_0}=h\sigma_{i_0}} (h\sigma_{i_0})\theta$ with $(g_{i_0} = h\sigma_{i_0}) \in (\cup_i E_i)$ since $(g\sigma_{i_0})\theta \leftrightarrow_{g=h} (h\sigma_{i_0})\theta$. Let $Q \equiv g_{i_0}\theta \leftrightarrow_{g_{i_0}=h\sigma_{i_0}} (h\sigma_{i_0})\theta$, we verify that $C(Q) < C(P)$:

- * **If $g_{i_0} \succ h\sigma_{i_0}$ then $C(Q) = (\{g_{i_0}\theta\}, \{(h\sigma_{i_0})\theta\}) < C(P)$ since $g_{i_0}\theta \prec (g\sigma_{i_0})\theta \preceq s\sigma$.**
- * **If $g_{i_0} \prec h\sigma_{i_0}$ then $C(Q) = (\{(h\sigma_{i_0})\theta\}, \{g_{i_0}\theta\}) < C(P)$ since $(h\sigma_{i_0})\theta \preceq s'\sigma = t\sigma$.**
- * **Otherwise:** $C(Q) = (\{g_{i_0}\theta, (h\sigma_{i_0})\theta\}, -) < C(P)$ since $\{g_{i_0}\theta, (h\sigma_{i_0})\theta\} \prec \{s\sigma, t\sigma\}$.

Contradiction.

ii. If $(h\sigma_{i_0})\theta \mapsto_{R[H \cup E]} h_{i_0}\theta$. Then $(g\sigma_{i_0})\theta \leftrightarrow_{g\sigma_{i_0}=h_{i_0}} (h_{i_0})\theta$ with $(g\sigma_{i_0} = h_{i_0}) \in (\cup_i E_i)$ since $(g\sigma_{i_0})\theta \leftrightarrow_{g=h} (h\sigma_{i_0})\theta$. Let $Q \equiv (g\sigma_{i_0})\theta \leftrightarrow_{g\sigma_{i_0}=h_{i_0}} h_{i_0}\theta$, we verify that $C(Q) < C(P) = (\{s\sigma, t\sigma\}, -)$:

- * **If $g\sigma_{i_0} \succ h_{i_0}$ then** $C(Q) = (\{(g\sigma_{i_0})\theta\}, \{h_{i_0}\theta\}) < C(P)$ since $(g\sigma_{i_0})\theta \preceq s\sigma$.
- * **If $g\sigma_{i_0} \prec h_{i_0}$ then** $C(Q) = (\{h_{i_0}\theta\}, \{(g\sigma_{i_0})\theta\}) < C(P)$ since $h_{i_0}\theta \prec (h\sigma_{i_0})\theta \preceq s'\sigma = t\sigma$.
- * **Otherwise:** $C(Q) = (\{(g\sigma_{i_0})\theta, h_{i_0}\theta\}, -) < C(P)$ since $\{(g\sigma_{i_0})\theta, h_{i_0}\theta\} \prec \{s\sigma, t\sigma\}$.

Contradiction.

Case 2. $t \rightsquigarrow_H t'$. Same reasoning as 1.

Generate

Since σ is a ground irreducible substitution hence there exists a test-substitution σ_{i_0} and a ground substitution θ such that: $\sigma = \sigma_{i_0}\theta$. Assume that the rule *Generate* applies to $s = t$, then we distinguish two cases:

Case 1. $s \succ t$. Then, $(s\sigma_{i_0})\theta \mapsto_{R[H \cup E]} s_{i_0}\theta$. Therefore, $s_{i_0}\theta \leftrightarrow_{s_{i_0}=t\sigma_{i_0}} (t\sigma_{i_0})\theta$ with $(s_{i_0} = t\sigma_{i_0}) \in (\cup_i E_i)$ since $(s\sigma_{i_0})\theta \leftrightarrow_{s=t} (t\sigma_{i_0})\theta$. Let $Q \equiv s_{i_0}\theta \leftrightarrow_{s_{i_0}=t\sigma_{i_0}} (t\sigma_{i_0})\theta$, we verify that $C(Q) < C(P)$:

- **If $s \succ t$ then** $C(Q) < C(P)$ since $\{s_{i_0}\theta, (t\sigma_{i_0})\theta\} \prec \{s\sigma\}$.
- **Otherwise:** $C(P) = (\{s\sigma, t\sigma\}, -)$
 - * **If $s_{i_0} \succ t\sigma_{i_0}$ then** $C(Q) = (\{s_{i_0}\theta\}, \{(t\sigma_{i_0})\theta\}) < C(P)$ since $s_{i_0}\theta \prec s\sigma$.
 - * **If $s_{i_0} \prec t\sigma_{i_0}$ then** $C(Q) = (\{(t\sigma_{i_0})\theta\}, \{s_{i_0}\theta\}) < C(P)$ since $(t\sigma_{i_0})\theta = t\sigma$.
 - * **Otherwise:** $C(Q) = (\{s_{i_0}\theta, (t\sigma_{i_0})\theta\}, -) < C(P)$ since $\{s_{i_0}\theta, (t\sigma_{i_0})\theta\} \prec \{s\sigma, t\sigma\}$.

$R \not\models s_{i_0}\theta = (t\sigma_{i_0})\theta$ since all the equations used in the proof of preconditions are valid (since they justify smaller proof-steps). On the other hand, $C(Q) < C(P)$, contradiction.

Case 2. $s \not\succ t$, the situation is either as in A. or we have $(t\sigma_{i_0})\theta \mapsto_{R[H \cup E]} t_{i_0}\theta$. Then $(s\sigma_{i_0})\theta \leftrightarrow_{s\sigma_{i_0}=t_{i_0}} t_{i_0}\theta$ with $(s\sigma_{i_0} = t_{i_0}) \in (\cup_i E_i)$ since $(s\sigma_{i_0})\theta \leftrightarrow_{s=t} (t\sigma_{i_0})\theta$. Let $Q \equiv (s\sigma_{i_0})\theta \leftrightarrow_{s\sigma_{i_0}=t_{i_0}} t_{i_0}\theta$ with $(s\sigma_{i_0} = t_{i_0}) \in (\cup_i E_i)$. $C(P) = (\{s\sigma, t\sigma\}, -)$ since $s \not\succ t$, we verify that $C(Q) < C(P)$:

- **If $s\sigma_{i_0} \succ t_{i_0}$ then** $C(Q) = (\{(s\sigma_{i_0})\theta\}, \{t_{i_0}\theta\}) < C(P)$ since $(s\sigma_{i_0})\theta = s\sigma$.
- **If $s\sigma_{i_0} \prec t_{i_0}$ then** $C(Q) = (\{t_{i_0}\theta\}, \{(s\sigma_{i_0})\theta\}) < C(P)$ since $t_{i_0}\theta \prec t\sigma$.
- **Otherwise:** $C(Q) = (\{(s\sigma_{i_0})\theta, t_{i_0}\theta\}, -) < C(P)$ since $\{(s\sigma_{i_0})\theta, t_{i_0}\theta\} \prec \{s\sigma, t\sigma\}$.

contradiction. □



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

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

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

ISSN 0249 - 6399



★ R R - 2 1 8 7 ★