A. Bouhoula and F. Jacquemard

Automated Induction for Complex Data Structures

Research Report LSV-05-11
July 2005

Laboratoire Spécification et Vérification

Ecole Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex France

Automated Induction for Complex Data Structures*

Adel Bouhoula
École supérieure des
communications de Tunis
Cité Technologique des Communications
2083 Cité El Ghazala, TUNISIE
bouhoula@planet.tn

Florent Jacquemard
INRIA Futurs and LSV, UMR CNRS ENS Cachan
61 avenue du Président Wilson
94235 Cachan Cedex, FRANCE
florent.jacquemard@inria.fr

July 11, 2005

Abstract

In this paper, we develop a new approach for mechanizing induction on complex data structures (like sets, sorted lists, trees, powerlists...). The key idea is to compute a tree grammar with constraints which describes exactly the initial model of the given specification, unlike test sets or cover sets which are approximative induction schemes when the constructors are not free. This grammar is used for the generation of subgoals during the proof by induction. Our procedure is sound and refutationally complete even with constrained axioms for constructors. Our method subsumes all test set induction techniques, and yields very natural proofs for several examples on which other approaches failed.

1 Introduction

Data structures like bags, sorted lists, powerlists, square matrices, complete binary trees, *etc* are ubiquitous in software and hardware verification. These complex data structures are classically axiomatized by constrained equational theories. However, little and restrictive work has been carried out on mechanizing induction on such theories since they generate very complex induction schemes. In this paper, we develop a new approach for solving this problem by adapting and generalizing works on test set induction using tree automata with constraints.

Test set induction is a goal-directed proof technique which combines the full power of the two classical methods for automatic induction: explicit induction [9, 33, 27] and proof by consistency [26, 22, 19, 20, 1, 18, 12]. It works by computing an appropriate explicit induction scheme called *a test set*, to trigger the induction proof, and then applying a refutation principle using proof by consistency techniques [8, 5]. The Spike system, which implements this method, has been successful in proving difficult theorems with few interaction, like the Gilbreath card trick [8], the validation of the JavaCard platform [2], the verification of an ABR conformance algorithm [25], and the soundness of many digital circuits [3]. Test set induction as well as

*This work has been partially supported by a grant *SSHN* of the french institute for cooperation in the french ambassy in Tunisia.

explicit induction usually require that the specification is built with the constructor discipline. In this settings, the constructors are generally assumed to be *free* (no equation between constructor terms are allowed), which prevents the axiomatization of complex data structures.

The first author and Jouannaud [6] use tree automata techniques to generalize test set induction to specifications with relations between constructors. However, the axioms for constructors are assumed to be non-constrained and non-conditional *left-linear* rewrite rules, which is still too restrictive for the specification of structures like sets, sorted lists, *etc.*

Independently, Kapur [21] has proposed some ideas for mechanizing cover set induction if the constructors are not free. He defines particular specifications which may include in the declaration of function symbols (including constructors) some *applicability conditions*. This handles in particular the specification of powerlists or sorted lists, as illustrated by some examples.

In [29], Sengler proposes a method for automated termination analysis of recursively defined algorithm over data types like sets and arrays. His method can handle constructor relations, under restrictions. When it succeeds, his method provides an explicit induction scheme which can be used in explicit inductive theorem proving (on the opposite, our method is based on implicit induction).

Tree automata with constraints are tree recognizers whose transition rules are constrained by, *e.g.*, equality or disequality between subtrees. They allow us to construct a finite description of the initial model of equational specifications, even with non-left-linear rewrite rules (a set of representatives of the initial model is the language of ground normal forms in this case). For this reason, this formalism has been studied in many works related to inductive theorem proving, in particular, for the decision of ground reducibility [11, 16, 15] and proof by consistency methods [12]. One common problem in this setting is the decidability (and complexity) of the non-emptiness: does there exist at least one term accepted by a given automaton (with constraints)? Non-emptiness decidability imposes some restrictions on the kind and the shape of constraints present in the tree automata, and, since this property is required in the context of automated induction, this restricts correspondingly the equational theories considered. The study of interesting constraint classes and restrictions is the central problem of many theoretical studies on tree automata with constraints [4, 10, 16, 15].

We present here a new technique based on constrained tree automata (or grammars) for proof by induction in the lines of [8], which permits to handle complex data structures by allowing *constrained* rewrite rules between constructors. The main steps of our procedure are described below – the inference system is presented in Section 5, see also the example in Section 3. We start with a conjecture (goal) C and a rewrite system $\mathcal{R}$, with a subset $\mathcal{R}_{\mathcal{C}}$ of constructor rewrite rules.

1. computation of a constrained grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ (it generates the set of ground constructor terms in normal form),
2. for each goal (or subgoal) C,
   (a) generate instances of C by using the production rules of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ (instead of a test set),
   (b) reduce each instance obtained using the axioms and the induction hypotheses to obtain a new subgoal S.

   C becomes then an *induction hypothesis*,

3. for each subgoal S,
   if S is a tautologie or S is subsumed by an axiom or an induction hypothesis
   then delete S
   else if S is a ground irreducible constructor clause
   then if S is valid in the initial model
   then delete S
   else disproof (S is a counterexample)
   else S becomes a new subgoal, go to 2.

If every subgoal is deleted, then C is an inductive theorem of $\mathcal{R}$. Termination of the process may be achieved if necessary by incorporating appropriate lemmas. For the purpose of 1, we generalize the construction of a constrained tree automaton characterizing a normal form language from rewrite rules (see, *e.g.*, [15]) to constrained rewrite rules – in Section 4. Both tests of ground irreducibility and validity in the initial model are performed by mean of a non-emptiness test on a constrained grammar constructed from S and $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ (Section 6).

Using tree automata (or grammars) with constraints in the setting of automated induction is not a new idea (see, *e.g.*, [6, 12]). Our method however pushes the idea one step beyond, since it is based only on constrained tree grammars, which are used here both as induction scheme and decision procedure for consistency check, as long as emptiness is decidable. This makes real the idea that constrained tree grammars can be fully integrated in an automated inductive theorem proving procedure, and permits to generalize the previous methods, highlighting former theoretical works on tree automata with constraints. Some other facts in favor of our method are that:

1. constrained tree grammars describe exactly the initial model, unlike test sets and cover sets which are approximative when the constructors are not free,
2. it subsumes all known test set induction procedures [8, 5, 6],
3. it is sound, it is refutationally complete (any conjecture that is not valid in the initial model will be disproved), and it allows the refutation of false conjectures even with constrained axioms for constructors,
4. with constraints in constructor rules, it is possible in some cases to transform a non-terminating theory into a (constrained) terminating one, using the constrained completion technique of [24],
5. it does not require termination of the whole set of rules, unlike *e.g.* [6].

2 Preliminaries

The reader is assumed familiar with the basic notions of term rewriting [17] and mathematical logic. Notions and notations not defined here are standard.

Terms and substitutions. We assume given a many sorted signature $(\mathcal{S}, \mathcal{F})$ (or simply $\mathcal{F}$, for short) where $\mathcal{S}$ is a set of *sorts* and $\mathcal{F}$ is a finite set of function symbols with arities. We assume moreover that the signature $\mathcal{F}$ comes in two parts, $\mathcal{F} = \mathcal{C} \uplus \mathcal{D}$ where $\mathcal{C}$ a set of *constructor symbols*, and $\mathcal{D}$ is a set of *defined symbols*.

Let $\mathcal{X}$ be a family of sorted variables. We sometimes denote variables with sort exponent like x^S in order to indicate that x has sort $S \in \mathcal{S}$. The set of well-sorted terms over $\mathcal{F}$

(resp. constructor well-sorted terms) with variables in $\mathcal{X}$ will be denoted by $\mathcal{T}(\mathcal{F},\mathcal{X})$ (resp. $\mathcal{T}(\mathcal{C},\mathcal{X})$). The subset of $\mathcal{T}(\mathcal{F},\mathcal{X})$ (resp. $\mathcal{T}(\mathcal{C},\mathcal{X})$) of variable-free terms, or *ground* terms, is denoted $\mathcal{T}(\mathcal{F})$ (resp. $\mathcal{T}(\mathcal{C})$). We assume that each sort contains a ground term. The sort of a term $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$ is denoted $sort(t)$.

A term t is identified as usual to a function from its set of *positions* (strings of positive integers) $\mathcal{P}os(t)$ to symbols of $\mathcal{F}$, where positions are strings of positive integers. We note Λ the empty string (root position). The length of a position p is denoted $|p|$. The *depth* of a term t, denoted $depth(\mathcal{R})$, is the maximum of $\{|p| \mid p \in \mathcal{P}os(t)\}$. The *subterm* of t at position p is denoted by $t|_p$. The result of replacing $t|_p$ with s at position p in t is denoted by $t[s]_p$. This notation is also used to indicate that s is a subterm of t, in which case p may be omitted. We note $var(t)$ the set of variables occurring in t. A term t is *linear* if every variable of $var(t)$ occurs exactly once in t.

A substitution is a finite mapping $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ where $x_1, \ldots, x_n \in \mathcal{X}$ and $t_1, \ldots t_n \in \mathcal{T}(\mathcal{F},\mathcal{X})$. As usual, we identify substitutions with their morphism extension to terms. A variable renaming is a (well) sorted bijective substitution which maps variables to variables. A substitution σ is *grounding* for a term t if the domain of σ contains all the variables of t and the codomain of σ contains only ground terms. We use postfix notation for substitutions application and composition. We write $\mathrm{mgu}(s,t)$ to denote the most general unifier of terms s and t.

Constraints for terms and clauses. We assume given a constraint language $\mathcal{L}$, which is a finite set of predicate symbols with a recursive Boolean interpretation in the domain of ground constructor terms of $\mathcal{T}(\mathcal{C})$. Typically, $\mathcal{L}$ contains the syntactic equality $. \approx .$ (syntactic disequality $. \not\approx .$) and some (recursive) simplification ordering $. \prec .$ on ground constructor terms. *Constraints* on the language $\mathcal{L}$ are Boolean combinations of atoms of the form $P(t_1, \ldots, t_n)$ where $P \in \mathcal{L}$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C},\mathcal{X})$. By convention, an empty combination is interpreted to true.

We may extend the application of substitutions from terms to constraints in a straightforward way, and therefore define a solution for a constraint c as a (constructor) substitution σ grounding for all terms in c and such that $c\sigma$ is interpreted to true. The set of solutions of the constraint c is denoted $sol(c)$. A constraint c is *satisfiable* if $sol(c) \neq \emptyset$ (and *unsatisfiable* otherwise).

A *constrained term* $t \,[\![c]\!]$ is a linear term $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$ together with a constraint c, which may share some variables with t. Note that the assumption that t is linear is not restrictive, since any non linearity may be expressed in the constraint, for instance $f(x,x) \,[\![c]\!]$ is semantically equivalent to $f(x_1,x_2) \,[\![c \wedge x_1 \approx x_2]\!]$. A *literal* is an equation $s = t$ or a disequation $s \neq t$ or an oriented equation $s \to t$ between two terms. A *clause* is a disjunction of literals. We find convenient to see clauses themselves as terms on a signature extended by the predicate symbols $=$, $\neq$, and $\to$ and the connective $\vee$ (or $\Rightarrow$). This way, we can define a *constrained clause* as a constrained term. A constrained clause $C \,[\![c]\!]$ is said to *subsume* a constrained clause $C' \,[\![c']\!]$ if there is a substitution σ such that $C\sigma$ is a sub-clause of C' and $c'\sigma \wedge \neg c$ is unsatisfiable.

A *tautology* is a constrained clause $s_1 = t_1 \wedge \ldots \wedge s_n = t_n \,[\![c \wedge d]\!]$ such that d is a conjunctions of equational constraints, $d = u_1 \approx v_1 \wedge \ldots \wedge u_k \approx v_k$ and for all $i \in [1..n]$, $s_i\sigma = t_i\sigma$ where σ is the mgu of d.

Orderings. We assume given a well-founded *precedence* ordering total $\mathcal{F}$, denoted $\succ_{\mathcal{F}}$ or $\succ$ for short, compatible with the arity of function symbols (*i.e.* if $f \in \mathcal{F}$ has higher arity than

$g \in \mathcal{F}$ then $f \succ g$). A *reduction ordering* is a well-founded ordering on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ monotonic wrt contexts and substitutions. A *simplification ordering* is a reduction ordering which moreover contains the strict subterm ordering.

We assume from now on given a simplification ordering $>$ total on $\mathcal{T}(\mathcal{F})$, defined, *e.g.*, on the top of $\succ$ as an rpo $\succ_{rpo}$ [17].

The *multiset* extension $>^{mul}$ of an ordering $>$ is defined as the smallest ordering relation on multisets such that $M \cup \{t\} >^{mul} M \cup \{s_1, \ldots, s_n\}$ if $t > s_i$ for all $i \in [1..n]$.

The extension $>_e$ of the ordering $>$ on terms to atoms is defined as the multiset extension $>^{mul}$ to the multisets containing left and right members of the atoms. The extension of the ordering $>$ on terms to clauses is the multiset extension $>_e^{mul}$ applied to the literals.

Constrained rewriting. A *conditional constrained rewrite rule* is a constrained clause of the form $\Gamma \Rightarrow l \to r \,[\![c]\!]$ such that Γ is a conjunction of equations $u = v$, called the *condition* of the rule, the terms l and r (called resp. left- and right-hand side) are linear and have the same sort, and c is a constraint. When the condition Γ is empty, it is called a *constrained rewrite rule*. A set of conditional constrained, resp. constrained, rules is called a *conditional constrained* (resp. *constrained*) *rewrite system*.

Let $\mathcal{R}$ be a conditional constrained rewrite system. A term $t[l\sigma]\,[\![d]\!]$ rewrites to $t[r\sigma]\,[\![d]\!]$ by a rule $\rho = \Gamma \Rightarrow l \to r \,[\![c]\!] \in \mathcal{R}$, written $t[l\sigma]\,[\![d]\!] \xrightarrow[\rho,\sigma]{} t[r\sigma]\,[\![d]\!]$ or simply $t[l\sigma]\,[\![d]\!] \to t[r\sigma]\,[\![d]\!]$ if the substitution σ is such that $d\sigma \wedge \neg c\sigma$ is unsatisfiable and $u\sigma \downarrow_{\mathcal{R}} v\sigma$ for all $u = v \in \Gamma$.

The rewrite relation induced by the rules of a (conditional) constrained system $\mathcal{R}$, and its transitive and the reflexive transitive closures, are respectively denoted $\xrightarrow[\mathcal{R}]{}$, $\xrightarrow[\mathcal{R}]{+}$ and $\xrightarrow[\mathcal{R}]{*}$, and $u \downarrow_{\mathcal{R}} v$ stands for $\exists w,\ u \xrightarrow[\mathcal{R}]{*} w \xleftarrow[\mathcal{R}]{*} v$.

Note the semantical difference between conditions and constraints in rewrite rules. The validity of the condition is defined wrt the system $\mathcal{R}$ whereas the interpretation of constraint is fixed and independent from $\mathcal{R}$.

A constrained term $s\,[\![c]\!]$ is *reducible* by $\mathcal{R}$ if there is some $t\,[\![c]\!]$ such that $s\,[\![c]\!] \xrightarrow[\mathcal{R}]{} t\,[\![c]\!]$. Otherwise $s\,[\![c]\!]$ is called *irreducible*, or an $\mathcal{R}$-normal form. A substitution σ is *irreducible* by $\mathcal{R}$ if its codomain contains only $\mathcal{R}$-normal forms. A constrained term $t\,[\![c]\!]$ is *ground reducible* (resp. *ground irreducible*) if $t\sigma$ is reducible (resp. irreducible) for every irreducible solution σ of c grounding for t.

The system $\mathcal{R}$ is terminating if there is no infinite sequence $t_1 \xrightarrow[\mathcal{R}]{} t_2 \xrightarrow[\mathcal{R}]{} \ldots$, $\mathcal{R}$ is ground confluent if for any ground terms $u, v, w \in \mathcal{T}(\mathcal{F})$, $v \xleftarrow[\mathcal{R}]{*} u \xrightarrow[\mathcal{R}]{*} w$, implies then $v \downarrow_{\mathcal{R}} w$, and $\mathcal{R}$ is *ground convergent* if $\mathcal{R}$ is both ground confluent and terminating.

The *depth* of a non-empty set $\mathcal{R}$ of rules, denoted $depth(\mathcal{R})$, is the maximum of the depths of the left-hand sides of rules in $\mathcal{R}$.

Constructor specifications. We assume from now on given a conditional constrained rewrite system $\mathcal{R}$. The subset of constrained rewrite rules of $\mathcal{R}$ containing only function symbols from $\mathcal{C}$ is denoted $\mathcal{R}_{\mathcal{C}}$ and $\mathcal{R} \setminus \mathcal{R}_{\mathcal{C}}$ is denoted $\mathcal{R}_{\mathcal{D}}$.

Inductive theorems. A clause C is a *deductive theorem* of $\mathcal{R}$ (denoted $\mathcal{R} \models C$) if it is valid in any model of $\mathcal{R}$. A clause C is an *inductive theorem* of $\mathcal{R}$ (denoted $\mathcal{R} \models_{\mathcal{I}nd} C$) iff for all for all substitution σ grounding for C, $\mathcal{R} \models C\sigma$.

We shall need below to generalize the definition of inductive theorems to constrained clauses as follows: A constrained clause $C\,[\![c]\!]$ is an *inductive theorem* of $\mathcal{R}$ (denoted $\mathcal{R} \models_{\mathcal{I}nd} C\,[\![c]\!]$) if for all substitutions $\sigma \in sol(c)$ we have $\mathcal{R} \models C\sigma$.

Completeness. A function symbol $f \in \mathcal{D}$ is sufficiently complete wrt $\mathcal{R}$ iff for all $t_1, \ldots, t_n$ in $\mathcal{T}(\mathcal{C})$, there exists t in $\mathcal{T}(\mathcal{C})$ such that $f(t_1, \ldots, t_n) \xrightarrow[\mathcal{R}]{+} t$. We say that the system $\mathcal{R}$ is

sufficiently complete iff every defined operator $f \in \mathcal{D}$ is sufficiently complete wrt $\mathcal{R}$.

Let $f \in \mathcal{D}$ be a function symbol and let

$$\{\Gamma_1 \Rightarrow f(t_1, \ldots, t_n) \to r_1 \, [\![c_1]\!], \ldots, \Gamma_n \Rightarrow f(t_1, \ldots, t_n) \to r_n \, [\![c_n]\!]\}$$

be the subset of rules of $\mathcal{R}_\mathcal{D}$ whose left-hand sides are identical up to a variable renaming $\{\mu_1, \ldots, \mu_n\}$ *i.e.* $f(t_1, \ldots, t_n)\mu_1 = f(t_2, \ldots, t_2)\mu_2 = \ldots f(t_n, \ldots, t_n)\mu_n$.

We say that f is strongly complete wrt $\mathcal{R}$ (see [5]) if f is sufficiently complete wrt $\mathcal{R}$ and $\mathcal{R} \models_{\mathcal{I}nd} \Gamma_1\mu_1 \, [\![c_1\mu_1]\!] \vee \ldots \vee \Gamma_n\mu_n \, [\![c_n\mu_n]\!]$.

The system $\mathcal{R}$ is said strongly complete if every function symbol $f \in \mathcal{D}$ is strongly complete wrt $\mathcal{R}$.

3 Example

Consider the following signature:

$$\begin{array}{rcl} \mathcal{S} & = & \{Bool, Nat, Set\} \\ \mathcal{C} & = & \{true : Bool,\ false : Bool,\ 0 : Nat,\ s : Nat \to Nat,\ \emptyset : Set,\ ins : Nat \times Set \to Set\} \end{array}$$

and a constructor rewrite system for ordered lists without duplication,

$$\mathcal{R}_\mathcal{C} \ = \ \left\{ \begin{array}{rcl} ins(x, ins(y, z)) & \to & ins(y, z) \, [\![x \approx y]\!] \\ ins(x, ins(y, z)) & \to & ins(y, ins(x, z)) \, [\![x \succ y]\!] \end{array} \right\}$$

We complete this signature with the following set of defined function symbols:

$$\mathcal{D} \ = \ \{\in : Nat \times Set \to Bool, sorted : Set \to Bool, min : Set \to Bool\}$$

and let $\mathcal{R}_\mathcal{D} = \left\{ \begin{array}{rcl} sorted(\emptyset) & \to & true \\ sorted(ins(x, \emptyset)) & \to & true \\ sorted(ins(x, ins(y, z))) & \to & sorted(ins(y, z)) \, [\![x \prec y]\!] \\ min(ins(x, \emptyset)) & \to & x \\ min(ins(x, ins(y, z))) & \to & min(ins(x, z)) \, [\![x \prec y]\!] \end{array} \right\}$

Note the the defined functions $\in$ and *sorted* are sufficiently complete wrt $\mathcal{R}$. Indeed, a rule of the form:

$$sorted(ins(x, ins(y, z))) \to sorted(ins(y, z)) \, [\![x \succeq y]\!]$$

is not necessary since, intuitively, it can be simplified, using the constructor rules of $\mathcal{R}_\mathcal{C}$ into:

$$\begin{array}{rcl} sorted(ins(y, z)) & \to & sorted(ins(y, z)) \, [\![x \approx y]\!] \\ sorted(ins(y, ins(x, z))) & \to & sorted(ins(y, z)) \, [\![x \succ y]\!] \end{array}$$

The first one is a tautology and the second is a rule of $\mathcal{R}_\mathcal{D}$.

The function *min* is not sufficiently complete wrt $\mathcal{R}$ (the case $min(\emptyset)$ is missing).

Let us prove by induction the following conjecture:

$$sorted(x) = true \tag{1}$$

3.1 Test set induction

Roughly, the principle of a proof by test set induction [8, 5] is the one presented in introduction (with steps 1, 2, 3) except that:

1. the induction scheme is a test set, *i.e.* a set of constructor terms ($\mathcal{R}$ is required to be sufficiently complete),
2. (a) induction variables in the goals are instantiated by terms from the test set (see [5]).

Let us try to prove (1) using the test set induction technique. A test set for $\mathcal{R}$ (and sort *Set*) has to contain:

$$\mathcal{TS}(Set, \mathcal{R}) = \{\emptyset,\ ins(x_1, \emptyset),\ ins(x_2, ins(x_3, x_4))\}$$

We start by replacing x in (1) by the terms from the test set $\mathcal{TS}(Set, \mathcal{R})$, and obtain:

$$sorted(\emptyset) = true \tag{2}$$
$$sorted(ins(x_1, \emptyset)) = true \tag{3}$$
$$sorted(ins(x_2, ins(x_3, x_4))) = true \tag{4}$$

Subgoals (2) and (3) are simplified by $\mathcal{R}_\mathcal{D}$ into $true = true$ which is a tautology. Subgoal (4) cannot be simplified by $\mathcal{R}_\mathcal{D}$, because of the constraints in rewrite rules. Subgoal (4) does not contain any induction variable, and therefore, it cannot be further instantiated. So, the proof stops without a conclusion. Hence, we fail to prove conjecture (1) with test set induction technique.

3.2 Constrained test set induction

Let us now try to prove (1) with an adaptation of test set induction for constrained rewriting. Since $\mathcal{R}_\mathcal{C}$ is a constrained rewrite system, we could imagine to use as a test set for $\mathcal{R}$ (and sort *Set*) a set of constrained terms. The following set seems a reasonable choice of such a "constrained test set":

$$\mathcal{CTS}(Set, \mathcal{R}) = \{\emptyset,\ ins(x_1, \emptyset),\ ins(x_2, ins(x_3, x_4))\ [\![x_2 \prec x_3]\!]\}$$

Replacing x in (1) by the terms from the constrained test set $\mathcal{CTS}(Set, \mathcal{R})$, we obtain the subgoals (2) and (3) and:

$$sorted(ins(x_2, ins(x_3, x_4))) = true\ [\![x_2 \prec x_3]\!] \tag{5}$$

Subgoals (2) and (3) are simplified by $\mathcal{R}_\mathcal{D}$, as previously, into a tautology. Let us prove subgoal (5). Subgoal (5) is simplified by $\mathcal{R}_\mathcal{D}$ into

$$sorted(ins(x_3, x_4)) = true \tag{6}$$

We replace x_4 in (6) by the terms from the test set $\mathcal{CTS}(Set, \mathcal{R})$, and obtain (3) (up to variable renaming) and :

$$sorted(ins(x_3, ins(x_1, \emptyset))) = true \tag{7}$$
$$sorted(ins(x_3, ins(x_4, ins(x_5, x_6)))) = true\ [\![x_4 \prec x_5]\!] \tag{8}$$

Subgoal (3) is simplified into a tautology as above. Subgoals (7) and (8) cannot be simplified by $\mathcal{R}_\mathcal{D}$ because the constraint $x_3 \prec x_1$, resp. $x_3 \prec x_4$, is missing, and the proof stops without a conclusion. Note that such constraints cannot be added by simple test set instantiation of variables. Hence, we fail to prove conjecture (1) with the "constrained test set induction" technique.

3.3 Our Approach with constrained grammars

As seen above, we need to add appropriate constraints while instantiating the induction goals. This is precisely what constrained tree grammars permit. These constrained terms generators are presented formally in Section 4, we shall just give here a flavor of them and their use on the automatic inductive proof of conjecture (1). The set of ground $\mathcal{R}_\mathcal{C}$-normal forms is described by the following set of patterns:

$$\begin{array}{rcl} \mathrm{NF}(\mathcal{R}_\mathcal{C}) = & & \{x : Bool\} \cup \{x : Nat\} \cup \{\emptyset\} \cup \{ins(x, \emptyset) \mid x : Nat\} \\ & \cup & \{ins(x, ins(y, z)) \mid x, y : Nat, ins(y, z) \in \mathrm{NF}(\mathcal{R}_\mathcal{C}), x \prec y\} \end{array}$$

We build a constrained grammar which generates $\mathrm{NF}(\mathcal{R}_\mathcal{C})$ by means of non-terminal replacement guided by some production rules. The four first subsets of $\mathrm{NF}(\mathcal{R}_\mathcal{C})$ are generated by a tree grammar from the four non-terminals: $\{\llcorner x^{Bool} \lrcorner, \llcorner x^{Nat} \lrcorner, \llcorner x^{Set} \lrcorner, \llcorner ins(x_1, x_2) \lrcorner\}$ and using the production rules:

$$\begin{array}{lcllcl} \llcorner x^{Bool} \lrcorner & := & true & \llcorner x^{Bool} \lrcorner & := & false \\ \llcorner x^{Nat} \lrcorner & := & 0 & \llcorner x^{Nat} \lrcorner & := & s(\llcorner x^{Nat} \lrcorner) \\ \llcorner x^{Set} \lrcorner & := & \emptyset & \llcorner ins(x_1, x_2) \lrcorner & := & ins(\llcorner x^{Nat} \lrcorner, \llcorner x^{Set} \lrcorner) \end{array}$$

For the last subset of $\mathrm{NF}(\mathcal{R}_\mathcal{C})$, we need to apply the negation of the constraint $x \approx y \vee x \succ y$ in the production rules of the grammar. For this purpose, we add the production rule:

$$\llcorner ins(x_1, x_2) \lrcorner \quad := \quad ins(\llcorner x^{Nat} \lrcorner, \llcorner ins(x_1, x_2) \lrcorner) \, [\![x^{Nat} \prec x_1]\!]$$

Let us first "decorate" (1) by attaching some non-terminals (of the appropriate sort) to the variable x (for sake of simplicity, we use below a notation slightly different from the formal one of Section 4):

$$sorted(\llcorner x^{Set} \lrcorner) = true \tag{9}$$

$$sorted(\llcorner ins(x_1, x_2) \lrcorner) = true \tag{10}$$

Our procedure, presented in Section 5, roughly works as follows: given a conjecture C, we try to apply the production rules of the normal form grammar to C until all obtained clauses can be rewritten under $\mathcal{R}$, or induction hypotheses or either other conjectures not yet proved provided they are smaller than C. If this succeeds, the clauses obtained after simplification are considered as new subgoals and for their proof, we can use C as an induction hypothesis.

Let us apply this principle to our example. Applying the production rules to (9) gives: $sorted(\emptyset) = true$ which is reduced by $\mathcal{R}_\mathcal{D}$ into $true = true$, a tautology.

From (10) we obtain by the production rules the following analogous of (3) and (5):

$$sorted(ins(\llcorner x^{Nat} \lrcorner, \llcorner x^{Set} \lrcorner) = true \tag{3'}$$

$$sorted(ins(\llcorner x\lrcorner^{Nat}, \llcorner ins(x_1, x_2)\lrcorner) = true \, [\![x^{Nat} \prec x_1]\!] \quad (5')$$

A further application of the production rules to (3') returns the clause:

$$sorted(ins(\llcorner x\lrcorner^{Nat}, \emptyset) = true$$

which is simplified by $\mathcal{R}_{\mathcal{D}}$ into a tautology as above.

Concerning (5'), the next application of the production rules returns the following clauses (7') and (8'):

$$sorted(ins(\llcorner x_1\lrcorner^{Nat}, ins(\llcorner x_2\lrcorner^{Nat}, \llcorner x_3\lrcorner^{Set}))) = true \, [\![x_1^{Nat} \prec x_2^{Nat}]\!] \quad (7')$$

$$sorted(ins(\llcorner x_1\lrcorner^{Nat}, ins(\llcorner x_2\lrcorner^{Nat}, \llcorner ins(x_3, x_4)\lrcorner))) = true \, [\![x_1^{Nat} \prec x_2^{Nat} \wedge x_2^{Nat} \prec x_3]\!] \quad (8')$$

The clause (7') is instantiated again by the grammar's production rules into:

$$sorted(ins(\llcorner x_1\lrcorner^{Nat}, ins(\llcorner x_2\lrcorner^{Nat}, \emptyset))) = true \, [\![x_1^{Nat} \prec x_2^{Nat}]\!] \quad (7'')$$

Following the procedure described above, we need now to simplify the obtained clauses (7") and (8'). Is it important to mention that, unlike the situation in Section 3.2 with Subgoals (7) and (8), (7") and (8') have the constraints which permit the necessary simplification, by $\mathcal{R}_{\mathcal{D}}$. It shows that constrained grammars are more suitable than constrained test set for our purpose, because they generate only the terms in $\mathcal{R}$-normal form.

Let us come back to the example. With $\mathcal{R}_{\mathcal{D}}$, (7") is simplified into:

$$sorted(ins(\llcorner x_2\lrcorner^{Nat}, \emptyset)) = true$$

The left-hand side of the latter equation is simplified by $\mathcal{R}_{\mathcal{D}}$ into *true*. The clause (8') is rewritten by $\mathcal{R}_{\mathcal{D}}$ into:

$$sorted(ins(\llcorner x_2\lrcorner^{Nat}, \llcorner ins(x_3, x_4)\lrcorner)) = true \, [\![x_2^{Nat} \prec x_3]\!] \quad (11)$$

Now, we restart the process with the new subgoal (11) and the induction hypothesis (10). The clause (11) is actually subsumed by (10) (see the formal definitions in the next section) and can be deleted. This ends the proof that (1) is an inductive theorem of $\mathcal{R}$.

4 Constrained Grammars

Constrained tree grammars were first introduced in [11], also in the context of automated induction. As outlined in Section 3, they have the asset to offer an exact representation of the set of ground terms in a normal form for a given rewrite system $\mathcal{R}$, *i.e.* the initial model of $\mathcal{R}$ for inductive theorem. Their role is twofold in our approach: on one hand, a grammar generates subgoals, by replacing variables by terms, following production rules (therefore, it can be seen as induction schemes), on the other hand, some decision algorithms on the grammar are involved during the procedure, for instance to check ground irreducibility of the derived constrained clauses.

Definition 1 *A* constrained grammar $\mathcal{G} = (Q, \Delta)$ *is given by: 1. a finite set Q of non-terminals of the form $\llcorner u\lrcorner$, where u is a linear term of $\mathcal{T}(\mathcal{F}, \mathcal{X})$, 2. a finite set Δ of production rules of the form $\llcorner t\lrcorner := f(\llcorner u_1\lrcorner, \ldots, \llcorner u_n\lrcorner) \, [\![c]\!]$ where $f \in \mathcal{F}$, $\llcorner t\lrcorner$, $\llcorner u_1\lrcorner, \ldots, \llcorner u_n\lrcorner \in Q$ and c is a constraint.*

The non-terminals are always considered modulo variable renaming. In particular, we assume that the above term $f(u_1, \ldots, u_n)$ is linear, to avoid confusion in the constraint c.

4.1 Terms Generation

We associate to a given constrained grammar $\mathcal{G} = (Q, \Delta)$ a finite set of new unary constraints predicates of the form $.:\llcorner u \lrcorner$, where $\llcorner u \lrcorner \in Q$ (modulo variable renaming). Constraints of the form $t{:}\llcorner u \lrcorner$ called *membership constraints* and their interpretation is given below. The production relation between constrained terms $\vdash_{\mathcal{G},y}$, or $\vdash_y$ or $\vdash$ for short when $\mathcal{G}$ is clear from context, is defined by:

$$t[y]\,[\![y{:}\llcorner v \lrcorner \wedge d]\!] \vdash t[f(y_1, \ldots, y_n)]\,[\![y_1{:}\llcorner u_1\theta \lrcorner \wedge \ldots \wedge y_n{:}\llcorner u_n\theta \lrcorner \wedge c\theta \wedge d\tau\theta]\!]$$

if there exists $\llcorner v \lrcorner := f(\llcorner u_1 \lrcorner, \ldots, \llcorner u_n \lrcorner)\,[\![c]\!] \in \Delta$ and $y_1, \ldots, y_n$ are fresh variables, such that $f(u_1, \ldots, u_n) = v\tau$, θ is a variable renaming by fresh variables. The reflexive transitive and transitive closures of the relation $\vdash$ are respectively denoted $\vdash^*$ and $\vdash^+$.

Definition 2 *The language $L(\mathcal{G}, \llcorner u \lrcorner)$ is the set of ground terms t* generated *by a constrained grammar $\mathcal{G}$ in non-terminal $\llcorner u \lrcorner$, i.e. such that $y\,[\![y{:}\llcorner u \lrcorner]\!] \vdash^* t\,[\![c]\!]$ where c is satisfiable.*

Given $Q' \subseteq Q$, we note $L(\mathcal{G}, Q') = \bigcup_{\llcorner u \lrcorner \in Q'} L(\mathcal{G}, \llcorner u \lrcorner)$ and $L(\mathcal{G}) = L(\mathcal{G}, Q)$.
Given a constrained grammar $\mathcal{G} = (Q, \Delta)$, we can now define $sol(t{:}\llcorner u \lrcorner)$, where $\llcorner u \lrcorner \in Q$, as $\{\sigma \mid t\sigma \in L(\mathcal{G}, \llcorner u \lrcorner)\}$.

4.2 Normal Forms

The constrained grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}}) = (Q_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}}), \Delta_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}}))$ described in Figure 1 generates the language of ground $\mathcal{R}_{\mathcal{C}}$-normal forms. For the construction of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$, we add a new sort *Red* to $\mathcal{S}$, (the sort of reducible terms), and hence a new variable x^{Red}. The construction of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ is a generalization of the one of [15] (Section 3.1). Intuitively, it corresponds to the complementation and completion of a grammar for $\mathcal{R}_{\mathcal{C}}$-reducible terms (such a grammar does mainly pattern matching of left members of rewrite rules), where every subset of states (for the complementation) is represented by the most general unifiers of its elements (if any).

$$Q_0(\mathcal{R}_{\mathcal{C}}) = \{\llcorner u \lrcorner \mid u \trianglelefteq l \text{ for some } l \to r\,[\![c]\!] \in \mathcal{R}_{\mathcal{C}}\} \cup \{\llcorner x^S \lrcorner \mid S \in \mathcal{S}\}$$

$$Q_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}}) = \left\{ \begin{array}{c} \llcorner \mathrm{mgu}(t_1, \ldots, t_n) \lrcorner \mid \{\llcorner t_1 \lrcorner, \ldots, \llcorner t_n \lrcorner\} \text{ is a maximal} \\ \text{subset of } Q_0(\mathcal{R}_{\mathcal{C}}) \text{ s.t. } t_1, \ldots, t_n \text{ are unifiable} \end{array} \right\} \uplus \{\llcorner x^{Red} \lrcorner\}$$

$\Delta_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ contains:
every $\llcorner x^{Red} \lrcorner := f(\llcorner u_1 \lrcorner, \ldots, \llcorner u_n \lrcorner)\,[\![\ ]\!]$ such that one of the u_i at least is x^{Red},
every $\llcorner x^{Red} \lrcorner := f(\llcorner u_1 \lrcorner, \ldots, \llcorner u_n \lrcorner)\,[\![c]\!]$ and every $\llcorner t \lrcorner := f(\llcorner u_1 \lrcorner, \ldots, \llcorner u_n \lrcorner)\,[\![\neg c]\!]$
such that $f \in \mathcal{F}$ with profile $S_1, \ldots, S_n \to S$
and $\llcorner u_1 \lrcorner, \ldots, \llcorner u_n \lrcorner \in Q_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$, $u_1, \ldots, u_n$ have respective sorts $S_1, \ldots, S_n$
$t = \mathrm{mgu}\{u \mid \llcorner u \lrcorner \in Q_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}}) \text{ and } u \text{ matches } f(u_1, \ldots, u_n)\}$
$c \equiv \bigvee_{l \to r\,[\![e]\!] \in \mathcal{R}_{\mathcal{C}},\ t = l\theta} e\theta$

Figure 1: Constrained grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ for $\mathcal{R}_{\mathcal{C}}$-normal forms

Lemma 1 *For every term $t \in \mathcal{T}(\mathcal{C})$, $t \in L(\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}}), \llcorner u \lrcorner)$, for some $\llcorner u \lrcorner \in Q_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}}) \setminus \{\llcorner x^{Red} \lrcorner\}$ iff t is an $\mathcal{R}_{\mathcal{C}}$-normal form.*

Proof. We shall use the following Fact, which can be proved by a straightforward induction on the length of the derivation $y \, [\![y{:}\llcorner u \lrcorner]\!] \vdash^* t \, [\![c]\!]$.

Fact 1 *For each* $\llcorner u \lrcorner \in Q_{\mathrm{NF}}(\mathcal{R}_\mathcal{C}) \setminus \{ \llcorner x^{Red}_\lrcorner \}$*, and each* $t \in L(\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C}), \llcorner u \lrcorner)$*, t is an instance of u and moreover, u is the sup, wrt to subsumption, of* $\{ v \mid \llcorner u \lrcorner \in Q_{\mathrm{NF}}(\mathcal{R}_\mathcal{C}) \setminus \{ \llcorner x^{Red}_\lrcorner \}$ *and t is an instance of v*$\}$.

Let us now show the 'only if' direction by induction on the length of the derivation $y \, [\![y{:}\llcorner u \lrcorner]\!] \vdash^* t \, [\![c']\!]$ (where c' is satisfiable).

If the length is 1, then t is a nullary symbol of $\mathcal{C}$, and by construction t is $\mathcal{R}_\mathcal{C}$-irreducible.

If $y \, [\![y{:}\llcorner u \lrcorner]\!] \vdash f(y_1, \ldots, y_n) \, [\![y_1{:}\llcorner u_1\theta \lrcorner \wedge \ldots \wedge y_n{:}\llcorner u_n\theta \lrcorner \wedge c\theta]\!] \vdash^* t \, [\![c']\!] = f(t_1, \ldots, t_n) \, [\![c']\!]$ for some production rule $\llcorner u \lrcorner := f(\llcorner u_1 \lrcorner, \ldots, \llcorner u_n \lrcorner) \, [\![c]\!] \in \Delta_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ (θ is a variable renaming by fresh variables), then for every $i \in [1..n]$, $t_i \in L(\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C}), \llcorner u_i \lrcorner)$, and $\llcorner u_i \lrcorner \neq \llcorner x^{Red}_\lrcorner$ (otherwise we would have $\llcorner u \lrcorner = \llcorner x^{Red}_\lrcorner$). Hence, by induction hypothesis, every t_i is a $\mathcal{R}_\mathcal{C}$-normal form. Assume that t is $\mathcal{R}_\mathcal{C}$-reducible (it must then be reducible at root position), and let $l \to r \, [\![d]\!] \in \mathcal{R}_\mathcal{C}$ be such that $t = l\tau$, $\tau \in sol(d)$ and l is maximum wrt subsumption among the rules of $\mathcal{R}_\mathcal{C}$ satisfying these conditions. By construction, $u = l$ and $c = \neg d\sigma \wedge c'$. It follows from the satisfiability of c' that $\tau \in sol(c)$ (the variables of c are instantiated by ground terms in the above grammar derivation). This is in contradiction with $c = \neg d\sigma \wedge c'$ and $\tau \in sol(d)$.

We show now the 'if' direction by induction on t.

If t is a nullary function symbol of sort S and is $\mathcal{R}_\mathcal{C}$-irreducible, then t is not the left hand side of a rule of $\mathcal{R}_\mathcal{C}$, and $y \, [\![y{:}\llcorner x^S_\lrcorner]\!] \vdash t$.

If $t = f(t_1, \ldots, t_n)$ and is $\mathcal{R}_\mathcal{C}$-irreducible, then every t_i is $\mathcal{R}_\mathcal{C}$-irreducible for $i \in [1..n]$, hence by induction hypothesis, $t_i \in L(\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C}), \llcorner u_i \lrcorner)$ for some $\llcorner u_i \lrcorner \in Q_{\mathrm{NF}}(\mathcal{R}_\mathcal{C}) \setminus \{ \llcorner x^{Red}_\lrcorner \}$. It means that for all $i \in [1..n]$, there is a derivation of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ of the form $y \, [\![y{:}\llcorner u_i \lrcorner]\!] \vdash^* t_i \, [\![c_i]\!]$. By Fact 1, every t_i is an instance of u_i, hence $t = f(u_1, \ldots, u_n)\tau$ for some ground substitution τ. If there is a production rule $\llcorner u \lrcorner := f(\llcorner u_1 \lrcorner, \ldots, \llcorner u_n \lrcorner) \, [\![c]\!] \in \Delta_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$, with $\llcorner u \lrcorner \in Q_{\mathrm{NF}}(\mathcal{R}_\mathcal{C}) \setminus \{ \llcorner x^{Red}_\lrcorner \}$ and $\tau \in sol(c)$, then the following derivation is possible: $y \, [\![y{:}\llcorner u \lrcorner]\!] \vdash f(y_1, \ldots, y_n) \, [\![y_1{:}\llcorner u_1\theta \lrcorner \wedge \ldots \wedge y_n{:}\llcorner u_n\theta \lrcorner \wedge c\theta]\!] \vdash^* t \, [\![c']\!]$ where c' is satisfiable, and $t \in L(\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C}), \llcorner u \lrcorner)$. Assume that for every such production rule, we have $\tau \notin sol(c)$. It means by construction that there is a rule $u \to r \, [\![d]\!] \in \mathcal{R}_\mathcal{C}$ such that $\tau \in sol(d)$, hence that t is $\mathcal{R}_\mathcal{C}$-reducible, a contradiction. □

5 Inference System

In this section, we present our inductive theorem proving procedure. In the following, the relation $\vdash$ and the membership constraints shall be understood wrt $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$. Conjectures to be proved (goals) are clauses.

The inference and simplification rules below rely on an ordering defined on the top of the following complexity measure on clauses.

Definition 3 *The complexity of a constrained clause $C \, [\![c]\!]$ is the pair made of the two following components: C, ordered by the multiset extension of the ordering $>_e$ on literals, and the number of constraints $d\sigma$ not occurring in c, such that there exists $l \to r \, [\![d]\!] \in \mathcal{R}_\mathcal{C}$ and $l\sigma$ is a subterm of C.*

We denote $\gg$ the ordering on constrained clauses defined as the lexicographic composition of the orderings on the two components on the complexities.

5.1 Simplification Rules for Defined Functions

We describe in this section the simplification rules that we use for defined symbols, presented in Figure 2. Inductive Rewriting simplifies goals with axioms as well as with instances of the induction hypotheses, provided that they are smaller than the goal. The underlying induction principle is based on the well-founded ordering $\gg$ on constrained clauses, hence is more powerful than structural induction. Contextual Rewriting can be viewed as a generalization of the corresponding rule given in [32] to handle constraints. Rewrite Splitting simplifies a constrained clause which contains a subterm matching some left member of rule of $\mathcal{R}_\mathcal{D}$. The inference checks moreover that all cases are covered for the application of such rules of $\mathcal{R}_\mathcal{D}$, *i.e.* that for each ground substitution τ, the conditions and the constraints of at least one rule is true wrt τ. Note that this condition is always true when $\mathcal{R}$ is sufficiently complete, and hence that this check is superfluous in this case. Inductive Deletion deletes tautologies and clauses with unsatisfiable constraints.

Inductive Rewriting $\quad \big(\{C \,[\![c]\!]\}, \mathcal{H}\big) \to_\mathcal{D} \{C' \,[\![c]\!]\}$
if $\quad C \,[\![c]\!] \xrightarrow[\rho,\sigma]{} C' \,[\![c]\!]$, $l\sigma > r\sigma$ and $l\sigma > \Gamma\sigma$
where $\quad \rho = \Gamma \Rightarrow l \to r \,[\![c]\!] \in \mathcal{R}_\mathcal{D} \cup \{\psi \;\; | \;\; \psi \in \mathcal{H} \text{ and } C \,[\![c]\!] \gg \psi\}$

Contextual Rewriting $\quad \big(\{\Upsilon \Rightarrow C[l\sigma] \,[\![c]\!]\}, \mathcal{H}\big) \to_\mathcal{D} \{\Upsilon \Rightarrow C[r\sigma] \,[\![c]\!]\}$
if $\quad \mathcal{R} \models_{\mathcal{I}nd} \Upsilon \Rightarrow \Gamma\sigma \,[\![c \wedge c'\sigma]\!]$, $l\sigma > r\sigma$ and $\{l\sigma\} >^{mul} \Gamma\sigma$
where $\quad \Gamma \Rightarrow l \to r \,[\![c']\!] \in \mathcal{R}_\mathcal{D}$

Rewrite Splitting $\quad \big(\{C[t]_p \,[\![c]\!]\}, \mathcal{H}\big) \to_\mathcal{D} \{\Gamma_i\sigma_i \Rightarrow C[r_i\sigma_i]_p \,[\![c \wedge c_i\sigma_i]\!]\}_{i\in[1..n]}$
if $\quad \mathcal{R} \models_{\mathcal{I}nd} \Gamma_1\sigma_1 \,[\![c_1\sigma_1]\!] \vee \ldots \vee \Gamma_n\sigma_n \,[\![c_n\sigma_n]\!]$, $t > r_i\sigma_i$ and $\{t\} >^{mul} \Gamma_i\sigma_i$
where $\quad$ the $\Gamma_i\sigma_i \Rightarrow l_i\sigma_i \to r_i\sigma_i \,[\![c_i\sigma_i]\!]$, $i \in [1..n]$,
are all the instances of rules $\Gamma_i \Rightarrow l_i \to r_i \,[\![c_i]\!] \in \mathcal{R}_\mathcal{D}$ such that $l_i\sigma_i = t$

Inductive Deletion $\quad \big(\{C \,[\![c]\!]\}, \mathcal{H}\big) \to_\mathcal{D} \emptyset$
if $\quad C \,[\![c]\!]$ is a tautology or c is unsatisfiable

Figure 2: Simplification Rules for Defined Functions

5.2 Simplification Rules for Constructors

The simplification rules for constructors are presented in Figure 3. Rewriting simplifies goals with axioms from $\mathcal{R}_\mathcal{C}$. Partial Splitting eliminates ground reducible terms in a constrained clause $C \,[\![c]\!]$ by adding to $C \,[\![c]\!]$ the negation of constraint of some rules of $\mathcal{R}_\mathcal{C}$. Therefore, the saturated application of Partial splitting and Rewriting will always lead to Deletion or to ground irreducible constructor clauses. Finally, Deletion and Validity remove respectively tautologies and clauses with unsatisfiable constraints, and ground irreducible constructor theorems of $\mathcal{R}$.

5.3 Induction Inference Rules

The main inference system is displayed in Figure 4. Its rules apply to pairs $(\mathcal{E}, \mathcal{H})$, where $\mathcal{E}$ is the set of current conjectures and $\mathcal{H}$ is the *set* of inductive hypotheses (constrained clauses). Two inference rules below use the normal form grammar $\mathcal{G}_{\text{NF}}(\mathcal{R}_\mathcal{C})$ in order to instantiate

Rewriting $\{C \llbracket c \rrbracket\} \rightarrow_{\mathcal{C}} \{C' \llbracket c \rrbracket\}$
if $C \llbracket c \rrbracket \xrightarrow[\mathcal{R}_{\mathcal{C}}]{+} C' \llbracket c \rrbracket$ and $C \llbracket c \rrbracket \gg C' \llbracket c \rrbracket$

Partial Splitting $\{C[l\sigma]_p \llbracket c \rrbracket\} \rightarrow_{\mathcal{C}} \{C[r\sigma]_p \llbracket c \wedge c'\sigma \rrbracket, C[l\sigma]_p \llbracket c \wedge \neg c'\sigma \rrbracket\}$
if $l \rightarrow r \llbracket c' \rrbracket \in \mathcal{R}_{\mathcal{C}}$, $l\sigma > r\sigma$,
and neither $c'\sigma$ nor $\neg c'\sigma$ is a subformula of c

Deletion $\{C \llbracket c \rrbracket\} \rightarrow_{\mathcal{C}} \emptyset$
if $C \llbracket c \rrbracket$ is a tautology or c is unsatisfiable

Validity $\{C \llbracket c \rrbracket\} \rightarrow_{\mathcal{C}} \emptyset$
if $C \llbracket c \rrbracket$ is a ground irreducible constructor clause and $\mathcal{R} \models_{\mathcal{I}nd} C \llbracket c \rrbracket$

Figure 3: Simplification Rules for Constructors

variables. In order to be able to apply these inferences, according to the definition of term generation in Section 4.1, we shall initiate the process by adding to the conjectures one membership constraint for each variable.

Definition 4 *Let $C \llbracket c \rrbracket$ be a constrained clause such that c contains no membership constraint. The* decoration *of $C \llbracket c \rrbracket$, denoted $decorate(C \llbracket c \rrbracket)$ is the set of clauses $C \llbracket c \wedge x_1{:}\llcorner u_1 \lrcorner \wedge \ldots \wedge x_n{:}\llcorner u_n \lrcorner \rrbracket$ where $\{x_1, \ldots, x_n\} = var(C)$, and for all $i \in [1..n]$, $\llcorner u_i \lrcorner \in Q_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ and $sort(u_i) = sort(x_i)\}$*

The definition of *decorate* is extended to set of constrained clauses as expected.

A constrained clause $C \llbracket c \rrbracket$ is said *decorated* if $c = d \wedge x_1{:}\llcorner u_1 \lrcorner \wedge \ldots \wedge x_n{:}\llcorner u_n \lrcorner]$ where $\{x_1, \ldots, x_n\} = var(C)$, and for all $i \in [1..n]$, $\llcorner u_i \lrcorner \in Q_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$, $sort(u_i) = sort(x_i)\}$, and d does not contain membership constraints.

Simplification reduces a conjecture according to the rules shown in Figure 3. Inductive Simplification reduces a conjecture according to the rules shown in Figure 2. Inductive Narrowing generates new subgoals by application of the production rules of the constrained grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ until the obtained clause is deep enough to cover left hand side of rules of $\mathcal{R}_{\mathcal{D}}$. Each obtained clause must be simplified by one the rules shown in Figure 2. Narrowing eliminates ground reducible constructor terms in a clause by simplifying their instances, while deriving conjectures considered as new subgoals. The criteria for the above notion of "deep enough" is the same for Inductive Narrowing and Narrowing and is a bit rough, for sake of clarity of the inference rules. However, in practice, it be replaced to a tighter condition (with, *e.g.*, a distinction between $\mathcal{R}_{\mathcal{C}}$ and $\mathcal{R}_{\mathcal{D}}$) while preserving the results of the next section. Subsumption deletes clauses redundant with axioms of $\mathcal{R}$, induction hypotheses of $\mathcal{H}$ and other conjectures not yet proved (in $\mathcal{E}$).

For sake of efficiency, the application of Narrowing and Inductive Narrowing could be restricted to so called *induction variables*, as defined in [5], while preserving all the results of the next section.

5.4 Soundness and Completeness

Our inference system is sound, and refutationally complete.

Simplification
$$\frac{(\mathcal{E} \cup \{C \,[\![c]\!]\}, \mathcal{H})}{(\mathcal{E} \cup \mathcal{E}', \mathcal{H})}$$
if $\{C \,[\![c]\!]\} \rightarrow_{\mathcal{C}} \mathcal{E}'$

Inductive Simplification
$$\frac{(\mathcal{E} \cup \{C \,[\![c]\!]\}, \mathcal{H})}{(\mathcal{E} \cup \mathcal{E}', \mathcal{H})}$$
if $(\{C \,[\![c]\!]\}, \mathcal{E} \cup \mathcal{H}) \rightarrow_{\mathcal{D}} \mathcal{E}'$

Narrowing
$$\frac{(\mathcal{E} \cup \{C \,[\![c]\!]\}, \mathcal{H})}{(\mathcal{E} \cup \mathcal{E}_1 \cup \ldots \cup \mathcal{E}_n, \mathcal{H} \cup \{C \,[\![c]\!]\})}$$
if for all i in $[1..n]$, $depth(C_i) - depth(C) \leq depth(\mathcal{R}) - 1$
and $\{C_i \,[\![c_i]\!]\} \rightarrow_{\mathcal{C}} \mathcal{E}_i$
where $\{C_1 \,[\![c_1]\!], \ldots, C_n \,[\![c_n]\!]\}$ is the set of clauses such that $C \,[\![c]\!] \vdash^* C_i \,[\![c_i]\!]$

Inductive Narrowing
$$\frac{(\mathcal{E} \cup \{C \,[\![c]\!]\}, \mathcal{H})}{(\mathcal{E} \cup \mathcal{E}_1 \cup \ldots \cup \mathcal{E}_n, \mathcal{H} \cup \{C \,[\![c]\!]\})}$$
if for all i in $[1..n]$, $depth(C_i) - depth(C) \leq depth(\mathcal{R}) - 1$
and $(C_i \,[\![c_i]\!], \mathcal{E} \cup \mathcal{H} \cup \{C \,[\![c]\!]\}) \rightarrow_{\mathcal{D}} \mathcal{E}_i$
where $\{C_1 \,[\![c_1]\!], \ldots, C_n \,[\![c_n]\!]\}$ is the set of clauses such that $C \,[\![c]\!] \vdash^+ C_i \,[\![c_i]\!]$

Subsumption
$$\frac{(\mathcal{E} \cup \{C \,[\![c]\!]\}, \mathcal{H})}{(\mathcal{E}, \mathcal{H})}$$
if $C \,[\![c]\!]$ is subsumed by another clause of $\mathcal{R} \cup \mathcal{E} \cup \mathcal{H}$

Disproof
$$\frac{(\mathcal{E} \cup \{C \,[\![c]\!]\}, \mathcal{H})}{(\bot, \mathcal{H})}$$
if no other rule applies to the clause $C \,[\![c]\!]$

Figure 4: Induction Inference Rules

Definition 5 *We call* derivation *a sequence of inference steps generated by a pair of the form* $(\mathcal{E}_0, \emptyset)$*, using the inference rules in* $\mathcal{I}$*, written* $(\mathcal{E}_0, \emptyset) \vdash_{\mathcal{I}} (\mathcal{E}_1, \mathcal{H}_1) \vdash_{\mathcal{I}} \cdots (\mathcal{E}_n, \mathcal{H}_n) \vdash_{\mathcal{I}} \cdots$*. We say that a derivation is* fair *if the set of persistent constrained clauses* $(\cup_i \cap_{j \geq i} \mathcal{E}_j)$ *is empty or equal to* $\{\bot\}$*. The derivation is said to be a* disproof *in the latter case, and a* success *in the former.*

Finite success is obtained when the set of conjectures to be proved is exhausted. Infinite success is obtained when the procedure diverges, assuming fairness. When it happens, the clue is to guess some lemmas which are used to subsume or simplify the generated infinite family of subgoals, therefore stopping the divergence. This is possible in our approach, since lemmas can be used in the same way as axioms are.

Theorem 1 (soundness of successful derivations) *Assume that* $\mathcal{R}_{\mathcal{C}}$ *is terminating and that* $\mathcal{R}$ *is sufficiently complete. Let* $\mathcal{G}_0$ *be a set of unconstrained clauses and let* $\mathcal{E}_0 = decorate(\mathcal{G}_0)$*, if there exists a successful derivation* $(\mathcal{E}_0, \emptyset) \vdash_{\mathcal{I}} (\mathcal{E}_1, \mathcal{H}_1) \vdash_{\mathcal{I}} \cdots$ *then* $\mathcal{R} \models_{\mathcal{I}nd} \mathcal{G}_0$*.*

Proof. Assume that $\mathcal{R} \not\models_{\mathcal{I}nd} \mathcal{G}_0$, and let $(\mathcal{E}_0, \emptyset) \vdash_{\mathcal{I}} (\mathcal{E}_1, \mathcal{H}_1) \vdash_{\mathcal{I}} \cdots$ be an arbitrary successful derivation. By the following Fact, we have that $\mathcal{R} \not\models_{\mathcal{I}nd} \mathcal{E}_0$.

Fact 2 *Assume that $\mathcal{R}_\mathcal{C}$ is terminating and that $\mathcal{R}$ is sufficiently complete. If $\mathcal{R} \models_{\mathcal{I}nd} \mathcal{E}_0$ then $\mathcal{R} \models_{\mathcal{I}nd} \mathcal{G}_0$.*

Proof. Assume that $\mathcal{R} \models_{\mathcal{I}nd} \mathcal{E}_0$ and that for some clause $C \in \mathcal{G}_0$ we have $\mathcal{R} \not\models_{\mathcal{I}nd} C$. Let $\{C\,[\![c_1]\!], \ldots, C\,[\![c_n]\!]\} = \mathit{decorate}(C)$. For all $i \in [1..n]$, we have $\mathcal{R} \models_{\mathcal{I}nd} C\,[\![c_i]\!]$, but there exists $\sigma \notin \cup_{i=1}^{n} \mathit{sol}(c_i)$ such that $\mathcal{R} \not\models C\sigma$. Since $\mathcal{R}$ is sufficiently complete and $\mathcal{R}_\mathcal{C}$ is terminating, we can rewrite σ into a constructor and $\mathcal{R}_\mathcal{C}$-irreducible ground substitution σ'. By Lemma 1, it follows that $\sigma \in \mathit{sol}(c_i)$ for some $i \in [1..n]$, and therefore that $\mathcal{R} \models C\sigma'$, a contradiction. □

Let D_0 be a clause, minimal wrt $\gg$, in the set:

$$\left\{ D\sigma \;\middle|\; D\,[\![d]\!] \in \cup_i \mathcal{E}_i, \sigma \in \mathit{sol}(d) \text{ is constructor and irreducible and } \mathcal{R} \not\models D\sigma \right\}$$

Note that such a clause exists since we have proved that $\mathcal{R} \not\models_{\mathcal{I}nd} \mathcal{E}_0$. Let $C\,[\![c]\!]$ be a clause of $\cup_i \mathcal{E}_i$ minimal by subsumption ordering and $\theta \in \mathit{sol}(c)$, irreducible and constructor ground substitution, be such that $C\theta = D_0$.

We show that whatever inference, other than Disproof, is applied to $C\,[\![c]\!]$, a contradiction is obtained, hence that the above derivation is not successful.

Inductive Narrowing. Suppose that the inference Inductive Narrowing is applied to $C\,[\![c]\!]$. By hypothesis, C has been decorated, *i.e.* $c = d \wedge x_1{:}\llcorner u_1 \lrcorner \wedge \ldots \wedge x_n{:}\llcorner u_n \lrcorner$ with $\{x_1, \ldots, x_n\} = \mathit{var}(C)$ and for all $i \in [1..n]$, $\llcorner u_i \lrcorner \in Q_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$. Hence, since $\theta \in \mathit{sol}(c)$, there exists σ and τ such that $\theta = \sigma\tau$ and $C\,[\![c]\!] \vdash^+ C\sigma\,[\![c']\!]$.

$C\,[\![c]\!]\sigma$ cannot be a tautology and c cannot be unsatisfiable and therefore the rule Inductive Deletion cannot be applied.

Let C' be the result of the application of the rule Inductive Rewriting to $C\sigma\,[\![c']\!]$. The instances of clauses of $\mathcal{H} \cup \mathcal{E} \cup \{C\}$ used in the rewriting step are smaller than $C\theta$ wrt $\gg$, and therefore, they are inductive theorems of $\mathcal{R}$. Hence $\mathcal{R} \not\models C'\tau$. Moreover, $C\theta \gg C'\tau$ and $C' \in \cup_i \mathcal{E}_i$, which is a contradiction.

With similar arguments as above, we can show that the rule Contextual Rewriting cannot be applied to $C\sigma\,[\![c']\!]$.

Assume that the rule Rewrite Splitting is applied to $C[t]_p\sigma\,[\![c']\!]$. Let

$$\{\Gamma_1 \Rightarrow l_1 \to r_1\,[\![c_1]\!],\ \ldots\ , \Gamma_n \Rightarrow l_n \to r_n\,[\![c_n]\!]\}$$

be the non-empty subset of $\mathcal{R}_\mathcal{D}$ such that for all i in $[1..n]$, $t = l_i\sigma_i$ and

$$\mathcal{R} \models_{\mathcal{I}nd} \Gamma_1\sigma_1\,[\![c' \wedge c_1\sigma_1]\!] \vee \ldots \vee \Gamma_n\sigma_n\,[\![c' \wedge c_n\sigma_n]\!]$$

The result of the application of Rewrite Splitting is:

$$\{\Gamma_1\sigma_1 \Rightarrow C[r_1\sigma_1]_p\,[\![c' \wedge c_1\sigma_1]\!], \ldots, \Gamma_n\sigma_n \Rightarrow C[r_n\sigma_n]_p\,[\![c' \wedge c_n\sigma_n]\!]\}$$

Then there exists k such that $\mathcal{R} \models \Gamma_k\sigma_k\delta$ for some $\delta \in \mathit{Sol}(c' \wedge c_k\sigma_k)$. Let $C_k \equiv \Gamma_k\sigma_k \Rightarrow C[r_k\sigma_k]_p\,[\![c' \wedge c_k\sigma_k]\!]$, we have $\mathcal{R} \not\models C_k\delta$, since $\mathcal{R} \models \Gamma_k\sigma_k\delta$, $\mathcal{R} \models t\delta = r_k\sigma_k\delta$,

and $\mathcal{R} \not\models C\theta$. On the other hand, $C\theta \gg C_k\delta$ since $\{t\} >^{mul} \Gamma_k\sigma_k$, and $t > r_k\sigma_k$. This contradicts the minimality of $C\theta$.

Narrowing, Inductive Simplification and Simplification. These cases are similar to the previous one.

Subsumption: Since $\mathcal{R} \not\models C\theta$, $C[\![c]\!]$ cannot be subsumed by an axiom of $\mathcal{R}$. If there exists $C'[\![c']\!] \in \mathcal{H} \cup (\mathcal{E} \setminus \{C[\![c]\!]\})$ such that $C[\![c]\!] \equiv C'\delta[\![c'\delta]\!] \vee D$, then we have $\mathcal{R} \not\models C'\delta\theta$ ($\theta \in sol(c')$). Hence, $r = \emptyset$ and $\delta = \emptyset$, since $C[\![c]\!]$ is minimum in $\cup_i \mathcal{E}_i$ wrt subsumption ordering. Therefore, $C' \notin (\mathcal{E} \setminus \{C\})$. Moreover, $C' \notin \mathcal{H}$, otherwise the inference Inductive Narrowing or Narrowing could also be applied to $C[\![c]\!]$, in contradiction with previous cases. Hence, Subsumption cannot be applied to $C[\![c]\!]$. □

Since there are only two kinds of fair derivations, we obtain as a corollary:

Corollary 1 (Refutational completeness) *Assume that $\mathcal{R}_\mathcal{C}$ is terminating and that $\mathcal{R}$ is sufficiently complete. Let $\mathcal{G}_0$ be a set of unconstrained clauses and let $\mathcal{E}_0 = decorate(\mathcal{G}_0)$. If $\mathcal{R} \not\models_{\mathcal{I}nd} \mathcal{E}_0$, then all fair derivations starting from $(\mathcal{E}_0, \emptyset)$ end up with $(\bot, \mathcal{H})$.*

We can generalize the soundness result of Theorem 1 to constrained goals, providing that all the variables of these goals are decorated. Under this assumption, the hypothesis of Theorem 1 that $\mathcal{R}_\mathcal{C}$ is terminating and that $\mathcal{R}$ is sufficiently complete can be dropped.

Theorem 2 (soundness of successful derivations) *Let $\mathcal{E}_0$ be a set of decorated constrained clauses. If there exists a successful derivation $(\mathcal{E}_0, \emptyset) \vdash_\mathcal{I} (\mathcal{E}_1, \mathcal{H}_1) \vdash_\mathcal{I} \cdots$ then $\mathcal{R} \models_{\mathcal{I}nd} \mathcal{E}_0$.*

Proof. The proof is the same as for Theorem 1 except we do no need the Fact 2 since the goals of $\mathcal{E}_0$ are already decorated. Hence we do neither need the hypotheses that $\mathcal{R}_\mathcal{C}$ is terminating and that $\mathcal{R}$ is sufficiently complete which where only used for the proof of Fact 2. □

As a consequence of the above theorem, we immediately have the refutational completeness of our inference system if the goals are decorated constrained clauses.

Corollary 2 (Refutational completeness) *Let $\mathcal{E}_0$ be a set of decorated constrained clauses. If $\mathcal{R} \not\models_{\mathcal{I}nd} \mathcal{E}_0$, then all fair derivations starting from $(\mathcal{E}_0, \emptyset)$ end up with $(\bot, \mathcal{H})$.*

Example. Let us consider the specification of Section 3. We shall prove, using our inference system, that the two following constrained and decorated conjectures are inductive theorems of $\mathcal{R}$. Recall that *min* is not sufficiently complete wrt $\mathcal{R}$.

$$min(ins(x, ins(y, z))) \rightarrow min(ins(y, z))\,[\![x \succcurlyeq y \wedge x, y{:}\llcorner x \lrcorner^{Nat} \wedge z{:}\llcorner x \lrcorner^{Set}]\!] \tag{12}$$

$$min(ins(x, ins(y, z))) \rightarrow min(ins(y, z))\,[\![x \succcurlyeq y \wedge x, y{:}\llcorner x \lrcorner^{Nat} \wedge z{:}\llcorner ins(x_1, x_2) \lrcorner]\!] \tag{13}$$

Let us now prove that the conjecture (12) is an inductive theorem of $\mathcal{R}$. We start by the simplification of (12) using a Partial Splitting. We obtain:

$$min(ins(y, z)) = min(ins(y, z))\,[\![x \approx y \wedge x \succcurlyeq y \wedge x, y{:}\llcorner x \lrcorner^{Nat} \wedge z{:}\llcorner x \lrcorner^{Set}]\!] \tag{14}$$

$$min(ins(x, ins(y, z))) = min(ins(y, z))\,[\![x \not\approx y \wedge x \succcurlyeq y \wedge x, y{:}\llcorner x \lrcorner^{Nat} \wedge z{:}\llcorner x \lrcorner^{Set}]\!] \tag{15}$$

The clause (14) is a tautology. Subgoal (15) is simplified using Partial Splitting again. We obtain:

$$min(ins(y, ins(x, z))) = min(ins(y, z)) \, [\![x \succ y \wedge x \succcurlyeq y \wedge x \not\approx y \wedge x, y{:}\lfloor x \rfloor^{Nat} \wedge z{:}\lfloor x \rfloor^{Set}]\!] \quad (16)$$

$$min(ins(y, ins(x, z))) = min(ins(y, z)) \, [\![x \not\succ y \wedge x \succcurlyeq y \wedge x \not\approx y \wedge x, y{:}\lfloor x \rfloor^{Nat} \wedge z{:}\lfloor x \rfloor^{Set}]\!] \quad (17)$$

Subgoal (16) is simplified by $\mathcal{R}_{\mathcal{D}}$ into $min(ins(y, z)) = min(ins(y, z))$, a tautology. Subgoal (17) can also be deleted since the constraint $x \not\succ y, x \succcurlyeq y, x \not\approx y$ is unsatisfiable This ends the proof that (12) is an inductive theorem of $\mathcal{R}$.

The proof of (13) follows the same steps.

Our inference system can refute false conjectures. This result is a consequence of the following lemma.

Lemma 2 *let* $(\mathcal{E}_i, \mathcal{H}_i) \vdash_{\mathcal{I}} (\mathcal{E}_{i+1}, \mathcal{H}_{i+1})$ *be a derivation step. If* $\mathcal{R} \models_{\mathcal{I}nd} \mathcal{E}_i \cup \mathcal{H}_i$ *then* $\mathcal{R} \models_{\mathcal{I}nd} \mathcal{E}_{i+1} \cup \mathcal{H}_{i+1}$.

Proof. Let $C \, [\![c]\!]$ be a clause in $\mathcal{E}_i$ and $(\mathcal{E}_i \cup \{C \, [\![c]\!]\}, \mathcal{H}_i) \vdash_{\mathcal{I}} (\mathcal{E}_{i+1}, \mathcal{H}_{i+1})$ be a derivation step obtained by the application of an inference to $C \, [\![c]\!]$ and assume that $\mathcal{R} \models_{\mathcal{I}nd} \mathcal{E}_i \cup \mathcal{H}_i$. By hypothesis, the instances of clauses of $\mathcal{H} \cup \mathcal{E} \cup \{C \, [\![c]\!]\}$ which are used during rewriting steps, are valid. Hence, we can show that $\mathcal{R} \models_{\mathcal{I}nd} \mathcal{E}_{i+1} \cup \mathcal{H}_{i+1}$ by a case analysis according to the rule applied to $C \, [\![c]\!]$. □

The following lemma is also used in the proof of soundness of disproof.

Lemma 3 *If* $\mathcal{R}$ *is ground confluent and sufficiently complete then for every constructor clause* $C \, [\![c]\!]$, *if* $\mathcal{R} \models_{\mathcal{I}nd} C \, [\![c]\!]$ *then* $\mathcal{R}_{\mathcal{C}} \models_{\mathcal{I}nd} C \, [\![c]\!]$.

Proof. Let $\tau \in sol(c)$ be a substitution grounding for C. By the sufficient completeness of $\mathcal{R}$, we may assume without loss of generality that τ is a constructor substitution. By hypothesis, $\mathcal{R} \models C\tau$. Assume that for some literal $u = v$ of C, we have $\mathcal{R} \models u\tau = v\tau$. Since $\mathcal{R}$ is ground confluent, it means that $u\tau \downarrow_{\mathcal{R}} v\tau$, and hence that $u\tau \downarrow_{\mathcal{R}_{\mathcal{C}}} v\tau$, *i.e.* $\mathcal{R}_{\mathcal{C}} \models u\tau = v\tau$, because $u\tau, v\tau \in \mathcal{T}(\mathcal{C})$. Moreover, if $\mathcal{R} \models u\tau \neq v\tau$ then $\mathcal{R}_{\mathcal{C}} \models u\tau \neq v\tau$ because $\mathcal{R}_{\mathcal{C}} \subseteq \mathcal{R}$. □

Theorem 3 (Soundness of disproof) *Assume that* $\mathcal{R}$ *is strongly complete and ground confluent. If a derivation starting from* $(\mathcal{E}_0, \emptyset)$ *returns the pair* $(\bot, \mathcal{H})$, *then* $\mathcal{R} \not\models_{\mathcal{I}nd} \mathcal{E}_0$.

Proof. Under our assumptions, there exists a step k in the derivation, such that Disproof applies to a constrained clause $C \, [\![c]\!]$ in $\mathcal{E}_k$.

We prove first that $C \, [\![c]\!]$ is a constructor clause. Assume indeed that $C \, [\![c]\!]$ contains a term of the form $f(t_1, \ldots, t_n)$, where $f \in \mathcal{D}$ and for all $i \in [1..n]$, $t_i \in T(\mathcal{C}, \mathcal{X})$. The constraint c is satisfiable, otherwise Inductive Deletion could be applied. Let $\tau \in sol(c)$. Hence by Lemma 1, for each $x \in var(C)$, $x\tau$ is in $\mathcal{R}_{\mathcal{C}}$-normal form.

We have now two possibilities:

1. for one $i \in [1..n]$, $t_i\tau$ is reducible. In this case, there exists a substitution σ such that $\tau = \sigma\theta$ and $t_i \, [\![c]\!] \vdash^{+} t_i\sigma \, [\![c']\!]$ and $t_i\sigma$ contains as a subterm an instance of a left-hand side of rule of $\mathcal{R}_{\mathcal{C}}$. Therefore, either Rewriting or Partial Splitting can be applied to $t_i\sigma \, [\![c']\!]$. It implies that Narrowing can be applied to $C \, [\![c]\!]$, which is a contradiction.

2. every $t_i\tau$ is irreducible. The term $f(t_1, \ldots, t_n)\tau$ is reducible at root position because f is strongly complete wrt $\mathcal{R}$. Then there exists σ such that $\tau = \sigma\theta$ and $f(t_1, \ldots, t_n) \llbracket c \rrbracket \vdash^+ f(t_1, \ldots, t_n)\sigma \llbracket c' \rrbracket$ and moreover $f(t_1, \ldots, t_n)\sigma$ is an instance of a left-hand side of rule of $\mathcal{R}_\mathcal{D}$. Therefore, either Inductive rewriting or Rewrite Splitting can be applied. Indeed the application condition of the latter inference is a consequence of the strongly completeness of $\mathcal{R}$. Hence, the inference Inductive Narrowing can be applied to $C \llbracket c \rrbracket$, which is a contradiction.

In conclusion, the clause $C \llbracket c \rrbracket$ contains only constructor terms.

Then, we deduce that $C \llbracket c \rrbracket$ contains ground irreducible terms only, otherwise Narrowing would apply. Since Validity does not apply either, $C \llbracket c \rrbracket$ is not an inductive consequence of $\mathcal{R}_\mathcal{C}$. By lemma 3, and since $\mathcal{R}$ is ground confluent, we conclude that $C \llbracket c \rrbracket$ is not an inductive theorem of $\mathcal{R}$. As a consequence, $\mathcal{R} \not\models_{\mathcal{I}nd} \mathcal{E}_k$. Finally, by lemma 2, we deduce that $\mathcal{R} \not\models_{\mathcal{I}nd} \mathcal{E}_0$. □

5.5 Handling Non-Terminating Constructor Systems

Our procedure assume the application of $\mathcal{R}_\mathcal{C}$ and $\mathcal{R}_\mathcal{D}$ which reduces the terms wrt $>$. This is ensured when the rewrite relation induced by $\mathcal{R}_\mathcal{C}$ and $\mathcal{R}_\mathcal{D}$ is compatible with $>$, and hence that $\mathcal{R}_\mathcal{C}$ and $\mathcal{R}_\mathcal{D}$ are terminating (separately), like in the example of Section 3. Note that this is in contrast with other procedures like [24] where the termination of the whole system $\mathcal{R}$ is required.

Moreover, if $\mathcal{R}_\mathcal{C}$ is non-terminating then one may apply, a constrained completion technique [24] in order to generate an equivalent orientable theory (with ordering constraints). The theory obtained (if completion succeeds) can then be handled by our approach.

Example. Consider the following non-terminating system for sets.

$$\begin{array}{rcl} ins(x, ins(x, y)) & = & ins(x, y) \\ ins(x, ins(x', y)) & = & ins(x', ins(x, y)) \end{array}$$

Applying the completion procedure we obtained the constrained system of Section 3.

6 Decision Procedures

We present a method to decide the oracles in the inference rules of Figures 2, 3, and 4, by reduction to emptiness decision for tree automata with constraints. We assume here that, like in Theorem 1, the inference system is applied to a set $decorate(\mathcal{G}_0)$ where $\mathcal{G}_0$ is a set of unconstrained clauses.

6.1 Reductions

Consider the following decision problems, given two constrained grammars $\mathcal{G}$, $\mathcal{G}'$ and two non terminals $\llcorner u \lrcorner$, $\llcorner u' \lrcorner$ of respectively $\mathcal{G}$ and $\mathcal{G}'$,

(ED) emptiness decision: $L(\mathcal{G}, \llcorner u \lrcorner) = \emptyset$?

(EI) emptiness of intersection: $L(\mathcal{G}, \llcorner u \lrcorner) \cap L(\mathcal{G}', \llcorner u' \lrcorner) = \emptyset$?

Ground instances. Let $t\,[\![c]\!]$ be a constrained term (or clause) such that the constraint c has the form $x_1{:}\llcorner u_1 \lrcorner \wedge \ldots \wedge x_m{:}\llcorner u_m \lrcorner \wedge d$ where d contains no membership constraints. Note that starting with decorated clauses, any goal or subgoal occurring during the inference is of the above form. The set of ground instances of t satisfying c is recognized by a constrained grammar $\mathcal{G}(t\,[\![c]\!]) = \big(Q(t\,[\![c]\!]), \Delta(t\,[\![c]\!])\big)$ whose construction is described in Figure 5.
For technical reasons concerning non-terminals separation, we use in the construction of $\mathcal{G}(t\,[\![c]\!])$ a relabeling isomorphism $^\circ$ from the signature $(\mathcal{S}, \mathcal{F})$ to the signature $(\mathcal{S}^\circ, \mathcal{F}^\circ)$, such that the function symbol f° has profile $S_1^\circ \times \ldots \times S_n^\circ \to S^\circ$ if f has profile $S_1 \times \ldots \times S_n \to S$, and its extension from $T(\mathcal{F}, \mathcal{X})$ to $T(\mathcal{F}^\circ, X)$, such that (recursively) $f(t_1, \ldots, t_n) = f^\circ(t_1^\circ, \ldots, t_n^\circ)$, and for each $x \in \mathcal{X}$, $x^\circ = x$.

$$Q(t\,[\![\textstyle\bigwedge_{i=1}^m x_i{:}\llcorner u_i \lrcorner \wedge d]\!]) = Q_{\mathrm{NF}}(\mathcal{R}_\mathcal{C}) \cup \{\, \llcorner u^\circ \lrcorner \;\big|\; u \trianglelefteq t\}$$

$\Delta(t\,[\![\bigwedge_{i=1}^m x_i{:}\llcorner u_i \lrcorner \wedge d]\!])$ contains all the production rules of $\Delta_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ plus:

$$\begin{array}{rcl} \llcorner t^\circ \lrcorner & := & g(\llcorner t_1^\circ \lrcorner, \ldots, \llcorner t_m^\circ \lrcorner)\,[\![d]\!], \text{ if } t = g(t_1, \ldots, t_m) \\ \text{and every } \llcorner f^\circ(v_1^\circ, \ldots, v_n^\circ) \lrcorner & := & f(\llcorner s_1 \lrcorner, \ldots, \llcorner s_n \lrcorner)\,[\![\;]\!] \text{ such that } f(u_1, \ldots, u_n) \lhd t, \end{array}$$

and $\forall j \leq m$ if $v_j^\circ = x_i$ for some i, then $\llcorner s_j \lrcorner = \llcorner u_i \lrcorner$
if $v_j^\circ \in \mathcal{X} \setminus \{x_1, \ldots, x_m\}$ then $\llcorner s_j \lrcorner \in Q_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$
if $v_j^\circ \notin \mathcal{X}$ then $\llcorner s_j \lrcorner = \llcorner v_j^\circ \lrcorner$

Figure 5: Constrained Grammar $\mathcal{G}(t, c)$ Ground instances

Lemma 4 $L(\mathcal{G}(t\,[\![c]\!]), \llcorner t \lrcorner) = \{t\sigma \;|\; \sigma|_{var(c)} \in sol(c)\}$.

Proof. The proofs of both directions $\subseteq$ are straightforward inductions resp. on the length of a derivation of a term of $L(\mathcal{G}(t\,[\![c]\!]), \llcorner t \lrcorner)$ and on a ground instance $t\sigma$ such that $\sigma|_{var(c)}$ is a solution of c. □

Constraints unsatisfiability. This property is required for rules Inductive Rewriting, Contextual Rewriting, Rewrite Splitting, Inductive Deletion, Deletion, and Subsumption.

Lemma 5 *Given a constraint c, there exists a constrained grammar $\mathcal{G}(c)$ such that c is unsatisfiable iff $L\big(\mathcal{G}(c)\big) = \emptyset$.*

Proof. Let $x_1, \ldots, x_m$ be the list of all the variables occurring in c, eventually with repetition in case of multiple occurrences. Let $y_1, \ldots, y_m$ be a list of fresh distinct variables, let f^m be a new function symbol of arity m and let $\tilde{c} = \bigwedge_{i=1}^m y_i \approx x_i$. The constrained grammar $\mathcal{G}(c)$ is defined by $\mathcal{G}(c) = \mathcal{G}\big(f^m(y_1, \ldots, y_m)\,[\![c \wedge \tilde{c}]\!]\big)$. □

Corollary 3 *Constraints unsatisfiability is reducible to* (ED).

Ground irreducibility. The rules Validity and Disproof check this property.

Lemma 6 *Ground irreducibility decision is reducible to* (EI).

Proof. By definition and Lemmas 1 and 4, a constrained clause $C\,[\![c]\!]$ is ground irreducible iff $L\big(\mathcal{G}(C\,[\![c]\!])\big) \cap L\big(\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C}), Q_{\mathrm{NF}}(\mathcal{R}_\mathcal{C}) \setminus \{\llcorner x^{Red} \lrcorner\}\big) = \emptyset$. □

Validity of ground irreducible constructor clauses. The rule Validity checks this property.

Lemma 7 *When $\mathcal{R}_\mathcal{C}$ is confluent, validity of ground irreducible constructor constrained clauses is reducible to* (ED).

Proof. Let $C\,[\![c]\!]$ be a ground ground irreducible constructor constrained clause. Let $\tilde{C}$ be the constraint obtained from C by replacement of every equation $s = t$ (resp. disequation $s \neq t$) by the atom $s \approx t$ (resp. $s \not\approx t$). Since $C\,[\![c]\!]$ is ground irreducible and $\mathcal{R}_\mathcal{C}$ is ground confluent, we have that $C\,[\![c]\!]$ is valid in the initial model of $\mathcal{R}$ iff there exists no substitution $\sigma \in sol(c)$ grounding for C such that $\mathcal{R}_\mathcal{C} \not\models C\sigma$ iff σ there exists no σ grounding for C such that $\sigma \in sol(c)$ and $\sigma \notin sol(\tilde{C})$ iff $L\big(\mathcal{G}(C\,[\![c \wedge \neg\tilde{C}]\!])\big) = \emptyset$. □

6.2 Tree Automata with (dis)Equality Constraints and Decision

It remains to give decision procedures for (ED) and (EI).

Definition 6 *A constrained grammar $\mathcal{G}$ is called* normalized *if for each of its productions $\llcorner t \lrcorner := f(\llcorner u_1 \lrcorner, \ldots, \llcorner u_n \lrcorner)\,[\![c]\!]$ all the atomic constraints in c have the form $P(s_1, \ldots, s_k)$ where $P \in \mathcal{L}$ and $s_1, \ldots, s_k$ are strict subterms of $f(u_1, \ldots, u_n)$.*

It is obvious that every normalized constrained grammar which contains only constraints with $\approx$, $\not\approx$ in its production rules is equivalent to a tree automaton with equality and disequality constraints (AWEDC), see [13] for a survey. Therefore, constrained grammars inherit the properties of AWEDC concerning emptiness decision, and (ED), (EI) are decidable for a normalized constrained grammar when for each production $\llcorner t \lrcorner := f(\llcorner u_1 \lrcorner, \ldots, \llcorner u_n \lrcorner)\,[\![c]\!]$:

1. the constraints in c have the form $u_i \approx u_j$ or $u_i \not\approx u_j$ [4],
2. the constraints in c are only disequalities $s_1 \not\approx s_2$ [15],
3. the constraints in c are equalities and disequalities, and for every (ground) constrained term $t\,[\![c]\!]$ generated by $\mathcal{G}$, for every path $p \in \mathcal{P}os(t)$, the number of subterms s occurring along p in t and such that $s \approx s'$ or $s' \approx s$ is an atomic constraint of c is bounded (independently from t and c) [16],
4. the constraints in c are equalities and disequalities, and for every (ground) constrained term $t\,[\![c]\!]$ generated by $\mathcal{G}$, for every path $p \in \mathcal{P}os(t)$, the number of subterms s satisfying the following conditions (i–iii) is bounded (independently from t and c) [10]
   (i) s occurs along p in t,
   (ii) $s \approx s'$ or $s' \approx s$ is an atomic constraint of c,
   (iii) s, s' are not brothers in a subterm $f(\ldots, s, \ldots, s', \ldots)$ occurring on p.

Theorem 4 *Assume that $\mathcal{R}_\mathcal{C}$ is confluent and such that, for all $l \to r\,[\![c]\!] \in \mathcal{R}_\mathcal{C}$, for all $s \approx s' \in c$, each of s and s' is either a variable or a strict subterm of l, and for all $s \not\approx s' \in c$, there is a subterm of l of the form $g(\ldots, s, \ldots, s', \ldots)$. All the conditions of the simplification rules in Figures 2,3 and the inference rules in Figure 4 are decidable or make recursive call to the procedure itself.*

Proof. When the constraints of $\mathcal{R}_\mathcal{C}$ fulfill the above conditions, then $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ is in category 4, hence (ED) and (EI) are decidable. Hence the conditions in the inference and simplification rules in Figures 2,3,4 which are not recursive call, are decidable by Corollary 3 and Lemmas 6,7. □

The algorithms provided in the literature for the emptiness decision for the classes 1 to 4 of tree automata with equality and disequality constraints are all very costly, due to the inherent complexity of the problem. For instance, for the "easiest" class 1, the problem is EXPTIME-complete [13], see also [23, 15] concerning class 2. The problem is however less difficult for deterministic automata (*e.g.*, PTIME for class 1), like the one of Figure 1. The best solution for optimizing our method is certainly a cleaning algorithm, in the lines of [10], which may behave well in the average. An interesting aspect of the cleaning algorithm is its monotonicity: an incremental change on the automaton in input causes only an incremental change of the intermediate structure constructed by the algorithm for emptiness decision. This should permit to reuse such structures in our setting because all the constrained grammars of Section 6.1 are incrementally obtained from the unique normal form grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$.

7 Conclusion

A fundamental issue in automatic theorem proving by induction is the computation of a suitable finite description of the set of $\mathcal{R}$-irreducible ground terms: the induction scheme. Normal form constrained tree grammars are perfect induction schemes in the sense that they generate *exactly* the set of normal forms. At the opposite, test sets and cover sets are approximative induction schemes when the constructors are not free. They may indeed also represent some reducible ground terms, and therefore shall cause the failure (a result of the form "don't know") of an induction proof when constructors are non-free. This explains our choice of constrained grammars for the generation of subgoals. Constrained grammars are also used (by means of emptiness test) to refute false constructor conjectures provided that the axioms for constructors are ground confluent.

Our inference system allows rewrite rules between constructors which can be constrained. This permits to automate induction proofs on complex data structures. This inference system is sound and refutationally complete, and allows for the refutation of false conjectures, even with constrained constructor rules. Moreover, all the conditions of inference rules are either recursive calls to the procedure (Rewrite Splitting or Contextual Rewriting), or either decidable tests, under some assumptions on the constraints of the rewrite system for constructors. These assumptions are concerned with emptiness decidability for constrained grammars.

Ordering constraints may also help in some cases to orient non-terminating constructor axioms, using ordering completion [24]. This could be the case, for instance, of associativity and commutativity axioms.

8 Further Works

Tree automata with ordering constraints. The biggest theoretical task related with this aork will be the development of emptiness decision procedures for tree automata with constraints. Some theoretical results still need to be investigated, for other constraints than equality or disequality, in particular for ordering constraints. For instance, results of decidability of the problems (ED) and (EI) for tree automata with ordering constraints interpreted

with a total ordering $\prec$ on ground terms could (once transposed to *normalized* constrained grammars) permit to apply our procedure for induction proofs in the axiomatization of sorted lists.

Indeed, we could deal with such data structures with axioms for constructors in $\mathcal{R}_{\mathcal{C}}$ which are linear rewrite rules or constrained rules of the form $cons(x, cons(y, z)) \rightarrow cons(x, z) \, [\![x \succeq y]\!]$. It gives normalized constrained grammars of a very restricted kind. More precisely, in this case, all the constrained production rules have the form:

$$\llcorner cons(u_1, u_2) \lrcorner := cons(\llcorner v \lrcorner, \llcorner cons(u_1, u_2) \lrcorner) \, [\![v \prec u_1]\!]$$

where $\llcorner cons(u_1, u_2) \lrcorner$ is the same non terminal in every constrained rule.

We believe that in the above special case, the decidability of (ED) and (EI) can be established with some pumping lemma. The general case of strict ordering constraints is lots more complicated and if (ED) is decidable in this case, the proof, out of the scope of this paper, should need combinatoric arguments like in [15]. Note that the problem (ED) is undecidable for constrained grammars with ordering constraints $\prec$ or $\preceq$ interpreted by a total ordering on ground terms. This is a consequence of the proof in [14] of undecidability of ground reducibility for ordered rewriting.

Tree automata with other constraints. It would be also interesting to study classes of tree automata with more unusual constraints, for instance constraints of the form $length(x) = length(y)$ useful for the axiomatization of powerlists, see [21]. Whether it is possible to get rid of the ground confluence for the constructor rewrite system (for the decision of validity of ground irreducible constructor clauses) with similar technique is an important issue as well.

Membership equational logic. We plan also to generalize our approach to membership equational theories [7].

Verification of key agreement protocols. We are also working on applying our method to the verification of authenticated group key agreement protocols, like the extensions of the Diffie-Hellman scheme to a group of participants studied in [28]. We plan to follow in particular the approach of [30] for the search of security attacks.

Automatic generation of lemmas. Of course our procedure can diverge (it is well known that the problem undecidable). However, in this case we can guess some lemmas which can be used to subsume or simplify the generated infinite family of subgoals which allow us to stop the divergence. This problem is addressed in [31]. An interesting further work is the use of constrained tree grammars for the generation of appropriate lemmas necessary to avoid the divergence of the proof.

Implementation. We are planning to implement the procedure presented on this paper, based on one hand on the core of the Spike system, and on the other hand, on an efficient library for tree automata with constraints.

Acknowledgments

We thank Michael Rusinowitch for many fruitful discussions, Laurent Fribourg for his comments on a preliminary version of this paper, and the anonymous referees whose many relevant comments helped us to improve the paper.

References

[1] L. Bachmair. Proof by consistency in equational theories. In *Proc. 3rd IEEE Symposium on Logic in Computer Science*, 1988.

[2] G. Barthe and S. Stratulat. Validation of the JavaCard Platform with Implicit Induction Techniques. In *Proc. of the 14th Conference on Rewriting Techniques and Applications*, vol. 2706 of Springer Lecture Notes in Computer Science, pages 337-351, 2003.

[3] N. Berregeb, A. Bouhoula, and M. Rusinowitch. Automated verification by induction with associative-commutative operator. In *Proc. of the 8th International Conference on Computer-Aided Verification*, Springer Lecture Notes in Computer Science, 1995.

[4] B. Bogaert and S. Tison. Equality and disequality constraints on brother terms in tree automata. In *Proc. of the 9th Symp. on Theoretical Aspects of Computer Science*, 1992.

[5] A. Bouhoula. Automated theorem proving by test set induction. *Journal of Symbolic Computation*, 23(1):47–77, 1997.

[6] A. Bouhoula and J.-P. Jouannaud. Automata-driven automated induction. *Information and Computation*, 169(1):1–22, 2001.

[7] A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and Proof in Membership Equational Logic. *Theoretical Computer Science*, 236(1-2) : 35-132, 2000.

[8] A. Bouhoula and M. Rusinowitch. Implicit induction in conditional theories. *Journal of Automated Reasoning*, 14(2):189–235, 1995.

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

[10] A.C. Caron, H. Comon, J.L. Coquidé, M. Dauchet, and F. Jacquemard. Pumping, cleaning and symbolic constraints solving. In *Proc. of the 21st ICALP Conference*, 1994.

[11] H. Comon. *Unification et disunification. Théories et applications*. Thèse de Doctorat d'Université, Institut Polytechnique de Grenoble (France), 1988.

[12] H. Comon. Inductionless Induction. *Handbook of Automated Reasoning*, chapter 14, Elsevier, 2001.

[13] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. `http://www.grappa.univ-lille3.fr/tata`, 2002.

[14] H. Comon, P. Narendran, R. Nieuwenhuis, and M. Rusinowitch. Decision problems in ordered rewriting. In *Proc. 13th IEEE Symposium on Logic in Computer Science*, 1998.

[15] H. Comon and F. Jacquemard. Ground reducibility is EXPTIME-complete. *Information and Computation*, 187(1):123–153, 2003.

[16] M. Dauchet, A.-C. Caron, and J.-L. Coquidé. Automata for reduction properties solving. *Journal of Symbolic Computation*, 20, 1995.

[17] N. Dershowitz and J.-P. Jouannaud. *Handbook of Theoretical Computer Science*, volume B, chapter 6: Rewrite Systems, pages 244–320. Elsevier Science Publishers B. V. (North-Holland), 1990.

[18] L. Fribourg. A Strong Restriction of the Inductive Completion Procedure. *Journal of Symbolic Computation*, 8(3): 253-276, 1989.

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

[20] J.-P. Jouannaud and E. Kounalis. Proof by induction in equational theories without constructors. In *Proc. 1st IEEE Symposium on Logic in Computer Science*, 1986.

[21] D. Kapur. Constructors can be Partial Too, *Essays in Honor of Larry Wos*, MIT Press, 1997.

[22] D. Kapur and D. R. Musser. Proof by Consistency. The *Artificial Intelligence Journal*, 31:125–157, 1987.

[23] D. Kapur, P. Narendran, D. Rosenkrantz, and H. Zhang. Sufficient completeness, ground reducibility and their complexity. *Acta Informatica*, 28:311–350, 1991.

[24] C. Kirchner, H. Kirchner, and M. Rusinowitch. Deduction with symbolic constraints. *Revue d'Intelligence Artificielle*, 4(3):9–52, 1990. Special issue on Automatic Deduction.

[25] F. Klay, M. Rusinowitch, and S. Stratulat. Mechanical Verification of an Ideal Incremental ABR Conformance Algorithm. *Journal of Automated Reasoning*, 30(2): 53-177, 2003.

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

[27] S. Owre, J. Rushby, N. Shankar, and F. von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, 1995.

[28] O. Pereira, J.-J. Quisquater. Some attacks upon authenticated group key agreement protocols *Journal of Computer Security archive* 11(4):555 - 580, 2004.

[29] C. Sengler. Termination of Algorithms over Non-freely Generated Data Types. In proceedings of the 13th International Conference on Automated Deduction, vol. 1104 of Springer Lecture Notes in Computer Science, pages 121-135, 1996.

[30] G. Steel, A. Bundy and E. Denny. Finding counterexamples to Inductive conjectures and discovering security protocol attacks. Presented at a joint session of the verify'02 workshop and the foundations of computer security workshop, part of Floc'02, on July 25th 2002.

[31] T. Walsh. A divergence Critic for Inductive Proof. *Journal of Artificial Intelligence Research* 4:209-235, 1996.

[32] H. Zhang. Implementing contextual rewriting. In *Proc. 3rd International Workshop on Conditional Term Rewriting Systems*, 1992.

[33] H. Zhang, D. Kapur, and M. S. Krishnamoorthy. A mechanizable induction principle for equational specifications. In *Proc. 9th International Conference on Automated Deduction*, vol. 310 of Springer Lecture Notes in Computer Science, pages 162–181, 1988.