Constrained Tree Grammars to Pilot Automated Proof by Induction[*]

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

Florent Jacquemard
INRIA Futurs and
LSV / CNRS UMR 8643
61 avenue du Président Wilson
94235 Cachan Cedex, FRANCE
florent.jacquemard@lsv.ens-cachan.fr

June 14, 2004

Abstract

In this paper, we develop a new approach for mechanizing induction on complex data structures (like bags, sorted lists, trees, powerlists...) by adapting and generalizing works in tree automata with constraints. The key idea of our approach is to compute a tree grammar with constraints which describes the initial model of the given specification. This grammar is used as an *induction schema* for the generation of subgoals during the proof. Our procedure is sound and refutationally complete even when the axioms for constructors are not left-linear, constrained, non-terminating. Moreover, it subsumes all test set induction approaches. Based on several examples, our method seems to yield very natural proofs.

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 schemas. In this paper, we develop a new approach for solving this problem by adapting and generalizing works on test set induction on one hand and tree automata with constraints on the other hand.

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, 29, 25] and proof by consistency [24, 22, 19, 20, 1, 15, 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, 4]. As a result, it requires less interaction than other provers, as exemplified by the proof with the SPIKE system of the Gilbreath card trick [8], the Ramsey's theorem, and the soundness

[*]Extended abstract. The full version of this paper is available in [5].

of many digital circuits [2]... This method as well as the two others usually require that the specification is built with the constructor discipline. Moreover, the constructors are 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, 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.

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 convergent 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, 17, 16] and proof by consistency methods [13, 15, 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 constraints classes and restriction is the central problem of many theoretical studies on tree automata with constraints [3, 10, 17, 16].

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 4, see also the example in Section 2. We start with a conjecture (goal) T 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) normalize each obtained instance by the axioms and the induction hypotheses,

   C becomes then an *induction hypothesis*,

```
3. for each subgoal $S$, **if** $S$ is a ground irreducible constructor clause
                         **then if** $S$ is valid in the initial model
                               **then** delete $S$
                               **else** failure ($S$ is a counterexample)
                         **else** $S$ becomes a new subgoal, go to 2.
```

If every subgoal is deleted, then T 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.* [16]) to constrained rewrite rules – in Section 3. Both tests of ground irreducibility and validity in the initial model are performed by mean of a non-emptiness test on a constrained grammar obtained from S and $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ (Section 5).

Using tree automata (or grammars) with constraints in the setting of automated induction is not a new idea. 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 schema and decision procedure for consistency check, as long as emptiness is decidable. This make precise 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 favour of our method are: (i) it subsumes all test set induction procedures [8, 4, 6], (ii) it is sound even when the axioms for defined functions are conditional, constrained and terminating and the axioms for constructors are constrained, non-left-linear and terminating, (iii) it allows the refutation of false conjectures provided that the axioms for constructors are ground convergent and the axioms for defined functions are terminating and completely defined over the constructors, (iv) it is refutationally complete in the following sense: any conjecture that is not valid in the initial model will be disproved, under the same assumption as in (ii), (v) in the case where the axioms for constructors are not terminating, it is possible to generate an equivalent orientable theory (with ordering constraints) by applying constrained completion technique [23], see Section 6. This new theory can then be handled by our approach. This allows in particular to make proofs modulo non orientable axioms, without having to modify the core of our procedure, (vi) it does not assume termination of the whole set of rules, but separately of the rules for defined function and of the rules for constructors. Most of the other methods allowing for non-free constructors assume termination of all rules together.

1 Preliminaries

We assume that the reader is familiar with the basic concepts of term rewriting [18] 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. Let $\mathcal{X}$ be a family of sorted variables. We write variables with sort exponent like x^S which has sort $S \in \mathcal{S}$. The set of well-sorted terms will be denoted by $\mathcal{T}(\mathcal{F}, \mathcal{X})$. 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. Λ is the empty string (root position). The *subterm* of t at position p is denoted by $t|_p$, and we write the (strict) subterm ordering $t \rhd t|_p$ (if p is not the root position). 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 use $\mathcal{V}ar(t)$ for the set of variables of t. By $\mathcal{T}(\mathcal{F})$ we denote the set of variable-free or *ground* terms. We assume that each sort contains a ground term. A term t is *linear* if every variable in $\mathcal{V}ar(t)$ occurs exactly once in t. A variable renaming is a (well sorted) 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. We write $\text{mgu}(s, t)$ 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 terms of $\mathcal{T}(\mathcal{F})$. Typically, $\mathcal{L}$ contains the syntactic equality $. \approx .$ (syntactic disequality $. \not\approx .$) and some (recursive) simplification ordering $. \prec .$ on ground 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 $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. An empty combination is interpreted to true.

We may extend the application of substitutions from terms to constraints in a straithforward way, and therefore define a solution for a constraint c as a 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 $\mathcal{S}ol(c)$. A constraint c is *satisfiable* if $\mathcal{S}ol(c) \neq \emptyset$ (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.

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,
- c is a constraint.

A rule with an empty condition is called a constrained rewrite rule. A set of conditional constrained, resp. constrained, rules is called a *conditional constrained* (resp. *constrained*) *rewrite system*.

Definition 1 *Let $\mathcal{R}$ be a conditional constrained rewrite system. A term $t[l\sigma]\,[\![d]\!]$ rewrites to $t[r\sigma]\,[\![d]\!]$ by a rule $\Gamma \Rightarrow l \to r\,[\![c]\!] \in \mathcal{R}$ if $\sigma \in \mathcal{S}ol(c)$ is such that $u\sigma \downarrow_{\mathcal{R}} v\sigma$ 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 w.r.t. to 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 $t\gamma$ is reducible (resp. irreducible) for every irreducible solution γ 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.

Constructor specifications. We assume that the signature comes in two parts, $\mathcal{F} = \mathcal{C} \cup \mathcal{D}$ where $\mathcal{C}$ a set of constructor symbols, and $\mathcal{D}$ is a set of defined symbols. We note $\mathcal{R} = \mathcal{R}_{\mathcal{C}} \uplus \mathcal{R}_{\mathcal{D}}$, where $\mathcal{R}_{\mathcal{C}}$ contains only function symbols from $\mathcal{C}$. An operator $f \in \mathcal{D}$ is completely defined w.r.t. $\mathcal{C}$ 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 $\mathcal{R}$ on $\mathcal{F}$ is sufficiently complete iff each defined operator $f \in \mathcal{D}$ is completely defined.

Inductive theorems. A formula φ is a *deductive theorem* of $\mathcal{R}$ (denoted $\mathcal{R} \models \varphi$) if it is valid in any model of $\mathcal{R}$. A formula φ is an *inductive theorem* of $\mathcal{R}$ (denoted $\mathcal{R} \models_{\mathcal{I}nd} \varphi$) if it is valid in the initial model of $\mathcal{R}$, or equivalently, for all ground instance $\varphi\sigma$ of φ, we have $\mathcal{R} \models \varphi\sigma$. Given a constrained clause $C\,[\![c]\!]$, we write $\mathcal{R} \models_{\mathcal{I}nd} C\,[\![c]\!]$ iff $\mathcal{R} \models_{\mathcal{I}nd} c \Rightarrow C$.

2 Example

Consider the following signature: $\mathcal{S} = \{\mathrm{Bool}, \mathrm{Nat}, \mathrm{Set}\}$,
$\mathcal{C} = \{\mathit{true} : \mathrm{Bool}, \mathit{false} : \mathrm{Bool}, 0 : \mathrm{Nat}, s : \mathrm{Nat} \to \mathrm{Nat}, \infty : \mathrm{Nat}, \emptyset : \mathrm{Set}, \mathit{ins} : \mathrm{Nat} \times \mathrm{Set} \to \mathrm{Set}\}$
and a constructor rewrite system for ordered lists,
$\mathcal{R}_{\mathcal{C}} = \big\{\mathit{ins}(x, \mathit{ins}(y, z)) \to \mathit{ins}(x, z)\,[\![x \approx y]\!],\quad \mathit{ins}(x, \mathit{ins}(y, z)) \to \mathit{ins}(y, \mathit{ins}(x, z))\,[\![x \succ y]\!]\big\}$.

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

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

$$\text{and let } \mathcal{R}_{\mathcal{D}} = \left\{ \begin{array}{c} x \in \emptyset \to \mathit{false}, \quad x \in \mathit{ins}(y, z) \to \mathit{true}\,[\![x \approx y]\!], \quad x \in \mathit{ins}(y, z) \to x \in z\,[\![x \not\approx y]\!], \\ \mathit{sorted}(\emptyset) \to \mathit{true}, \quad \mathit{sorted}(\mathit{ins}(x, \emptyset)) \to \mathit{true}, \\ \mathit{sorted}(\mathit{ins}(y, z)) = \mathit{true} \;\Rightarrow\; \mathit{sorted}(\mathit{ins}(x, \mathit{ins}(y, z))) \to \mathit{true}\,[\![x \prec y]\!], \\ \mathit{sorted}(\mathit{ins}(x, y)) = \mathit{true} \;\Rightarrow\; \mathit{min}(\mathit{ins}(x, y)) \to x \end{array} \right\}$$

Now we want to prove by induction the following conjecture:

$$z = \mathit{min}(y) \Rightarrow x \in y = \mathit{false}\,[\![x \prec z]\!] \qquad (1)$$

2.1 Test set induction

Roughly, the principle of a proof by test set induction [8, 4] presented in Introduction (with steps 1. 2. 3.) except that the induction schema is a test set, *i.e.* a set of constructor terms, and the goals are instantiated by mean of variable instantiation by terms of the test set.

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

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

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

$$z = \mathit{min}(\emptyset) \Rightarrow x \in \emptyset = \mathit{false} \qquad [\![x \prec z]\!] \tag{2}$$

$$z = \mathit{min}(\mathit{ins}(x_1, \emptyset)) \Rightarrow x \in \mathit{ins}(x_1, \emptyset) = \mathit{false} \qquad [\![x \prec z]\!] \tag{3}$$

$$z = \mathit{min}(\mathit{ins}(x_2, \mathit{ins}(x_3, x_4))) \Rightarrow x \in \mathit{ins}(x_2, \mathit{ins}(x_3, x_4)) = \mathit{false}\, [\![x \prec z \wedge x_2 \prec x_3]\!] \tag{4}$$

The subgoals (2) and (3) can be easily proved using test set induction. Let us prove subgoal (4). A case splitting based on the rules of $\mathcal{R}_\mathcal{D}$ (see Section 4.1 for details) gives us the two subgoals:

$$z = \mathit{min}(\mathit{ins}(x_2, \mathit{ins}(x_3, x_4))) \Rightarrow \mathit{true} = \mathit{false} \qquad [\![x \prec z \wedge x_2 \prec x_3 \wedge x \approx x_2]\!] \tag{5}$$

$$z = \mathit{min}(\mathit{ins}(x_2, \mathit{ins}(x_3, x_4))) \Rightarrow x \in \mathit{ins}(x_3, x_4) = \mathit{false}\, [\![x \prec z \wedge x_2 \prec x_3 \wedge x \not\approx x_2]\!] \tag{6}$$

In order to simplify the constrained clause (5) by $\mathcal{R}_\mathcal{D}$, we have to prove that the conjecture $\mathit{sorted}(\mathit{ins}(x_2, \mathit{ins}(x_3, x_4))) = \mathit{true}$ is an inductive theorem and hence that:

$$\mathit{sorted}(\mathit{ins}(x_3, x_4)) = \mathit{true} \tag{7}$$

is an inductive theorem using test set induction technique. If, in (7), we replace x_4 by the terms from the test set $\mathcal{TS}(\mathit{Set})$, we obtain:

$$\mathit{sorted}(\mathit{ins}(x_3, \emptyset)) = \mathit{true} \tag{8}$$

$$\mathit{sorted}(\mathit{ins}(x_3, \mathit{ins}(x'_1, \emptyset))) = \mathit{true} \tag{9}$$

$$\mathit{sorted}(\mathit{ins}(x_3, \mathit{ins}(x'_2, \mathit{ins}(x'_3, x'_4)))) = \mathit{true}\, [\![x'_2 \prec x'_3]\!] \tag{10}$$

But now, the subgoal (9), resp. (10), cannot be simplified by $\mathcal{R}_\mathcal{D}$ (by its last but one rule) because the constraint $x_3 \prec x'_1$, resp. $x_3 \prec x'_2$ is missing, and the proof stops without a conclusion. Note that such constraints cannot be added by test set instantiation of variables. Hence, we fail to prove conjectures (7) and (1) with test set induction technique.

2.2 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 3, 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:

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

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 whose non-terminals are $\{\llcorner x^{\mathrm{Bool}} \lrcorner, \llcorner x^{\mathrm{Nat}} \lrcorner, \llcorner x^{\mathrm{Set}} \lrcorner, \llcorner \mathit{ins}(x_1, x_2) \lrcorner\}$ and with production rules:

$$\begin{array}{llll} \llcorner x^{\mathrm{Bool}} \lrcorner := \mathit{true} & \llcorner x^{\mathrm{Bool}} \lrcorner := \mathit{false} & \llcorner x^{\mathrm{Nat}} \lrcorner := 0 & \llcorner x^{\mathrm{Nat}} \lrcorner := \infty \\ \llcorner x^{\mathrm{Nat}} \lrcorner := s(\llcorner x^{\mathrm{Nat}} \lrcorner) & \llcorner x^{\mathrm{Set}} \lrcorner := \emptyset & \multicolumn{2}{l}{\llcorner \mathit{ins}(x_1, x_2) \lrcorner := \mathit{ins}(\llcorner x^{\mathrm{Nat}} \lrcorner, \llcorner x^{\mathrm{Set}} \lrcorner)} \end{array}$$

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:

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

Let us first "decorate" (1) by attaching some non-terminals to the variable y:

$$z = \mathit{min}(\llcorner x^{\mathrm{Set}} \lrcorner) \Rightarrow x \in \llcorner x^{\mathrm{Set}} \lrcorner = \mathit{false} \qquad [\![x \prec z]\!] \tag{11}$$

$$z = \mathit{min}(\llcorner \mathit{ins}(x_1, x_2) \lrcorner) \Rightarrow x \in \llcorner \mathit{ins}(x_1, x_2) \lrcorner = \mathit{false} \, [\![x \prec z]\!] \tag{12}$$

This notation actually differs from the formal one of Section 3, for sake of simplicity. Then, applying the production rules to (11) gives:
$z = \mathit{min}(\emptyset) \Rightarrow x \in \emptyset = \mathit{false} \, [\![x \prec z]\!] \xrightarrow[\mathcal{R}_\mathcal{D}]{} z = \mathit{min}(\emptyset) \Rightarrow \mathit{false} = \mathit{false} \, [\![x \prec z]\!]$ and we are done. From (12) we obtain (among two subgoals) the following analogous of (4):

$$z = \mathit{min}(\mathit{ins}(\llcorner x^{\mathrm{Nat}} \lrcorner, \llcorner \mathit{ins}(x_1, x_2) \lrcorner)) \Rightarrow x \in \mathit{ins}(\llcorner x^{\mathrm{Nat}} \lrcorner, \llcorner \mathit{ins}(x_1, x_2) \lrcorner) = \mathit{false} \, [\![x \prec z \wedge x^{\mathrm{Nat}} \prec x_1]\!] \tag{4'}$$

After the case splitting, we have to prove the conjecture:

$$\mathit{sorted}(\llcorner \mathit{ins}(x_1, x_2) \lrcorner) = \mathit{true} \tag{13}$$

From (13), grammar derivations using production rules (denoted $\vdash$) give:

$$\mathit{sorted}\big(\mathit{ins}(\llcorner x^{\mathrm{Nat}} \lrcorner, \llcorner x^{\mathrm{Set}} \lrcorner)\big) = \mathit{true} \vdash \mathit{sorted}\big(\mathit{ins}(\llcorner x^{\mathrm{Nat}} \lrcorner, \emptyset)\big) = \mathit{true} \tag{8'}$$

$$\mathit{sorted}\big(\mathit{ins}(\llcorner x^{\mathrm{Nat}} \lrcorner, \llcorner \mathit{ins}(x_1, x_2) \lrcorner)\big) = \mathit{true} \, [\![x^{\mathrm{Nat}} \prec x_1]\!] \tag{14}$$

$$(14) \vdash \mathit{sorted}\big(\mathit{ins}(\llcorner x^{\mathrm{Nat}} \lrcorner, \mathit{ins}(\llcorner x_2^{\mathrm{Nat}} \lrcorner, \llcorner x^{\mathrm{Set}} \lrcorner))\big) = \mathit{true} \, [\![x^{\mathrm{Nat}} \prec x_2^{\mathrm{Nat}}]\!] \tag{9'}$$

$$(14) \vdash \mathit{sorted}\big(\mathit{ins}(\llcorner x^{\mathrm{Nat}} \lrcorner, \mathit{ins}(\llcorner x_2^{\mathrm{Nat}} \lrcorner, \llcorner \mathit{ins}(x_1, x_2) \lrcorner))\big) = \mathit{true} \, [\![x^{\mathrm{Nat}} \prec x_2^{\mathrm{Nat}} \wedge x_2^{\mathrm{Nat}} \prec x_1]\!] \tag{10'}$$

Now we have the constraints that were missing in (9) and (10) and the subgoals can be simplified by $\mathcal{R}_\mathcal{D}$.

The inference procedure presented in Section 4 proves that (1) is an inductive theorem w.r.t. $\mathcal{R}$, with no need of additional lemmas. The whole proof of (1) may be found in the long version of this paper [5]. Note that this proof is fully automatic (does not require any user interaction to add additional lemmas).

3 Constrained Grammars

Constrained tree grammars were first introduced in [11], also in the context of automated induction. Their role is twofold in our approach: on one hand, a grammar generates subgoals, by replacing some variables by terms, following production rules, therefore, they can be seen as induction schemas; 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 2 *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.

3.1 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). 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:

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

if there exists $\llcorner t \lrcorner := f(\llcorner u_1 \lrcorner, \ldots, \llcorner u_n \lrcorner)\, [\![c]\!] \in \Delta$ and $y_1, \ldots, y_n$ are fresh variables, such that $f(u_1, \ldots, u_n) = t\tau$, 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 3 *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 $y\, [\![y : \llcorner u \lrcorner]\!] \vdash^* t\, [\![c]\!]$ and 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 $L(\mathcal{G}) = L(\mathcal{G}, Q)$.

3.2 Normal Forms

Given the set of constructor rules $\mathcal{R}_\mathcal{C}$, we can construct a constrained grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C}) = (Q_{\mathrm{NF}}(\mathcal{R}_\mathcal{C}), \Delta_{\mathrm{NF}}(\mathcal{R}_\mathcal{C}))$ which generates the set of ground $\mathcal{R}_\mathcal{C}$-normal forms. The construction of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ is given in [5], it generalises the construction of [16] (Section 3.1)a and, intuitively, 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). For the sake of the construction, we add a new sort Red to $\mathcal{S}$, (the sort of reducible terms), and hence a new variable x^{Red}, and obtain that:

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

4 Inference System

In this section, we present our inductive theorem proving procedure. We assume a given rewrite ordering $>$, that is, an ordering on terms which is monotonic with respect to contexts and substitutions [18]. In the following, the relation $\vdash$ and the atomic constraints of the form "$y\colon \llcorner u \lrcorner$" shall be understood w.r.t. $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$. Conjectures to be proved (goals) are constrained clauses $C\,[\![c]\!]$.

4.1 Simplification Rules for Defined Functions

We start describing the simplification rules that we use for defined symbols, collected in Figure 1.

Definition 4 *The complexity of a constrained clause $C\,[\![c]\!]$ is the (multiset) union of the complexities of the atoms occurring in $C\,[\![c]\!]$, where the complexity of an atom $u = v$ is equal to $\{u, v\}$. Complexities are compared in the well-founded monotonic ordering $>_c = (> \cup \rhd)_{mul}$. The fresh symbol $\bot$ will be considered minimal w.r.t. $>_c$.*

We write $>_{mul}$ to denote the multiset extension of the ordering $>$.

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 a well-founded ordering used to order constrained clauses, hence is more powerful than structural induction as used in most other methods. Contextual Rewriting can be viewed as a generalization of the corresponding rule given in [28] to handle constraints. Rewrite Splitting simplifies a constrained clause which contains a subterm matching some left member of rule of $\mathcal{R}_{\mathcal{D}}$. Each rewritten clause is valid in the context of the condition and the constraints of the rule used for simplification, provided that all cases are covered. This is true when $\mathcal{R}$ is sufficiently complete, making this check superfluous in this case.

Inductive Rewriting: $\Big(\{C\,[\![c]\!]\}, \mathcal{H}\Big) \to_{\mathcal{D}} \{C'\,[\![c]\!]\}$

if $C\,[\![c]\!] \xrightarrow[\mathcal{R}_{\mathcal{D}} \cup \Psi]{} C'\,[\![c]\!]$ where $\Psi = \{\psi \mid \psi \in \mathcal{H} \text{ and } \psi <_c C\}$

Contextual Rewriting: $\Big(\{\Upsilon \Rightarrow C[l\sigma]\,[\![c]\!]\}, \mathcal{H}\Big) \to_{\mathcal{D}} \{\Upsilon \Rightarrow C[r\sigma]\,[\![c]\!]\}$

if $\mathcal{R} \models_{\mathcal{I}nd} \Upsilon \Rightarrow \Gamma\sigma\,[\![c]\!]$ and $c \wedge \neg c'\sigma$ is unsatisfiable, where $\Gamma \Rightarrow l \to r\,[\![c']\!] \in \mathcal{R}_{\mathcal{D}}$

Rewrite Splitting: $\Big(\{C[t]_p\,[\![c]\!]\}, \mathcal{H}\Big) \to_{\mathcal{D}} \{\Gamma_i\sigma_i \Rightarrow C[r_i\sigma_i]_p\,[\![c \wedge c_i\sigma_i]\!]\}_{i\in[1..n]}$

if $\neg c_1\sigma_1 \wedge \ldots \wedge \neg c_n\sigma_n$ is unsatisfiable and $\mathcal{R} \models_{\mathcal{I}nd} \Gamma_1\sigma_1 \vee \ldots \vee \Gamma_n\sigma_n$

$$\text{where} \left\{ \begin{array}{l} \text{the } \Gamma_i\sigma_i \Rightarrow l_i\sigma_i \to r_i\sigma_i\,[\![c_i\sigma_i]\!],\ i \in [1..n],\ \text{are all the instances of} \\ \text{rules } \Gamma_i \Rightarrow l_i \to r_i\,[\![c_i]\!] \in \mathcal{R}_{\mathcal{D}} \text{ such that } l_i\sigma_i = t \end{array} \right.$$

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

Figure 1: Simplification Rules for Defined Functions

4.2 Simplification Rules for Constructors

All simplification rules for constructors are collected in Figure 2. 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 $\mathcal{R}_\mathcal{C}$. This way, the saturated application of Partial splitting and Rewriting will always lead to Deletion or to ground irreducible constructor clauses.

Rewriting: $\{C\,[\![c]\!]\} \to_\mathcal{C} \{C'\,[\![c]\!]\}$ if $C\,[\![c]\!] \xrightarrow[\mathcal{R}_\mathcal{C}]{+} C'\,[\![c]\!]$

Partial Splitting: $\{C[l\sigma]_p\,[\![c]\!]\} \to_\mathcal{C} \{C[r\sigma]_p\,[\![c \wedge c'\sigma]\!], C[l\sigma]_p\,[\![c \wedge \neg c'\sigma]\!]\}$
if $l \to r\,[\![c']\!] \in \mathcal{R}_\mathcal{C}$ and neither $c'\sigma$ nor $\neg c'\sigma$ is a subformula of c

Deletion: $\{C\,[\![c]\!]\} \to_\mathcal{C} \emptyset$ if $C\,[\![c]\!]$ is a tautology or c is unsatisfiable

Validity: $\{C\,[\![c]\!]\} \to_\mathcal{C} \emptyset$
if $C\,[\![c]\!]$ is a ground irreducible constructor clause valid in the initial model of $\mathcal{R}$

Figure 2: Simplification Rules for Constructors

4.3 Induction Inference Rules

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

Definition 5 *A term $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$ is said to be* $\mathcal{R}$-covered *(resp. weakly* $\mathcal{R}$-covered*) if for each subterm $f(s_1,\ldots,s_n)$ of t, where $f \in \mathcal{D}$ and $s_i \in \mathcal{T}(\mathcal{C},\mathcal{X})$ for each $i \leq n$, for each $p \in Pos(f(s_1,\ldots,s_n))$, if $f(s_1,\ldots,s_n)|_p \in \mathcal{X}$ then $|p| = \mathrm{depth}(\mathcal{R})$ (resp. $|p| \leq \mathrm{depth}(\mathcal{R})$) else $|p| \leq \mathrm{depth}(\mathcal{R})$.*

The notion of (weakly) $\mathcal{R}$-coveredness is extended to constrained clauses as expected.

The inference system is displayed in Figure 3. Its rules apply to pairs $(\mathcal{E},\mathcal{H})$, where $\mathcal{E}$ is the set of current conjectures (constrained clauses) and $\mathcal{H}$ is the *set* of inductive hypotheses (constrained clauses as well). Decoration adds to a conjecture $C\,[\![c]\!]$ new constraints of the form $x{:}\llcorner u \lrcorner$ for each variable x of $C\,[\![c]\!]$, where $\llcorner u \lrcorner$ is a non terminal of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ with same sort as x. Each obtained new conjecture will be considered as a new subgoal. Inductive Simplification reduces a conjecture according to the rules shown in Figure 1. Inductive Narrowing generates new subgoals by application of the production rules of the constrained grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ up to weakly $\mathcal{R}_\mathcal{D}$-covered clauses. Each obtained clause is then simplified by the rules shown in Figure 1. Narrowing proceed the same way with constructor clauses. It eliminates ground reducible terms in a clause by simplifying their instances, while deriving new conjectures considered as subgoals. Subsumption is an additional simplification rule, which cannot be used as the other rules in Figure 1.

The separation of the * Narrowing rules into the constructor and non-constructor cases permits us to relax the assumption of termination of the whole system $\mathcal{R}_\mathcal{C} \cup \mathcal{R}_\mathcal{D}$ into the separate termination of $\mathcal{R}_\mathcal{C}$ and $\mathcal{R}_\mathcal{D}$.

Decoration: $\dfrac{\Big(\mathcal{E} \cup \{C\,[\![c]\!]\}, \emptyset\Big)}{(\mathcal{E} \cup \mathcal{E}', \emptyset)}$ where $\mathcal{E}' = \{C\,[\![c \wedge x_1 \colon \llcorner u_1 \lrcorner \wedge \ldots \wedge x_n \colon \llcorner u_n \lrcorner]\!] \mid$
$\forall i \leq n \; \llcorner u_i \lrcorner \in Q_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ and $sort(u_1) = sort(x_i)\}$

Inductive Simplification: $\dfrac{\Big(\mathcal{E} \cup \{C\,[\![c]\!]\}, \mathcal{H}\Big)}{(\mathcal{E} \cup \mathcal{E}', \mathcal{H})}$ if $C\,[\![c]\!]$ is a non-constructor clause,
and $(\{C\,[\![c]\!]\}, \mathcal{E} \cup \mathcal{H}) \rightarrow_{\mathcal{D}} \mathcal{E}'$

Inductive Narrowing: $\dfrac{\Big(\mathcal{E} \cup \{C\,[\![c]\!]\}, \mathcal{H}\Big)}{\Big(\mathcal{E} \cup (\bigcup\limits_{i \in [1..n]} \mathcal{E}_i), \mathcal{H} \cup \{C\,[\![c]\!]\}\Big)}$ if $C\,[\![c]\!]$ is a non-constr. clause,
$\forall i \in [1..n] : \; (C_i\,[\![c_i]\!], \mathcal{E} \cup \mathcal{H} \cup \{C\,[\![c]\!]\}) \rightarrow_{\mathcal{D}} \mathcal{E}_i$where $\{C_1\,[\![c_1]\!], \ldots, C_n\,[\![c_n]\!]\}$ is the set of clauses s.t. $C\,[\![c]\!] \vdash^{+} C_i\,[\![c_i]\!]$ and $C_i\,[\![c_i]\!]$ is weakly $\mathcal{R}_{\mathcal{D}}$-covered

Narrowing: $\dfrac{\Big(\mathcal{E} \cup \{C\,[\![c]\!]\}, \mathcal{H}\Big)}{\Big(\mathcal{E} \cup (\bigcup\limits_{i \in [1..n]} \{C_i\,[\![c_i]\!]\}\downarrow_{\mathcal{C}}), \mathcal{H} \cup \{C\,[\![c]\!]\}\Big)}$ if $C\,[\![c]\!]$ is a constructor clause,
where $\{C_1\,[\![c_1]\!], \ldots, C_n\,[\![c_n]\!]\}$ is the set of clauses s.t. $C\,[\![c]\!] \vdash^{*} C_i\,[\![c_i]\!]$ and $\forall p \in \mathcal{P}os(C_i)$ s.t. $C_i|_p \in \mathcal{X}$, $|p| \leq \mathrm{depth}(\mathcal{R}_{\mathcal{C}})$

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

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

Figure 3: Induction Inference Rules

4.4 Soundness and Completeness

Our inference system is sound, and refutationally complete. The formal statement and proofs of these properties can be found in [5].

5 Decision Procedures

We present a method to decide the oracles in the inference rules of Figures 1, 2, and 3, by reduction to emptiness decision for tree automata with constraints.

5.1 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: $L(\mathcal{G}, \llcorner u \lrcorner) \cap L(\mathcal{G}', \llcorner u' \lrcorner) = \emptyset$?

In this section, we present some results (the proofs may be found in [5]) of reduction of the problems checked by the inference rules of Figures 1,2,3 into (ED) and (EI).

Lemma 2 *Given a constraint c, there exists a constrained grammar $\mathcal{G}(c)$ such that c is unsatisfiable iff $L\big(\mathcal{G}(c)\big) = \emptyset$.*

Corollary 1 *Constraints unsatisfiability is reducible to* (ED)*.*

Lemma 3 *Ground irreducibility decision is reducible to* (EI)*.*

Lemma 4 *When $\mathcal{R}_\mathcal{C}$ is confluent, validity of ground irreducible constructor constrained clauses is reducible to* (ED)*.*

5.2 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 [14] 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 $u_i \not\approx u_j$ [3],
2. the constraints in c are only disequalities $s_1 \not\approx s_2$ [16],
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) [17],
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 1 *Let $\mathcal{R}_\mathcal{D}$ be a terminating conditional constrained rewrite system such that all defined functions are completely defined, and let $\mathcal{R}_\mathcal{C}$ be a constrained rewrite system which is terminating and 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 $g(\ldots, s, \ldots, s', \ldots)$. Then, the inference system of Figure 3 gives a sound and refutationally complete procedure for inductive theorem proving. Moreover, all the conditions of inference rules 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 system of Figures 1,2,3 is decidable by Corollary 1 and Lemmas 3,4. The rest follows from the results of soundness of successful derivations, soundness of disproof, and refutational completeness of our procedure, see Section 4.4. □

5.3 Tree Automata with Ordering Constraints and Sorted Lists

Our method could also benefit from progresses in the study of classes of tree automata with 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 $cons(x, cons(y, z)) \to cons(x, z)\, [\![x \succeq y]\!]$. 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.

6 Non Terminating Constructor Systems

If the axioms for constructors are non-terminating then one may apply, for example, a constrained completion technique [23] in order to generate an equivalent orientable theory (with ordering constraints). The obtained theory (if completion succeeds) can then be handled by our approach. This allows in particular to make proofs modulo non orientable axioms, without having to modify the core of our procedure.

Example 1 *If we apply the completion procedure to the following system for sets: $\{ins(x, ins(x, y)) = ins(x, y),\ ins(x, ins(x', y)) = ins(x', ins(x, y))\}$, we obtain the constrained system of Section 2.*

7 Conclusion

A fundamental issue in automatic theorem proving by induction (w.r.t. some terminating rewrite system $\mathcal{R}$) is the computation of a suitable finite description of the set of $\mathcal{R}$-irreducible ground terms: the induction schema. Normal form constrained tree grammars are perfect induction schemas in the sense that they generate *exactly* the set of normal forms. At the opposite, test sets and cover-sets are approximative induction schemas. 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 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 provided that the constrained rewrite system for constructors is ground convergent, that the constrained and conditional rewrite system for defined functions is terminating, and that the whole system is sufficiently complete. Moreover, all the conditions of inference rules are either decidable or (well founded) recursive calls to the procedure assuming some more hypotheses on the rewrite system for constructors. These restrictions 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 [7]. This could be the case, for instance, of associativity / commutativity axioms.

Some theoretical results still need to be investigated, for other constraints than equality or disequality, in particular for ordering constraints but also for more unusual constraints, for instance constraints of the form $\mathit{length}(x) = \mathit{length}(y)$ useful for the axiomatization of powerlists, see [21]. Whether it is possible to get rid of the ground confluence for constructor rewrite system with similar technique is an important issue as well. We plan also to generalize our approach to membership equational theories.

We are working on applying our method to the verification of cryptographic protocols, in particular for the search of security attacks, following the approaches of [26, 27].

Acknowledgements

We thank Michael Rusinowitch for many fruitful discussions.

References

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

[2] 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.

[3] 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.

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

[5] A. Bouhoula and F. Jacquemard. Constrained Tree Grammars to Pilot Automated Proof by Induction. LSV Research Report, 2004.

[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. Inductive proofs by specification transformations. In *Proc. of the 3rd Conference on Rewriting Techniques and Applications*, vol. 355 of Springer Lecture Notes in Computer Science, 1989.

[14] 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.

[15] H. Comon and R. Nieuwenhuis. Induction = I-axiomatization + first-order consistency. *Information and Computation*, 159(1/2):151-186, 2000.

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

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

[18] 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.

[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] 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.

[24] 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.

[25] 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.

[26] L.C. Paulson. The inductive Approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[27] 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.

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

[29] 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.