Automated Induction with Constrained Tree Automata[⋆,⋆⋆]

Adel Bouhoula[1] and Florent Jacquemard[2]

[1] Higher School of Communications of Tunis (Sup'Com),
University of November 7th at Carthage, Tunisia
adel.bouhoula@supcom.rnu.tn
[2] INRIA and LSV, CNRS/ENS Cachan, France
florent.jacquemard@inria.fr

Abstract. We propose a procedure for automated implicit inductive theorem proving for equational specifications made of rewrite rules with conditions and constraints. The constraints are interpreted over constructor terms (representing data values), and may express syntactic equality, disequality, ordering and also membership in a fixed tree language. Constrained equational axioms between constructor terms are supported and can be used in order to specify complex data structures like sets, sorted lists, trees, powerlists...

Our procedure is based on tree grammars with constraints, a formalism which can describe exactly the initial model of the given specification (when it is sufficiently complete and terminating). They are used in the inductive proofs first as an induction scheme for the generation of subgoals at induction steps, second for checking validity and redundancy criteria by reduction to an emptiness problem, and third for defining and solving membership constraints.

We show that the procedure is sound and refutationally complete. It generalizes former test set induction techniques and yields natural proofs for several non-trivial examples presented in the paper, these examples are difficult (if not impossible) to specify and carry on automatically with other induction procedures.

1 Introduction

Given a specification $\mathcal{R}$ of a program or system S made of equational Horn clauses, proving a property P for S generally amounts to show the validity of P in the minimal Herbrand model of $\mathcal{R}$, also called *initial model* of $\mathcal{R}$ (inductive validity). In this perspective, it is important to have automated induction theorem proving procedures supporting a specification language expressive enough to axiomatize complex data structures like sets, sorted lists, powerlists, complete binary trees, *etc.* Moreover, it is also important to be able to automatically generate induction schemas used for inductive proofs in order to minimize user interaction. However,

[⋆] A long version of this extended abstract is available as a research report [3].

[⋆⋆] This work has been partially supported by INRIA/DGRSRT grants 06/I09 and 0804.

A. Armando, P. Baumgartner, and G. Dowek (Eds.): IJCAR 2008, LNCS 5195, pp. 539–554, 2008.

theories of complex data structures generate complex induction schemes, and the automation of inductive proofs is therefore difficult for such theories.

It is common to assume that $\mathcal{R}$ is built with *constructor function symbols* (to construct terms representing data) and *defined symbols* (representing the operations defined on constructor terms). Assuming in addition the *sufficient completeness* of $\mathcal{R}$ (every ground (variable-free) term is reducible, using the axioms of $\mathcal{R}$, to a constructor term) and the strong termination of the axioms of $\mathcal{R}$, a set of representants for the initial model of $\mathcal{R}$ (the model in which we want to proof the validity of conjectures) is the set of ground constructor terms not reducible by $\mathcal{R}_\mathcal{C}$ (the subset of equations of $\mathcal{R}$ between terms made of constructor symbols), called constructor *normal forms*.

In the case where the constructors are *free* ($\mathcal{R}_\mathcal{C} = \emptyset$), the set of constructor normal forms is simply the set of ground terms built with constructors and it is very easy to define an induction schema. This situation is therefore convenient for inductive reasoning, and many inductive theorem provers require free constructors, termination and sufficient completeness. However, it is not expressive enough to define complex data structures. With rewrite rules between constructors, the definition of induction schema is more complex, and requires a finite description of the set of constructor normal-forms. Some progress has been done *e.g.* in [4] and [5] in the direction of handling specification with non-free constructors, with severe restrictions (see related work below).

Tree automata (TA), or equivalently regular tree grammars, permit a finite representation of the set of constructor normal-forms when $\mathcal{R}_\mathcal{C}$ is a left-linear rewrite system (set of rewrite rules without multiple occurrences of variables in their left-hand-sides). Indeed, on one hand TA can do linear pattern-matching, hence they can recognize terms which are reducible by $\mathcal{R}_\mathcal{C}$, and on the other hand, the class of TA languages is closed under complementation. When the axioms of $\mathcal{R}_\mathcal{C}$ are not linear, or are constrained, some extensions of TA (or grammars) are necessary, with transitions able to check constraints on the term in input, see e.g. [8].

In this paper, we propose a framework for inductive theorem proving for theories containing constrained rewrite rules between constructor terms and conditional and constrained rewrite rules for defined functions. The key idea is a strong and natural integration of tree grammars with constraints in an implicit induction procedure, where they are used as induction schema. Very roughly, our procedure starts with the automatic computation of an induction schema, in the form of a constrained tree grammar generating constructor normal form. This grammar is used later for the generation of subgoals from a conjecture C, by instantiation of variables using the grammar's production rules, triggering induction steps during the proof. All generated subgoals are either deleted, following some criteria, or they are reduced, using axioms or induction hypotheses, or conjectures not yet proved, providing that they are smaller than the goal to be proved. Reduced subgoals become then new conjectures and C becomes an induction hypothesis. Moreover, constrained tree grammars are used as a decision procedure for checking the deletion criteria during induction steps.

Our method subsumes former test set induction procedures like [6,1,4], by reusing former theoretical works on tree automata with constraints. It is *sound* and *refutationally complete* (any conjecture that is not valid in the initial model will be disproved) when $\mathcal{R}$ is sufficiently complete and the constructor subsystem $\mathcal{R}_\mathcal{C}$ is terminating. Without the above hypotheses, it still remains sound and refutationally complete for a restricted kind of conjectures, where all the variables are constrained to belong to the language of constructor normal forms. This restriction is expressible in the specification language (see below). When the procedure fails, it implies that the conjecture is not an inductive theorem, provided that $\mathcal{R}$ is *strongly* complete (a stronger condition for sufficient completeness) and ground confluent. There is no requirement for *termination* of the whole set of rules $\mathcal{R}$, unlike [6,1], but instead only for separate termination of the respective sets of rules for defined function and for the constructors.

Moreover, if a conjecture C restricted as above is proved in a sufficiently complete specification $\mathcal{R}$ and $\mathcal{R}$ is further consistently extended into $\mathcal{R}'$ with additional axioms for specifying *partial* (non-constructor) functions, then the former proof of C remains valid in $\mathcal{R}'$, see Section 5.

The support of constraints permits in some cases to use the constrained completion technique of [16] in order to transform a non-terminating theory into a terminating one, by the addition of ordering constraints in constructor rules, see Section 4.5. It permits in particular to make proofs modulo non orientable axioms, without having to modify the core of our procedure.

We shall consider a specification of ordered lists as a running example throughout the paper. Consider first non-stuttering lists (lists which do not contain two equal successive elements) built with the constructor symbols $\emptyset$ (empty list) and *ins* (list insertion) and following this rewrite rule:

$$\mathit{ins}(x, \mathit{ins}(x, y)) \to \mathit{ins}(x, y) \qquad (\mathsf{c}_0)$$

Rewrite rules can be enriched with constraints built on predicates with a fixed interpretation on ground constructor terms. For example, using ordering constraints built with $\succ$ we can specify ordered lists by the following axiom:

$$\mathit{ins}(x_1, \mathit{ins}(x_2, y)) \to \mathit{ins}(x_2, \mathit{ins}(x_1, y))\, [\![x_1 \succ x_2]\!] \qquad (\mathsf{c}_1)$$

Another interesting example is the case of membership constraints of the form $x : L$ where L is a fixed regular tree language (containing only terms made of constructor symbols). We consider also stronger constraints which restrict constructor terms to be in normal form (i.e. not reducible by the axioms). Let us come back to the example of non-stuttering sorted lists (sorted lists without duplication), and add to the above rules the axioms below which define a membership predicate $\Subset$, using the information that lists are sorted:

$$\begin{aligned} x \Subset \emptyset &\to \mathsf{false} && (\mathsf{m}'_0)\\ x_1 \Subset \mathit{ins}(x_2, y_2) &\to \mathsf{true}\, [\![x_1 \approx x_2]\!] && (\mathsf{m}'_1)\\ x_1 \Subset y_1 &\to \mathsf{false}\, [\![y_1 \approx \mathit{ins}(x_2, y_2), x_1 \prec x_2, y_1{:}\mathsf{NF}]\!] && (\mathsf{m}'_2)\\ x_1 \Subset \mathit{ins}(x_2, y_2) &\to x_1 \Subset y_2\, [\![x_2 \prec x_1]\!] && (\mathsf{m}'_3) \end{aligned}$$

The constraint y_1:NF expresses the fact that this subterm is a constructor term in normal form, i.e. that it is a sorted list. Without this constraint, the specification would be inconsistent. Indeed, let us consider the ground term $t = 0 \Subset ins(s(0), ins(0, \emptyset))$. This term t can be reduced into both true and false, since $ins(s(0), ins(0, \emptyset))$ is not in normal form. Using constraints of the form $. : \mathsf{NF}$ as above also permits the user to specify, directly in the rewrite rules, some ad-hoc reduction strategies for the application of rewriting. Such strategies include for instance several refinements of the innermost strategy which corresponds to the *call by value* computation in functional programming languages, where arguments are fully evaluated before the function application.

Related work. The principle of our procedure is close to test-set induction approaches [6,1]. The real novelty here is that test-sets are replaced by constrained tree grammars, the latter being more precise induction schemes. Indeed, they provide an *exact* finite description of the initial model of the given specification, (under some assumptions like sufficient completeness and termination for axioms), whereas cover-sets and test-sets are over-approximative in similar cases.

The first author and Jouannaud [4] have used tree automata techniques to generalize test set induction to specifications with non-free constructors. This work has been generalized in [5] for membership equational logic. These approaches, unlike the procedure presented in this paper, work by transforming the initial specification in order to get rid of rewrite rules for constructors. Moreover, the axioms for constructors are assumed to be unconstrained and unconditional *left-linear* rewrite rules, which is still too restrictive for the specification of structures like sets or sorted lists...

Kapur [15] has proposed a method (implemented in the system RRL) for mechanizing cover set induction if the constructors are not free. This handles in particular the specification of powerlists or sorted lists. We show in Section 5 how our method can address similar problems.

We describe in [3] two proofs, done resp. by Jared Davis and Sorin Stratulat, of a conjecture on sorted lists, done resp. by Jared Davis and Sorin Stratulat, with ACL2 using a library for ordered sets [12] and with SPIKE [6,1,17]. Both proofs require the addition of non-trivial lemmas whereas our procedure can prove the conjecture without additional lemma.

2 Preliminaries

The reader is assumed familiar with the basic notions of term rewriting [13] and first-order 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 with a function from its set of *positions* (strings of positive integers) $\mathcal{P}os(t)$ to symbols of $\mathcal{F}$ and $\mathcal{X}$, where positions are strings of positive integers. We denote the empty string (root position) by Λ. The length of a position p is denoted $|p|$. The *depth* of a term t, denoted $d(t)$, 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 denote the set of variables occurring in t by $var(t)$. 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 substitution mapping variables to variables. We use postfix notation for substitutions application and composition. A substitution σ is *grounding* for a term t if $t\sigma$ is ground.

Constraints and constrained terms. 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, $\mathcal{L}$ may contain the syntactic equality $. \approx .$ (syntactic disequality $. \not\approx .$), some (recursive) simplification ordering $. \prec .$ on ground constructor terms (for instance a lexicographic path ordering [13]), and membership $. {:} L$ to a fixed tree language $L \subseteq \mathcal{T}(\mathcal{C})$ (like for instance the languages of well sorted terms or constructor terms in normal-form). *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.

The application of substitutions is extended from terms to constraints in a straightforward way, and we may 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 $f(x,x')\, [\![c \wedge x \approx x']\!]$, where the variable x' does not occur in c.

Constrained clauses. A *literal* is an equation $s = t$ or a disequation $s \neq t$ or an oriented equation $s \rightarrow t$ between two terms. A *constrained clause* $C\, [\![c]\!]$ is a disjunction C of literals together with a constraint c. 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 $c' \wedge \neg c\sigma$ is unsatisfiable.

A *tautology* is a constrained clause $s_1 = t_1 \vee \ldots \vee s_n = t_n \llbracket d \rrbracket$ such that d is a conjunction of equational constraints, $d = u_1 \approx v_1 \wedge \ldots \wedge u_k \approx v_k$ and there exists $i \in [1..n]$ such that $s_i\sigma = t_i\sigma$ where σ is the mgu of d.

Constrained rewriting. A *conditional constrained rewrite rule* is a constrained clause of the form $\Gamma \Rightarrow l \rightarrow r \llbracket c \rrbracket$ such that Γ is a conjunction of equations, 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. The relation $s \llbracket d \rrbracket$ rewrites to $t \llbracket d \rrbracket$ by $\mathcal{R}$, denoted $s \llbracket d \rrbracket \xrightarrow[\mathcal{R}]{} t \llbracket d \rrbracket$, is defined recursively by the existence of a rule $\rho \equiv \Gamma \Rightarrow \ell \rightarrow r \llbracket c \rrbracket \in \mathcal{R}$, a position $p \in \mathcal{P}os(s)$, and a substitution σ such that $s|_p = \ell\sigma$, $t|_p = r\sigma$, $d\sigma \wedge \neg c\sigma$ is unsatisfiable, and $u\sigma \downarrow_{\mathcal{R}} v\sigma$ for all $u = v \in \Gamma$. The transitive and reflexive transitive closures, of $\xrightarrow[\mathcal{R}]{}$ are denoted $\xrightarrow[\mathcal{R}]{+}$ 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 $s \llbracket c \rrbracket$ is *reducible* by $\mathcal{R}$ if there is some $t \llbracket c \rrbracket$ such that $s \llbracket c \rrbracket \xrightarrow[\mathcal{R}]{} t \llbracket c \rrbracket$. Otherwise $s \llbracket c \rrbracket$ is called *irreducible*, or an $\mathcal{R}$-normal form. A substitution σ is *irreducible* by $\mathcal{R}$ if its image contains only $\mathcal{R}$-normal forms. A constrained term $t \llbracket c \rrbracket$ 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 that $v \downarrow_{\mathcal{R}} w$, 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 $d(\mathcal{R})$, 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 $\mathcal{R}$ 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 $C \llbracket c \rrbracket$ is an inductive theorem of $\mathcal{R}$ (denoted $\mathcal{R} \models_{\mathcal{I}nd} C \llbracket c \rrbracket$) if for all substitutions $\sigma \in sol(c)$ grounding for 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:

$$\left\{\Gamma_1 \Rightarrow f(t_1^1, \ldots, t_k^1) \rightarrow r_1 \llbracket c_1 \rrbracket, \ldots, \Gamma_n \Rightarrow f(t_1^n, \ldots, t_k^n) \rightarrow r_n \llbracket c_n \rrbracket\right\}$$

be a maximal subset of rules of $\mathcal{R}_{\mathcal{D}}$ whose left-hand sides are identical up to variable renamings $\mu_1,\ldots,\mu_n$, *i.e.* $f(t^1_1,\ldots,t^1_k)\mu_1 = f(t^2_1,\ldots,t^2_k)\mu_2 = \ldots f(t^n_1,\ldots,t^n_k)\mu_n$. We say that f is *strongly complete* wrt $\mathcal{R}$ (see [1]) if f is sufficiently complete wrt $\mathcal{R}$ and $\mathcal{R} \models_{\mathcal{I}nd} \Gamma_1\mu_1 [\![c_1\mu_1]\!] \vee \ldots \vee \Gamma_n\mu_n [\![c_n\mu_n]\!]$ for every subset of $\mathcal{R}$ as above. The system $\mathcal{R}$ is said strongly complete if every function symbol $f \in \mathcal{D}$ is strongly complete wrt $\mathcal{R}$.

3 Constrained Grammars

Constrained tree grammars have been introduced in [7], in the context of automated induction. The idea of using such formalism for induction theorem proving is also in *e.g.* [4,10], because it is known that they can generate the languages of normal-forms for arbitrary term rewriting systems.

We present in this section the definitions and results suited to our purpose.

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* $\llcorner v \lrcorner := f(\llcorner u_1 \lrcorner,\ldots,\llcorner u_n \lrcorner)\,[\![c]\!]$ *where* $f \in \mathcal{F}$, $\llcorner v \lrcorner$, $\llcorner u_1 \lrcorner,\ldots,\llcorner u_n \lrcorner \in Q$ *(modulo variable renaming) and* c *is a constraint.*

The non-terminals are always considered modulo variable renaming. In particular, we assume *wlog* (for technical convenience) that the above term $f(u_1,\ldots,u_n)$ is linear and that $var(v) \cap var(f(u_1,\ldots,u_n)) = \emptyset$.

3.1 Term Generation

We associate to a given constrained grammar $\mathcal{G} = (Q, \Delta)$ a finite set of new unary predicates of constraint 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 $\vdash^y_{\mathcal{G}}$ is defined by:

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

if there exists $\llcorner v \lrcorner := f(\llcorner u_1 \lrcorner,\ldots,\llcorner u_n \lrcorner)\,[\![c]\!] \in \Delta$ such that $f(u_1,\ldots,u_n) = v\tau$, and $y_1,\ldots,y_n$ are fresh variables. The variable y, constrained to be in the language defined by the non-terminal $\llcorner v \lrcorner$ is replaced by $f(y_1,\ldots,y_n)$ where the variables $y_1,\ldots,y_n$ are constrained to the respective languages of non-terminals $\llcorner u_1 \lrcorner,\ldots,\llcorner u_n \lrcorner$. The union of the relations $\vdash^y_{\mathcal{G}}$ for all y is denoted $\vdash_{\mathcal{G}}$ and the reflexive transitive and transitive closures of the relation $\vdash_{\mathcal{G}}$ are respectively denoted by $\vdash^*_{\mathcal{G}}$ and $\vdash^+_{\mathcal{G}}$ ($\mathcal{G}$ may be omitted).

Definition 2. *The language* $L(\mathcal{G}, \llcorner u \lrcorner)$ *is the set of ground terms* t generated *by* $\mathcal{G}$ *from a 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 write $L(\mathcal{G}, Q') = \bigcup_{\llcorner u \lrcorner \in Q'} L(\mathcal{G}, \llcorner u \lrcorner)$ 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 $\{\sigma \mid t\sigma \in L(\mathcal{G}, \llcorner u \lrcorner)\}$.

Example 1. Let us consider the sort Nat of natural integers built with the constructor symbols 0 and s. These terms are generated by the grammar with production rules $\llcorner x\lrcorner^{\mathsf{Nat}} := 0$ and $\llcorner x\lrcorner^{\mathsf{Nat}} := s(\llcorner x_2\lrcorner^{\mathsf{Nat}})$. $\diamond$

3.2 Normal Forms

In [3], we present the automatic construction of 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 language of ground $\mathcal{R}_{\mathcal{C}}$-normal forms. Its construction is a generalization of the one of [9]. 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 common instance of its elements (if they are unifiable). Due to space limitations, we cannot describe the general construction of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ here but we rather present the case where $\mathcal{R}_{\mathcal{C}}$ contains the constructor axioms given in introduction.

Example 2. Let $\mathcal{R}_{\mathcal{C}}$ contain the axioms (c_0) and (c_1) given in introduction. Let $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ contain the two productions rules given Example 1 and $\llcorner x\lrcorner^{\mathsf{Set}} := \emptyset$, $\llcorner ins(x, y)\lrcorner := ins(\llcorner x\lrcorner^{\mathsf{Nat}}, \llcorner x\lrcorner^{\mathsf{Set}})$ (for singleton lists) and $\llcorner ins(x, y)\lrcorner := ins(\llcorner x\lrcorner^{\mathsf{Nat}}, \llcorner ins(x_2, y_2)\lrcorner)\, [\![x^{\mathsf{Nat}} \prec x_2]\!]$. Note that the variables in the non terminal $\llcorner ins(x_2, y_2)\lrcorner$ in the right member of the latter production rule have been renamed in order to be distinguished from the variables in the non terminal in the left member. This grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ generates the set of ground constructor terms in normal-form for $\mathcal{R}_{\mathcal{C}}$. They represent the ordered lists of natural numbers (of sort List). $\diamond$

4 Inference System

In this section, we present an inference system for our inductive theorem proving procedure. The principle is the following: given a goal (conjecture) C, we use the grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ of Section 3.2 in order to expand C into some subgoals. All the generated subgoals must then either be deleted, following some criteria, or be reduced, using axioms or induction hypotheses, or conjectures not yet proved, providing that they are smaller than the goal to be proved. Reduced subgoals become then new conjectures and C becomes an induction hypothesis.

The deletion criteria include tautologies, forward subsumption, clauses with an unsatisfiable constraint, and constructor clause that can be detected as inductively valid, under some conditions defined precisely below. The decision of these criteria, using $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$, is discussed in Section 4.6.

The reduction of subgoals is performed with the rules defined in Sections 4.1 and 4.2. If a subgoal generated cannot be deleted or reduced, then the procedure stop with a refutation (the initial goal is not an inductive theorem of $\mathcal{R}$). If every subgoal is deleted, then the initial goal is an inductive theorem of $\mathcal{R}$.

The procedure may not terminate (the conditions in inference rules other than the deletion criteria are recursive calls of the procedure of the form $\mathcal{R} \models_{\mathcal{I}nd}$

subgoal). In this case appropriate lemmas should be added by the user in order to achieve termination.

4.1 Simplification Rules for Defined Functions

Our procedure uses the simplification rules for defined symbols presented in Figure 1. They simplify constrained clauses according to $\mathcal{R}_\mathcal{D}$ and to a set $\mathcal{H}$ of induction hypotheses (constrained clauses), which is given as the second component of the left-hand sides of rules. Inductive Rewriting simplifies goals using the axioms as well as instances of the induction hypotheses of $\mathcal{H}$, provided that they are smaller than the goal. The underlying induction principle is based on a well-founded ordering $\gg$ on constrained clauses (see [3]). This approach is more general than structural induction which is more restrictive concerning simplification with induction hypotheses (see e.g. [6]). Inductive Contextual Rewriting can be viewed as a generalization of a rule in [18] to handle constraints by recursively discharging them as inductive conjectures. Rewrite Splitting simplifies a clause which contains a subterm matching some left member of rule of $\mathcal{R}_\mathcal{D}$. This inference checks moreover that all cases are covered for the application 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: $\big(\{C\,[\![c]\!]\}, \mathcal{H}\big) \rightarrow_\mathcal{D} \{C'\,[\![c]\!]\}$
if $\quad C\,[\![c]\!] \xrightarrow[\rho,\sigma]{} C'\,[\![c]\!]$, $l\sigma > r\sigma$ and $l\sigma > \Gamma\sigma$
where $\rho = \Gamma \Rightarrow l \rightarrow r\,[\![c]\!] \in \mathcal{R}_\mathcal{D} \cup \{\psi \;\mid \psi \in \mathcal{H}$ and $C\,[\![c]\!] \gg \psi\}$

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

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

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

Fig. 1. Simplification Rules for Defined Functions

4.2 Simplification Rules for Constructors

The simplification rules for constructors are presented in Figure 2. 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 $\mathcal{R}$.

Rewriting: $\big\{C\,\llbracket c\rrbracket\big\} \to_{\mathcal{C}} \big\{C'\,\llbracket c\rrbracket\big\}$ 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: $\big\{C[l\sigma]_p\,\llbracket c\rrbracket\big\} \to_{\mathcal{C}} \big\{C[r\sigma]_p\,\llbracket c \wedge c'\sigma\rrbracket, C[l\sigma]_p\,\llbracket c \wedge \neg c'\sigma\rrbracket\big\}$
if $l \to r\,\llbracket c'\rrbracket \in \mathcal{R}_\mathcal{C}$, $l\sigma > r\sigma$, and neither $c'\sigma$ nor $\neg c'\sigma$ is a subformula of c

Deletion: $\big\{C\,\llbracket c\rrbracket\big\} \to_{\mathcal{C}} \emptyset$ if $C\,\llbracket c\rrbracket$ is a tautology or c is unsatisfiable

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

Fig. 2. Simplification Rules for Constructors

Simplification: $\dfrac{\big(\mathcal{E} \cup \big\{C\,\llbracket c\rrbracket\big\}, \mathcal{H}\big)}{\big(\mathcal{E} \cup \mathcal{E}', \mathcal{H}\big)}$ if $\big\{C\,\llbracket c\rrbracket\big\} \to_{\mathcal{C}} \mathcal{E}'$

Inductive Simplification: $\dfrac{\big(\mathcal{E} \cup \big\{C\,\llbracket c\rrbracket\big\}, \mathcal{H}\big)}{\big(\mathcal{E} \cup \mathcal{E}', \mathcal{H}\big)}$ if $\big(\{C\,\llbracket c\rrbracket\}, \mathcal{E} \cup \mathcal{H}\big) \to_{\mathcal{D}} \mathcal{E}'$

Narrowing: $\dfrac{\big(\mathcal{E} \cup \big\{C\,\llbracket c\rrbracket\big\}, \mathcal{H}\big)}{\big(\mathcal{E} \cup \mathcal{E}_1 \cup \ldots \cup \mathcal{E}_n, \mathcal{H} \cup \big\{C\,\llbracket c\rrbracket\big\}\big)}$
if $\big\{C_i\,\llbracket c_i\rrbracket\big\} \to_{\mathcal{C}} \mathcal{E}_i$, where $\{C_1\,\llbracket c_1\rrbracket, \ldots, C_n\,\llbracket c_n\rrbracket\}$ is the set of all clauses such that $C\,\llbracket c\rrbracket \vdash^* C_i\,\llbracket c_i\rrbracket$ and $d(C_i) - d(C) \leq d(\mathcal{R}) - 1$

Inductive Narrowing: $\dfrac{\big(\mathcal{E} \cup \big\{C\,\llbracket c\rrbracket\big\}, \mathcal{H}\big)}{\big(\mathcal{E} \cup \mathcal{E}_1 \cup \ldots \cup \mathcal{E}_n, \mathcal{H} \cup \big\{C\,\llbracket c\rrbracket\big\}\big)}$
if $\big(C_i\,\llbracket c_i\rrbracket, \mathcal{E} \cup \mathcal{H} \cup \big\{C\,\llbracket c\rrbracket\big\}\big) \to_{\mathcal{D}} \mathcal{E}_i$, where $\{C_1\,\llbracket c_1\rrbracket, \ldots, C_n\,\llbracket c_n\rrbracket\}$ is the set of all clauses such that $C\,\llbracket c\rrbracket \vdash^+ C_i\,\llbracket c_i\rrbracket$ and $d(C_i) - d(C) \leq d(\mathcal{R}) - 1$

Subsumption: $\dfrac{\big(\mathcal{E} \cup \big\{C\,\llbracket c\rrbracket\big\}, \mathcal{H}\big)}{\big(\mathcal{E}, \mathcal{H}\big)}$ if $C\,\llbracket c\rrbracket$ is subsumed by another clause of $\mathcal{R} \cup \mathcal{E} \cup \mathcal{H}$

Disproof: $\dfrac{\big(\mathcal{E} \cup \big\{C\,\llbracket c\rrbracket\big\}, \mathcal{H}\big)}{\big(\bot, \mathcal{H}\big)}$ if no other rule applies to the clause $C\,\llbracket c\rrbracket$

Fig. 3. Induction Inference Rules

4.3 Induction Inference Rules

The main inference system is displayed in Figure 3. Its rules apply to pairs $(\mathcal{E}, \mathcal{H})$ whose components are respectively the sets of current conjectures and of inductive hypotheses. Two inference rules below, Narrowing and Inductive Narrowing, use the grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ for instantiating variables. In order to be able to apply these inferences, we shall initiate the process by adding to the conjectures one membership constraint for each variable.

Definition 3. *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, resp. Inductive Simplification, reduces conjectures according to the rules of Section 4.2, resp. 4.1. 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 of Figure 1 (otherwise, if one instance cannot be simplified, then the rule Inductive Narrowing cannot be applied). For sake of efficiency, the application can be restricted to so called *induction variables*, as defined in [1] while preserving all the results of the next section. Narrowing is similar and uses the rules of Figure 2 for simplification. This rule permits to eliminate the ground reducible constructor terms in a clause by simplifying their instances, while deriving conjectures considered as new subgoals. The criteria on depth 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 can be replaced by 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 $\mathcal{E}$).

Example 3. Let us come back to the running example of sorted lists, with the constructor system $\mathcal{R}_{\mathcal{C}}$ containing (c_0) and (c_1) and the defined system $\mathcal{R}_{\mathcal{D}}$ containing the axioms (m'_0-m'_3) given in introduction[1] together with the following axioms defining a variant $\in$ for the membership:

$$x \in \emptyset \to \mathsf{false} \tag{m_0}$$

$$x_1 \in ins(x_2, y) \to \mathsf{true}\,\llbracket x_1 \approx x_2 \rrbracket \tag{m_1}$$

$$x_1 \in ins(x_2, y) \to x_1 \in y\,\llbracket x_1 \not\approx x_2 \rrbracket \tag{m_2}$$

We show, using our procedure, that the conjecture $x \Subset y = x \in y$ is an inductive theorem of $\mathcal{R}$, i.e. that the two variants $\Subset$ and $\in$ of membership are equivalent. The normal-form grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ is described in example 2. The decoration of the conjecture with its non-terminal gives the two clauses: $x \Subset y = x \in y\,\llbracket x{:}\llcorner x^{\mathsf{Nat}}_{\lrcorner}, y{:}\llcorner x^{\mathsf{Set}}_{\lrcorner} \rrbracket$ and $x \Subset y = x \in y\,\llbracket x{:}\llcorner x^{\mathsf{Nat}}_{\lrcorner}, y{:}\llcorner ins(x_1, y_1)\lrcorner \rrbracket$.

The application of the production rules of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ to the first of these clauses (in Narrowing) gives: $x \Subset \emptyset = x \in \emptyset$ which is reduced, using (m'_0) and (m_0), to the tautology $\mathsf{false} = \mathsf{false}$. For the second clause, applying $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ returns:

$$x \Subset ins(x_1, \emptyset) = x \in ins(x_1, \emptyset)\,\llbracket x, x_1{:}\llcorner x^{\mathsf{Nat}}_{\lrcorner} \rrbracket \tag{1}$$

[1] In (m'_2), the constraints $y_1 \approx ins(x_2, y_2), y_1{:}\mathsf{NF}$ can be replaced by $y_1{:}\llcorner ins(x_2, y_2)\lrcorner$.

$$x \Subset ins(x_1, ins(x_2, \emptyset)) = x \in ins(x_1, ins(x_2, \emptyset)) \llbracket x, x_1, x_2 \colon \llcorner x \lrcorner^{\mathsf{Nat}}, x_1 \prec x_2 \rrbracket \quad (2)$$

$$x \Subset ins(x_1, ins(x_2, y_2)) = x \in ins(x_1, ins(x_2, y_2)) \llbracket x, x_1, x_2 \colon \llcorner x \lrcorner^{\mathsf{Nat}}, y_2 \colon \llcorner ins(x_3, y_3) \lrcorner, x_1 \prec x_2, x_2 \prec x_3 \rrbracket \quad (3)$$

The subgoals (1) and (2) can be simplified by Rewrite Splitting with (m'_1), (m'_2) and (m'_3) into clauses reduced into tautologies (see [3] for details).

The subgoal (3) is implified by Rewrite Splitting with $(\mathsf{m}'_1\text{-}\mathsf{m}'_3)$ into 3 clauses. Let us consider the third one, obtained with (m'_3): $x \Subset ins(x_2, y_2) = x \in ins(x_1, ins(x_2, y_2)) \llbracket x, x_1, x_2, x_3 \colon \llcorner x \lrcorner^{\mathsf{Nat}}, y_2 \colon \llcorner ins(x_3, y_3) \lrcorner, x_1 \prec x_2, x_2 \prec x_3, x_1 \prec x \rrbracket$. It is simplified by Inductive Rewriting with (m_2) into:
$x \Subset ins(x_2, y_2) = x \in ins(x_2, y_2) \llbracket x, x_2, x_3 \colon \llcorner x \lrcorner^{\mathsf{Nat}}, y_2 \colon \llcorner ins(x_3, y_3) \lrcorner, x_2 \prec x_3 \rrbracket$.
At this point, we are allowed to use the conjecture $x \Subset y = x \in y$ as an induction hypothesis with Inductive Rewriting, it returns the tautology:

$$x \Subset ins(x_2, y_2) = x \Subset ins(x_2, y_2) \llbracket x, x_2, x_3 \colon \llcorner x \lrcorner^{\mathsf{Nat}}, y_2 \colon \llcorner ins(x_3, y_3) \lrcorner, x_2 \prec x_3 \rrbracket$$

The ommited details in the proof of the conjecture can be found in [3]. Note that this proof does not require the manual addition of lemma. ◇

4.4 Soundness and Completeness

We show now that our inference system is sound and refutationally complete. The proof of soundness is not straightforward. The main difficulty is to make sure that the exhaustve application of the rules preserve a counterexample when one exists. We will show more precisely that a *minimal* counterexample is preserved along a *fair* derivation.

A *derivation* is a sequence of inference steps generated by a pair of the form $(\mathcal{E}_0, \emptyset)$, using the inference rules in $\mathcal{I}$, written $(\mathcal{E}_0, \emptyset) \vdash_{\mathcal{I}} (\mathcal{E}_1, \mathcal{H}_1) \vdash_{\mathcal{I}} \ldots$ It is called *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 principle with our approach, since lemmas can be specified 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 $\mathcal{D}_0$ be a set of unconstrained clauses and let $\mathcal{E}_0 = decorate(\mathcal{D}_0)$. If there exists a successful derivation $(\mathcal{E}_0, \emptyset) \vdash_{\mathcal{I}} (\mathcal{E}_1, \mathcal{H}_1) \vdash_{\mathcal{I}} \cdots$ then $\mathcal{R} \models_{\mathcal{I}nd} \mathcal{D}_0$.*

Proof. (sketch, see [3] for a complete proof). The proof uses the fact that, under the hypotheses of Theorem 1, $\mathcal{R} \models_{\mathcal{I}nd} \mathcal{E}_0$ implies $\mathcal{R} \models_{\mathcal{I}nd} \mathcal{D}_0$.

Intuitively, the reason is that in order to show that $\mathcal{R} \models_{\mathcal{I}nd} \mathcal{D}_0$, it is sufficient to show that $\mathcal{R} \models \mathcal{D}_0\sigma$ for all substitutions σ whose images contain only ground constructor terms in normal form. Every ground σ can indeed be normalized into a substitution of this form because $\mathcal{R}_{\mathcal{C}}$ is terminating and $\mathcal{R}$ sufficiently complete. By definition of the decoration, the membership constraints and by construction of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$, this sufficient condition is a consequence of $\mathcal{R} \models_{\mathcal{I}nd} \mathcal{E}_0$.

We then show that $\mathcal{R} \models_{\mathcal{I}nd} \mathcal{E}_0$ by minimal counter-example. Assume that $\mathcal{R} \not\models_{\mathcal{I}nd} \mathcal{E}_0$ and let D_0 be a clause, minimal wrt $\gg$, in the set:

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

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 in [3] that whatever inference, other than Disproof, is applied to $C\,[\![c]\!]$, a contradiction is obtained, hence that the above derivation is not successful. □

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 $\mathcal{D}_0$ be a set of unconstrained clauses and let $\mathcal{E}_0 = \mathit{decorate}(\mathcal{D}_0)$. If $\mathcal{R} \not\models_{\mathcal{I}nd} \mathcal{E}_0$, then all fair derivations starting from $(\mathcal{E}_0, \emptyset)$ end up with $(\bot, \mathcal{H})$.*

When we assume that all the variables in goals are decorated (restricting the domain for this variables to ground constructor irreducible terms), the above hypotheses that $\mathcal{R}_{\mathcal{C}}$ is terminating and $\mathcal{R}$ 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. (sketch). We use the second part of the proof of Theorem 1 (which does not use the sufficient completeness of $\mathcal{R}$ and termination of $\mathcal{R}_{\mathcal{C}}$). With the hypothesis that the clauses of $\mathcal{E}_0$ are decorated, the fact given at the beginning of this proof is indeed no more needed ($\mathcal{D}_0 = \mathcal{E}_0$). The restriction to substitutions into ground constructor normal forms in order to show that $\mathcal{R} \not\models_{\mathcal{I}nd} \mathcal{E}_0$ is made explicit by the membership constraints in the decoration. □

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 $(\bot, \mathcal{H})$.*

We shall see in Section 5 some example of applications of Theorem 2 and Corollary 2 to specifications which are not sufficiently complete.

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 $\mathcal{R} \not\models_{\mathcal{I}nd} \mathcal{E}_0$.*

4.5 Handling Non-terminating Constructor Systems

Our procedure applies rules of $\mathcal{R}_\mathcal{C}$ and $\mathcal{R}_\mathcal{D}$ only when they reduce the terms wrt the given simplification ordering $>$. 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). Note that this is in contrast with other procedures like [16] where the termination of the whole system $\mathcal{R}$ is required.

If $\mathcal{R}_\mathcal{C}$ is non-terminating then one can apply a constrained completion technique [16] in order to generate an equivalent orientable theory (with ordering constraints). The theory obtained (if the completion succeeds) can then be handled by our approach.

Example 4. Consider this non-terminating system for sets: $\{ins(x, ins(x, y)) = ins(x, y), ins(x, ins(x', y)) = ins(x', ins(x, y))\}$. Applying the completion procedure we obtain the constrained rules (c_0) and (c_1). $\diamond$

4.6 Decision Procedures for Conditions in Inferences

Constrained tree grammars are involved in the inferences Narrowing and Inductive Narrowing in order to generate subgoals from goals, by instantiation using the productions rules. They are also the key for the decision procedures applied in order to check the conditions of constraint unsatisfiability (in rules for rewriting and Inductive Deletion, Deletion, Subsumption), ground irreducibility and validity of ground irreducible constructor clauses (in the rules Validity, hence Simplification, and Disproof). These conditions are decided by reduction into the decision problem of emptiness (of $L(\mathcal{G}, \llcorner u \lrcorner)$) for constrained tree grammars build from $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$. The decision rely on similar decision results for constrained tree automata, some cases are summarized in [8]. The reductions are detailed in [3].

5 Handling Partial Specifications

The example of sorted lists (Example 3) can be treated with our procedure because it is based on a sufficiently complete and ground confluent conditional constrained TRS $\mathcal{R}$ whose constructor part $\mathcal{R}_\mathcal{C}$ is terminating. Indeed, under these hypotheses, Theorem 1 ensures the soundness of our procedure for proving inductive conjectures on this specification, and Corollary 1 and Theorem 3 ensure respectively refutational completeness and soundness of disproof.

For sound proofs of inductive theorems wrt specifications which are not sufficiently complete, we can rely on Theorem 2 and Corollary 2 which do not require sufficient completeness of the specification but instead suppose that the conjecture is decorated, i.e. that each of its variables is constrained to belong to a language associated to a non-terminal of the normal-form (constrained) grammar. In this section, we propose two applications of this principle of decoration of conjectures to the treatment of partial specifications.

Partially Defined Functions. An inductive proof of a decorated conjecture C in $\mathcal{R}$ remains valid in an extension of $\mathcal{R}$ (possibly not complete).

Theorem 4. *Assume that $\mathcal{R}$ is sufficiently complete and let $\mathcal{R}'$ be an consistent extension of $\mathcal{R}$ where ${\mathcal{R}_\mathcal{C}}' = \mathcal{R}_\mathcal{C}$ and ${\mathcal{R}_\mathcal{D}}' = \mathcal{R}_\mathcal{D} \cup {\mathcal{R}_\mathcal{D}}''$ (${\mathcal{R}_\mathcal{D}}''$ defines additional partial defined functions). Let $\mathcal{E}_0$ be a set of decorated constrained clauses. Every derivation $(\mathcal{E}_0, \emptyset) \vdash_\mathcal{I} \cdots$ successful wrt $\mathcal{R}$ is also a successful derivation wrt $\mathcal{R}'$.*

In [3], we use Theorem 4 for the proof of conjectures on an extension of the specification of Example 3 with the incomplete definition of a function *min*.

Partial Constructors. The restriction to decorated conjectures also permits to deal with partial constructor functions. In this case, we are generally interested in proving conjectures only for constructor terms in the definition domain of the defined function (well-formed terms).

In [3], we present an example of automatic proof where $\mathcal{R}_\mathcal{C}$ is such that the set of well-formed terms is the set of constructor $\mathcal{R}_\mathcal{C}$-normal forms. Hence, decorating the conjecture with grammar's non-terminals, as in Theorem 2, amounts in this case at restricting the variables to be instantiated by well-formed terms.

The example is a specification of powerlists (lists of 2^n integers stored in the leaves of a complete binary tree) also treated in [15]. A particularity of this example is that $\mathcal{R}_\mathcal{C}$ contains constraints of the form $t \sim t'$ meaning that t and t' are well-formed lists of the same length (*i.e.* balanced trees of the same depth). Such constraints are added to $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ and we show that emptiness is decidable for these grammars by reduction to the same problem for visibly tree automata with one memory [11].

6 Conclusion

We have proposed a procedure for automated inductive theorem proving in specification made of conditional and constrained rewrite rules. Constraints in rules can serve to transform non terminating specifications into terminating ones (ordering constraints), define ad-hoc evaluation strategies (normal form constraints), or for the analysis of trace properties of infinite state systems like security protocols (constraints of membership in a regular tree language representing faulty traces [2]). The expressiveness and efficiency of the procedure are obtained by the use of constrained tree grammars as a finite representation of the initial model of specifications. They are used both as induction schema and for decision procedures.

The procedure presented in this paper is currently under implementation, on the top of the theorem prover SPASS, whose main loop is used for TA decision procedure by saturation, following the approach of e.g. [14].

Acknowledgments

We wish to thank Michael Rusinowitch, Hubert Comon-Lundh, Laurent Fribourg and Deepak Kapur for the fruitful discussions that we had together regarding this work. We are also grateful to Jared Davis and Sorin Stratulat for having processed the example on sorted lists with respectively ACL2 and SPIKE.

References

1. Bouhoula, A.: Automated theorem proving by test set induction. Journal of Symbolic Computation 23(1), 47–77 (1997)
2. Bouhoula, A., Jacquemard, F.: Verifying regular trace properties of security protocols with explicit destructors and implicit induction. In: Proc. of the workshop FCS-ARSPA, pp. 27–44 (2007)
3. Bouhoula, A., Jacquemard, F.: Automated induction with constrained tree automata. Research Report LSV-08-07, `http://www.lsv.ens-cachan.fr/Publis`
4. Bouhoula, A., Jouannaud, J.-P.: Automata-driven automated induction. Information and Computation 169(1), 1–22 (2001)
5. Bouhoula, A., Jouannaud, J.-P., Meseguer, J.: Specification and proof in membership equational logic. Theoretical Computer Science 236(1-2), 35–132 (2000)
6. Bouhoula, A., Rusinowitch, M.: Implicit induction in conditional theories. Journal of Automated Reasoning 14(2), 189–235 (1995)
7. Comon, H.: Unification et disunification. Théories et applications. PhD thesis, Institut Polytechnique de Grenoble (France) (1988)
8. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, C. Löding, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications (2007), `http://www.grappa.univ-lille3.fr/tata`
9. Comon, H., Jacquemard, F.: Ground reducibility is exptime-complete. Information and Computation 187(1), 123–153 (2003)
10. Comon-Lundh, H.: Handbook of Automated Reasoning, ch. 14. Elsevier, Amsterdam (2001)
11. Comon-Lundh, H., Jacquemard, F., Perrin, N.: Tree automata with memory, visibility and structural constraints. In: Seidl, H. (ed.) FOSSACS 2007. LNCS, vol. 4423, pp. 168–182. Springer, Heidelberg (2007)
12. Davis, J.: Finite set theory based on fully ordered lists. In: In 5th International Workshop on the ACL2 Theorem Prover and Its Applications (ACL2, 2004) Sets Library Website (2004), `http://www.cs.utexas.edu/users/jared/osets/Web`
13. Dershowitz, N., Jouannaud, J.-P.: Rewrite systems. In: Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics, pp. 243–320. MIT Press, Cambridge (1990)
14. Jacquemard, F., Rusinowitch, M., Vigneron, L.: Tree automata with equality constraints modulo equational theories. Journal of Logic and Algebraic Programming (to appear, 2008)
15. Kapur, D.: Constructors can be partial too. In: Essays in Honor of Larry Wos, MIT Press, Cambridge (1997)
16. Kirchner, C., Kirchner, H., Rusinowitch, M.: Deduction with symbolic constraints. Revue d'Intelligence Artificielle 4(3), 9–52 (1990); Special issue on Automatic Deduction
17. Stratulat, S.: A general framework to build contextual cover set induction provers. Journal of Symbolic Computation 32(4), 403–445 (2001)
18. Zhang, H.: Implementing contextual rewriting. In: Proc. 3rd Int. Workshop on Conditional Term Rewriting Systems (1992)