Adel Bouhoula
Florent Jacquemard

Automated Induction with Constrained Tree Automata

Research Report LSV-08-07

March 2008

Laboratoire Spécification et Vérification

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

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 to specify and carry on automatically with related 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

* A preliminary version of these results appeared in the proceedings of the 4th International Joint Conference on Automated Reasoning (IJCAR'08) [4].

** This work has been partially supported by INRIA/DGRSRT grants 06/I09 and 08–04 and a grant *SSHN* of the French Institute for Cooperation in the French Embassy in Tunisia.

generate induction schemas used for inductive proofs in order to minimize user interaction. However, 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 termination 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 in this case 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 [5] and [6] in the direction of handling specification with non-free constructors, with severe restrictions (see related work below).

Tree automata (TA) with constraints, or equivalently regular tree grammars with constraints, have appeared to be a well suited framework for the decision of problems related to term rewriting (see [10] for a survey). This is the case for instance of ground reducibility, the property that all the ground instances of a given term are reducible by a given term rewriting system (TRS). This property was originally shown decidable for all TRS by David Plaisted [26]; it is reducible to the (decidable) problem of emptiness for tree automata with disequality constraints (see e.g. [11]). TA with constraints permit a finite representation of the set of constructor normal-forms when $\mathcal{R}_\mathcal{C}$ is a left-linear TRS (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. [10].

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 the 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 [7, 2, 5], 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 [7, 2], 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 7.

The support of constraints permits in some cases to use the constrained completion technique of [23] in order to transform a non-terminating theory into a terminating one, by the addition of ordering constraints in constructor rules, see Section 5.6. 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:

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

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:

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

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). Such constraints can be useful in the context of system

verification. Assume that we have specified a defined symbol *trace* characterizing the set of possible sequences of events of some system i.e. $trace(\ell)$ reduces to *true* iff ℓ is a correct list of events (represented as constructor terms). Now, assume also that we have defined a regular language *Bad* (of ground constructor terms) representing lists of faulty events, by mean e.g. of a (finite) tree grammar. We can express in this way, for instance, that some undesirable event occurs eventually, or that some event is always followed (eventually) by an expected answer, or any kind of linear temporal property. We can express with the constrained conjecture $trace(y) \neq true \, [\![y : Bad]\!]$ that no bad list is a trace of the system. Hence, showing that this conjecture is an inductive consequence of the specification of the system amounts to do verification of trace properties (i.e. reachability properties). More details about this problematic, in the context of security protocol verification, are given in Section 3.7.

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:

$$x \Subset \emptyset \to false \qquad (\mathsf{m}'_0)$$

$$x_1 \Subset ins(x_2, y_2) \to true \, [\![x_1 \approx x_2]\!] \qquad (\mathsf{m}'_1)$$

$$x_1 \Subset y_1 \to false \, [\![y_1 \approx ins(x_2, y_2), x_1 \prec x_2, y_1{:}\mathsf{NF}]\!] \qquad (\mathsf{m}'_2)$$

$$x_1 \Subset ins(x_2, y_2) \to x_1 \Subset y_2 \, [\![x_2 \prec x_1]\!] \qquad (\mathsf{m}'_3)$$

The constraint $y_1{:}\mathsf{NF}$ 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. In Section 3, we elaborate on these examples on sorted lists. 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.

Some non-trivial examples, including the above one, treated with our method are given in Section 3 (sorted lists and verification of trace properties) and Section 7 (powerlists). Our procedure yields very natural and readable proofs on these examples which are difficult (if not impossible) to specify and to carry on with the most of the other induction procedures.

Related work. The principle of our procedure is close to test-set induction approaches [7, 2]. 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 soundness of cover-set [30] and test-set [7, 2] induction techniques do not require that the constructors are free. But, in this case, cover-sets and test-sets are over-approximating induction schemas, in the sense that they may represent some reducible ground terms. This may cause the failure (a result of the form "don't know") of the induction proof. On the other hand, the refutational completeness of test-set induction technique is not guaranteed in this case.

The first author and Jouannaud [5] have used tree automata techniques to generalize test set induction to specifications with non-free constructors. This work has been generalized in [6] 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...

The theorem prover of ACL2 [22] is a new version of the Boyer-Moore theorem prover, Nqthm. Its input language is a subset of the programming language Common LISP. It is a very general formalism for the specification of systems, and therefore permits in particular the specification of complex data structures mentioned above. The example of sorted lists, presented in Section 3 can be processed with ACL2, but the proof requires the user to add manually some lemmas, whereas the proof with our procedure does not require any lemma (see Section 3.5). The specification language of our approach is much less expressive than the one of ACL2, but the intention is to minimize the interaction with the user during the proof process, in order to prevent the user from time consumption and the good level of expertise (both in the system to be verified and in the theorem prover) which are often required in order to come up with the necessary key lemmas. An interactive proof on the same specification with SPIKE is also presented in Section 3.

Kapur [20] has proposed a method (implemented in the system RRL) 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, as illustrated by some examples. We show in Section 7 how our method can address similar problems.

In [27], Sengler proposes a system INKA for automated termination analysis of recursively defined algorithm over data types like sets and arrays. It can handle constructor relations, under restrictions. When it succeeds, this method provides an explicit induction scheme which can be exploited with an explicit inductive theorem proving procedure.

We lack a concrete base of comparison between our method and the two above approaches, because it was impossible for us to process our examples with INKA (which is discontinued since 1997) or RRL. Let us outline some other important differences between our procedure and these approaches. The above explicit induction procedures are not well suited for the refutation of false conjectures. When such a system fails, it is not possible to conclude whether the

conjecture is not valid or if the system need assistance from the user in order to complete the proof. On the opposite, our implicit induction procedure is refutationally complete: any false conjecture will be refuted, under the assumptions mentioned above. This property is of particular interest for debugging specifications of flawed systems or programs or also for the detection of attacks on security protocols like in [3] (see Section 3.7). Finally, unlike explicit induction systems which are hierarchical, our procedure supports mutual induction. It is crucial for handling mutually recursive functions [2].

2 Preliminaries

The reader is assumed familiar with the basic notions of term rewriting [16] 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. The most general common instance of some terms $t_1, \ldots, t_n$ is denoted by $\mathrm{mgi}(t_1, \ldots, t_n)$.

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 [16]), 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 $s \to t$ 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 $s_1 = t_1 \vee \ldots \vee s_n = t_n \, [\![d]\!]$ 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.

Orderings. A *reduction ordering* is a well-founded ordering on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ monotonic wrt contexts and substitutions. A *simplification ordering* is a reduction ordering which moreover contains the strict subterm ordering. We assume from now on given a simplification ordering $>$ total on $\mathcal{T}(\mathcal{F})$, defined, *e.g.*, on the top of a precedence as an lpo $\succ_{lpo}$ [16].

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

Constrained rewriting. A *conditional constrained rewrite rule* is a constrained clause of the form $\Gamma \Rightarrow l \to r \, [\![c]\!]$ such that Γ is a conjunction of equations, 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 $s \, [\![d]\!]$ rewrites to $t \, [\![d]\!]$ by $\mathcal{R}$, denoted $s \, [\![d]\!] \xrightarrow[\mathcal{R}]{} t \, [\![d]\!]$, is defined recursively by the existence of a rule $\rho \equiv \Gamma \Rightarrow \ell \to r \, [\![c]\!] \in \mathcal{R}$, 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 $s\,[\![c]\!]$ 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 image contains only $\mathcal{R}$-normal forms. A constrained term $t\,[\![c]\!]$ is *ground reducible* (resp. *ground irreducible*) if $t\sigma$ is reducible (resp. irreducible) for every irreducible solution σ of c grounding for t.

The system $\mathcal{R}$ is *terminating* if there is no infinite sequence $t_1 \xrightarrow[\mathcal{R}]{} t_2 \xrightarrow[\mathcal{R}]{} \ldots$, $\mathcal{R}$ is *ground confluent* if for any ground terms $u, v, w \in \mathcal{T}(\mathcal{F})$, $v \xleftarrow[\mathcal{R}]{*} u \xrightarrow[\mathcal{R}]{*} w$, implies 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 $C\,[\![c]\!]$ is an inductive theorem of $\mathcal{R}$ (denoted $\mathcal{R} \models_{\mathcal{I}nd} C\,[\![c]\!]$) if for all substitutions $\sigma \in sol(c)$ 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:

$$\Big\{\Gamma_1 \Rightarrow f(t_1^1, \ldots, t_k^1) \rightarrow r_1\,[\![c_1]\!], \ldots, \Gamma_n \Rightarrow f(t_1^n, \ldots, t_k^n) \rightarrow r_n\,[\![c_n]\!]\Big\}$$

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.* $f(t_1^1, \ldots, t_k^1)\mu_1 = f(t_1^2, \ldots, t_k^2)\mu_2 = \ldots f(t_1^n, \ldots, t_k^n)\mu_n$. We say that f is *strongly complete* wrt $\mathcal{R}$ (see [2]) 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 Sorted Lists and Verification of Trace Properties

In this section, we present some examples for motivating the techniques introduced in this paper. These examples illustrate the fact that our approach supports constraints in the axioms (both for constructor and defined functions)

and the conjectures. Note that constrained rules are not supported by test set induction procedures.

3.1 Constructor Specification, Normal Form Grammar

Consider a signature with sort $\mathcal{S} = \{\mathsf{Bool}, \mathsf{Nat}, \mathsf{Set}\}$, and constructor symbols:

$$\mathcal{C} = \big\{ \mathit{true}, \mathit{false} : \mathsf{Bool},\ 0 : \mathsf{Nat},\ s : \mathsf{Nat} \to \mathsf{Nat},\ \emptyset : \mathsf{Set},\ \mathit{ins} : \mathsf{Nat} \times \mathsf{Set} \to \mathsf{Set} \big\}$$

and a constructor rewrite system for ordered lists without duplication:

$$\mathcal{R}_{\mathcal{C}} = \left\{ \begin{array}{l} \mathit{ins}(x_1, \mathit{ins}(x_2, y)) \to \mathit{ins}(x_2, y)\, [\![x_1 \approx x_2]\!] \\ \mathit{ins}(x_1, \mathit{ins}(x_2, y)) \to \mathit{ins}(x_2, \mathit{ins}(x_1, y))\, [\![x_1 \succ x_2]\!] \end{array} \right\}$$

Note the presence of constraints in these rewrite rules. The equality constraint in the first rule permits the elimination of (successive) redundancies in lists, and the ordering constraint in the second rule ensures that the application of this rule will sort the lists. Note that the first rule actually corresponds to the unconstrained rewrite rule: $\mathit{ins}(x, \mathit{ins}(x, y)) \to \mathit{ins}(x, y)$. As outlined in introduction, this rule cannot be handled by the procedures of [5, 6], because it is not not left-linear.

Constrained grammar are presented formally in Section 4. In this section, we shall only give a taste of this formalism and how their are used in the automatic inductive proof of conjectures.
The set of ground $\mathcal{R}_{\mathcal{C}}$-normal forms is described by the following set of patterns:

$$\begin{array}{rl} \mathrm{NF}(\mathcal{R}_{\mathcal{C}}) = & \{x : \mathsf{Bool}\} \cup \{x : \mathsf{Nat}\} \cup \{\emptyset\} \cup \{\mathit{ins}(x, \emptyset) \mid x : \mathsf{Nat}\} \\ & \cup \{\mathit{ins}(x_1, \mathit{ins}(x_2, y)) \mid x_1, x_2 : \mathsf{Nat}, \mathit{ins}(x_2, y) \in \mathrm{NF}(\mathcal{R}_{\mathcal{C}}), x_1 \prec x_2\} \end{array}$$

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

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

For the last subset of $\mathrm{NF}(\mathcal{R}_{\mathcal{C}})$, we need to apply the negation of the constraint $x_1 \approx x_2 \vee x_1 \succ x_2$ in the production rules of the grammar. For this purpose, we add the production rule:

$$\llcorner \mathit{ins}(x, y) \lrcorner \quad := \quad \mathit{ins}(\llcorner x \lrcorner^{\mathsf{Nat}}, \llcorner \mathit{ins}(x_2, y_2) \lrcorner)\, [\![x^{\mathsf{Nat}} \prec x_2]\!]$$

Note that the variables in the non terminal $\llcorner \mathit{ins}(x_2, y_2) \lrcorner$ in the right member of the above production rule have been renamed in order to be distinguished from the variables in the non terminal in the left member.

3.2 Defined Symbols and Conjectures

We complete the above signature with the set of defined function symbols:

$$\mathcal{D} = \{\mathit{sorted} : \mathsf{Set} \to \mathsf{Bool}, \in, \Subset\colon \mathsf{Nat} \times \mathsf{Set} \to \mathsf{Bool}\}$$

and the conditional constrained TRS $\mathcal{R}_\mathcal{D}$ containing the following rules:

$$\mathit{sorted}(\emptyset) \to \mathit{true} \tag{s_0}$$
$$\mathit{sorted}(\mathit{ins}(x, \emptyset)) \to \mathit{true} \tag{s_1}$$
$$\mathit{sorted}(\mathit{ins}(x_1, \mathit{ins}(x_2, y))) \to \mathit{sorted}(\mathit{ins}(x_2, y)) \, [\![x_1 \prec x_2]\!] \tag{s_2}$$

Note that there is no axiom for the case $[\![x_1 \succeq x_2]\!]$. The defined function *sorted* is nevertheless sufficiently complete wrt $\mathcal{R}$. We can show with an induction (on the size of the term) that every term t of the form $\mathit{sorted}(\mathit{ins}(t_1, \mathit{ins}(t_2, \ell)))$ can be reduced to a constructor term. If $t_1 \prec t_2$, then (s_2) applies and the term obtained is smaller than t. If $t_1 \succeq t_2$, then t is reducible by $\mathcal{R}_\mathcal{C}$ into the smaller $\mathit{sorted}(\mathit{ins}(t_2, \ell)))$ if $t_1 \approx t_2$ or into $\mathit{sorted}(\mathit{ins}(t_2, \mathit{ins}(t_1, \ell)))$ if $t_1 \succ t_2$, and this latter term is furthermore reduced by the rule (s_2) of $\mathcal{R}_\mathcal{D}$ into $\mathit{sorted}(\mathit{ins}(t_1, \ell))$.

The rules (m'_0-m'_3) implements a membership test restricted to ordered lists. The function $\in$ specified below another variant of a membership test on lists.

$$x \in \emptyset \to \mathit{false} \tag{m_0}$$
$$x_1 \in \mathit{ins}(x_2, y) \to \mathit{true} \, [\![x_1 \approx x_2]\!] \tag{m_1}$$
$$x_1 \in \mathit{ins}(x_2, y) \to x_1 \in y \, [\![x_1 \not\approx x_2]\!] \tag{m_2}$$

$$x \Subset \emptyset \to \mathit{false} \tag{m'_0}$$
$$x_1 \Subset \mathit{ins}(x_2, y_2) \to \mathit{true} \, [\![x_1 \approx x_2]\!] \tag{m'_1}$$
$$x_1 \Subset y_1 \to \mathit{false} \, [\![x_1 \prec x_2, y_1 \colon\!\lfloor \mathit{ins}(x_2, y_2) \rfloor]\!] \tag{m'_2}$$
$$x_1 \Subset \mathit{ins}(x_2, y_2) \to x_1 \Subset y_2 \, [\![x_2 \prec x_1]\!] \tag{m'_3}$$

Like *sorted*, the defined functions $\in$ and $\Subset$ are sufficiently complete wrt $\mathcal{R}$.

The above version of the rule (m'_2) is the formal one (the version in introduction was given in a simplified notation). Note the presence of the membership constraint $y_1 \colon\!\lfloor \mathit{ins}(x_2, y_2) \rfloor$ in (m'_2). It refers to the above normal form grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ and hence restricts the variable y_1 to be a constructor term headed by *ins* and in normal form.

One may wonder why we added this membership constrained and why a rule (m''_2) of the form $x_1 \Subset \mathit{ins}(x_2, y_2) \to \mathit{false} \, [\![x_1 \prec x_2]\!]$ would not be satisfying. The reason is that with the rule (m''_2) instead of (m'_2), the specification is not consistent. Indeed, let us consider the ground term $t = 0 \Subset \mathit{ins}(s(0), \mathit{ins}(0, \emptyset))$. Note that t is not in normal form. It can be rewritten on one hand into $0 \Subset \mathit{ins}(0, \mathit{ins}(s(0), \emptyset))$ by $\mathcal{R}_\mathcal{C}$, which is in turn rewritten into *true* using (m'_1). On the other hand, t can be rewritten into *false* by (m''_2). This second rewriting is not possible with (m'_1), because of the membership constraint in this rule.

Another idea to overcome this problem should be to add a condition as in:

$$sorted(y) = true \Rightarrow x_1 \Subset ins(x_2, y) \rightarrow false \; [\![x_1 \prec x_2]\!] \qquad (\mathsf{m}_2''')$$

The specification with (m_2''') is inconsistent as well since the term t is rewritten by $\mathcal{R}_\mathcal{C}$ into $sorted(ins(0, ins(s(0), \emptyset))$, which is rewritten into $true$ by $\mathcal{R}_\mathcal{D}$. Therefore, the addition of the membership constraint in rule (m_2') is necessary for the specification of $\Subset$.

Let us consider the two following conjectures that we are willing to prove by induction:

$$sorted(y) = true \qquad (1)$$

$$x \Subset y = x \in y \qquad (2)$$

3.3 Test Set Induction

Roughly, the principle of a proof by test set induction [7, 2] is the one presented in introduction except that:

1. the induction scheme is a *test set* (a finite set of terms).
2. variables in the goals are instantiated by terms from the test set.

Moreover, the instantiation in 2 can be restricted to so called *induction variables* (see [2]), which are the variables occurring (in a term of a goal) at a non-variable and non-root position of some left-hand sides of rules of $\mathcal{R}_\mathcal{D}$.

Let us try to prove (1) using the test set induction technique. A test set[3] for $\mathcal{R}$ (and sort Set) has to contain:

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

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

$$sorted(\emptyset) = true \qquad (3)$$

$$sorted(ins(x_1, \emptyset)) = true \qquad (4)$$

$$sorted(ins(x_1, ins(x_2, y))) = true \qquad (5)$$

Subgoals (3) and (4) are simplified by $\mathcal{R}_\mathcal{D}$ (respectively with rules (s_0) and (s_1)) into $true = true$ which is a tautology. Subgoal (5) cannot be simplified by $\mathcal{R}_\mathcal{D}$, because of the constraints in rewrite rules. Subgoal (5) does not contain any induction variable, and therefore, it cannot be further instantiated. So, the proof stops without a conclusion. Hence, we fail to prove Conjecture (1) with test set induction technique.

Concerning Conjecture (2), the specification of the rules for $\Subset$ contains membership constraints. This kind of specification is not supported by the current test-set induction procedures.

[3] This test set is an over approximating description of the set of constructor terms in normal form. For instance, the term $ins(s(0), ins(0, \emptyset))$ is an instance of the third element of the test set but it is not in normal form.

3.4 Constrained Grammars based Induction

As discussed above, we need to add appropriate constraints while instantiating the induction goals. This is precisely what constrained tree grammars do.

Our procedure, presented in Section 5, roughly works as follows: given a conjecture C we try to apply the production rules of the normal form grammar to C (instead of instantiating by terms of a test set) as long as the depth of the clauses obtained is smaller or equal to the maximal depth of a left-hand-side of $\mathcal{R}_\mathcal{D}$. All clauses obtained must be reducible by $\mathcal{R}$, or by induction hypotheses or either by others conjectures not yet proved and smaller than C. If this succeeds, the clauses obtained after simplification are considered as new subgoals and for their proof we can use C as an induction hypothesis. Otherwise, the procedure fails and we have established a disproof under some assumptions on $\mathcal{R}$.

In order to prove Conjecture (1), we constraint the variable y of this clause to belong to one of the languages defined by non-terminals (of a compatible sort) of the normal form grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$. This is not restrictive since $\mathcal{R}_\mathcal{C}$ is terminating and $\mathcal{R}$ is sufficiently complete.

$$sorted(y) = true\, [\![y{:}\llcorner x \lrcorner^{\mathsf{Set}}]\!] \tag{1.a}$$
$$sorted(y) = true\, [\![y{:}\llcorner ins(x_1, y_1) \lrcorner]\!] \tag{1.b}$$

Let us apply the above principle to the proof of Conjecture (1). The application of the production rules of the grammar to (1.a) and (1.b) returns:

$$sorted(\emptyset) = true \tag{3'}$$
$$sorted(ins(x_1, \emptyset)) = true\, [\![x_1{:}\llcorner x \lrcorner^{\mathsf{Nat}}]\!] \tag{4'}$$
$$sorted(ins(x_1, ins(x_2, \emptyset))) = true\, [\![x_1, x_2{:}\llcorner x \lrcorner^{\mathsf{Nat}}, x_1 \prec x_2]\!] \tag{5'}$$
$$sorted(ins(x_1, ins(x_2, y_2))) = true \tag{5"}$$
$$[\![x_1, x_2, x_3{:}\llcorner x \lrcorner^{\mathsf{Nat}}, y_2{:}\llcorner ins(x_3, y_3) \lrcorner, x_1 \prec x_2, x_2 \prec x_3]\!]$$

For obtaining (4'), (5') and (5"), several steps of application of the production rules of the grammar are necessary. Subgoals (3'), (4') are simplified by $\mathcal{R}_\mathcal{D}$ into a tautology, like in Section 3.3. Unlike Section 3.3, Subgoal (5') can now be simplified using the rule (s_2) of $\mathcal{R}_\mathcal{D}$, because of its constraint $x_1 \prec x_2$. Moreover, Subgoal (5") can be reduced by the rule (s_2) into:

$$sorted(ins(x_2, y_2)) = true\, [\![x_2, x_3{:}\llcorner x \lrcorner^{\mathsf{Nat}}, y_2{:}\llcorner ins(x_3, y_3) \lrcorner, x_2 \prec x_3]\!]$$

This latter subgoal can be itself simplified into $true = true$ by (1), used here as an induction hypothesis. This terminates the inductive proof of (1).

For the proof of Conjecture (2), the situation is more complicated. The decoration of the variables of (2) with non terminals of the grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ returns:

$$x \Subset y = x \in y\, [\![x{:}\llcorner x \lrcorner^{\mathsf{Nat}}, y{:}\llcorner x \lrcorner^{\mathsf{Set}}]\!] \tag{2.a}$$
$$x \Subset y = x \in y\, [\![x{:}\llcorner x \lrcorner^{\mathsf{Nat}}, y{:}\llcorner ins(x_1, y_1) \lrcorner]\!] \tag{2.b}$$

The application of the production rules of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ to these clauses gives:

$$x \Subset \emptyset = x \in \emptyset \tag{6}$$

$$x \Subset ins(x_1, \emptyset) = x \in ins(x_1, \emptyset)\, [\![x, x_1 {:}_{\llcorner} x^{\mathsf{Nat}}_{\lrcorner}]\!] \tag{7}$$

$$x \Subset ins(x_1, ins(x_2, \emptyset)) = x \in ins(x_1, ins(x_2, \emptyset))\, [\![x, x_1, x_2 {:}_{\llcorner} x^{\mathsf{Nat}}_{\lrcorner}, x_1 \prec x_2]\!] \tag{8}$$

$$x \Subset ins(x_1, ins(x_2, y_2)) = x \in ins(x_1, ins(x_2, y_2))\, [\![x, x_1, x_2 {:}_{\llcorner} x^{\mathsf{Nat}}_{\lrcorner}, y_2 {:}_{\llcorner} ins(x_3, y_3)_{\lrcorner}, x_1 \prec x_2, x_2 \prec x_3]\!] \tag{9}$$

The clause (6) is reduced, using (m'_0) and (m_0), to the tautology $\mathit{false} = \mathit{false}$.

In order to simplify (7), we restrict to the cases corresponding to the constraints of the rules (m'_1), (m'_2) and (m'_3). This technique, called *Rewrite Splitting*, is defined formally in Section 5. We obtain respectively:

$$\mathit{true} = x \in ins(x_1, \emptyset)\, [\![x_1 {:}_{\llcorner} x^{\mathsf{Nat}}_{\lrcorner}, x \approx x_1]\!] \tag{7.1}$$

$$\mathit{false} = x \in ins(x_1, \emptyset)\, [\![x_1 {:}_{\llcorner} x^{\mathsf{Nat}}_{\lrcorner}, ins(x_1, \emptyset) {:}_{\llcorner} ins(x_2, y_2)_{\lrcorner}, x \prec x_2]\!] \tag{7.2}$$

$$x \Subset \emptyset = x \in ins(x_1, \emptyset)\, [\![x_1 {:}_{\llcorner} x^{\mathsf{Nat}}_{\lrcorner}, x_1 \prec x]\!] \tag{7.3}$$

Note that the constraint in (7.2) implies that $x_1 = x_2$. All these subgoal are reduced into tautologies $\mathit{true} = \mathit{true}$ or $\mathit{false} = \mathit{false}$ using respectively the following rules of $\mathcal{R}_{\mathcal{D}}$:

- (m_1) for (7.1),
- (m_2) and (m_0) for (7.2) (with $x_1 = x_2$),
- (m'_0) for the left member of (7.3), and (m_2) then (m_0) for its right member.

The subgoal (8) is also treated by *Rewrite Splitting* with the rules (m'_1), (m'_2), (m'_3) of $\mathcal{R}_{\mathcal{D}}$, similarly as above.

Let us now finish the proof of Conjecture (2), with the subgoal (9). By rewrite splitting with the rules (m'_1), (m'_2), (m'_3), we obtain:

$$\mathit{true} = x \in ins(x_1, ins(x_2, y_2))\, [\![x, x_1, x_2, x_3 {:}_{\llcorner} x^{\mathsf{Nat}}_{\lrcorner}, y_2 {:}_{\llcorner} ins(x_3, y_3)_{\lrcorner}, x_1 \prec x_2, x_2 \prec x_3, x \approx x_1]\!] \tag{9.1}$$

$$\mathit{false} = x \in ins(x_1, ins(x_2, y_2))\, \left[\!\!\left[\begin{array}{l} x, x_1, x_2, x_3, x_4 {:}_{\llcorner} x^{\mathsf{Nat}}_{\lrcorner}, y_2 {:}_{\llcorner} ins(x_3, y_3)_{\lrcorner}, x_1 \prec x_2, x_2 \prec x_3, \\ ins(x_1, ins(x_2, y_2)) {:}_{\llcorner} ins(x_4, y_4)_{\lrcorner}, x \prec x_4 \end{array} \right]\!\!\right] \tag{9.2}$$

$$x \Subset ins(x_2, y_2) = x \in ins(x_1, ins(x_2, y_2))\, [\![x, x_1, x_2, x_3 {:}_{\llcorner} x^{\mathsf{Nat}}_{\lrcorner}, y_2 {:}_{\llcorner} ins(x_3, y_3)_{\lrcorner}, x_1 \prec x_2, x_2 \prec x_3, x_1 \prec x]\!] \tag{9.3}$$

The subgoal (9.1) is simplified by (m_1) into the tautology $\mathit{true} = \mathit{true}$.

The subgoal (9.3) is simplified by (m_2) into:

$$x \Subset ins(x_2, y_2) = x \in ins(x_2, y_2)\, [\![x, x_2, x_3 {:}_{\llcorner} x^{\mathsf{Nat}}_{\lrcorner}, y_2 {:}_{\llcorner} ins(x_3, y_3)_{\lrcorner}, x_2 \prec x_3]\!] \tag{10}$$

At this point, we are allowed to use the goal (2) as an induction hypothesis since we have perform a reduction step on the subgoals. A simplification of (10) using (2) gives the tautology:

$$x \Subset ins(x_2, y_2) = x \Subset ins(x_2, y_2)\, [\![x, x_2, x_3{:}\llcorner x \lrcorner^{\mathsf{Nat}}, y_2{:}\llcorner ins(x_3, y_3) \lrcorner, x_2 \prec x_3]\!]$$

For the subgoal (9.2), note that in the constraints, $x \prec x_4$ implies $x \prec x_1$. Hence (9.2) can be simplified by (m_2) into: $false = x \in ins(x_2, y_2)\, [\![\ldots]\!]$. A simplification of the above subgoal using (2) (as an induction hypothesis) gives: $false = x \Subset ins(x_2, y_2)\, [\![\ldots]\!]$. The above subgoal has the same constraints as (9.2), and it can be observed that this constraint implies $x \prec x_2$. Therefore, we can simplify this subgoal using (m'_2) into the tautology $false = false$.

In conclusion, Conjecture (2) can be proved with our approach based on constrained grammars without the addition of any lemmas.

3.5 Proof with ACL2

A proof of Conjecture (2) was done by Jared Davis[4] with the ACL2 theorem prover, using his library osets for finite set theory [15]. In this library, sets are implemented on fully ordered lists (wrt an ordering <<). The definition in osets of a function `insert a X`, for insertion of an element `a` to a list `X` is the same as the above axioms of $\mathcal{R}_\mathcal{C}$:

```
(defun insert (a X)
  (declare (xargs :guard (setp X)))
  (cond ((empty X) (list a))
        ((equal (head X) a) X)
        ((<< a (head X)) (cons a X))
        (t (cons (head X) (insert a (tail X))))))
```

It refers to the functions `head` and `tail` which return respectively the first (smallest) element in list (the LISP `car`) and the rest of a list (LISP `cdr`). The guard `(setp X)` ensures that `X` is a fully ordered list without duplication.

The library osets contains a definition of membership similar to the axioms of (m_0–m_2) of $\mathcal{R}_\mathcal{D}$ for the definition of $\in$:

```
(defun in (a X)
  (declare (xargs :guard (setp X)))
  (and (not (empty X))
       (or (equal a (head X))
   (in a (tail X)))))
```

Next, our defined function $\Subset$ becomes the following `inb`:

```
(defun inb (a X)
 (declare (xargs :guard (setp X)))
```

[4] Jared Davis, personal communication.

```
(and (not (empty X))
     (not (and (setp X) (<< a (head X))))
     (or (equal a (head X))
         (inb a (tail X)))))
```

The conjecture (2) becomes:

```
(defthm in-is-inb
 (equal (in a X)
        (inb a X)))
```

Using the osets library, the system proved everything except the following subgoal:

```
(IMPLIES (AND (NOT (EMPTY X))
              (SETP X)
              (<< A (HEAD X)))
         (EQUAL (IN A X) (INB A X))).
```

The following lemma permits to finish the proof:

```
(defthm head-minimal
  (implies (<< a (head X))
   (not (in a X)))
  :hints(("Goal"
  :in-theory (enable primitive-order-theory))))
```

The lemma `head-minimal` was not available to users of the library osets. It will be incorporated (together with the technical lemma for its proof) in the appropriate file of the osets library.

```
(local (defthm lemma
   (implies (and (not (empty X))
 (not (equal a (head X)))
 (not (<< a (head (tail X))))
 (<< a (head X)))
    (not (in a X)))
  :hints(("Goal"
  :in-theory (enable primitive-order-theory)
  :cases ((empty (tail X)))))))
```

Note that this proof uses several theorems and hints included in the osets library. Without this library, the ACL2 theorem prover would need the addition of several key lemmas and hints. For finding them, the user would be required both experience and a good understanding of the problem and how to solve it.

3.6 Assisted Proof with SPIKE

Conjecture (2) was proved with the last version of SPIKE by Sorin Stratulat[5]

[5] Sorin Stratulat, personal communication.

Since SPIKE does not support constrained axioms, constraints are expressed as conditions. The specification of *sorted* becomes:

```
sorted(Nil) = true;
sorted(ins(x, Nil)) = true;
x1 <= x2 = true => sorted(ins(x1, ins(x2, y))) = sorted(ins(x2, y));
x1 <= x2 = false => sorted(ins(x1, ins(x2, y))) = false;
```

The axioms for $\in$ and $\Subset$ are respectively:

```
in(x1, Nil) = false;
x1 = x2 => in(x1, ins(x2, y)) = true;
x1 <> x2 => in(x1, ins(x2, y)) = in(x1, y);
```

and

```
in'(x1, Nil) = false;
x1 = x2 => in'(x1, ins(x2, y)) = true;
x2 < x1 = true => in'(x1, ins(x2, y)) = in'(x1, y);
x1 < x2 = true, osetp(ins(x2,y)) = true => in'(x1, ins(x2, y)) = false;
x1 < x2 = true, osetp(ins(x2,y)) = false => in'(x1, ins(x2, y)) = in'(x1, y);
```

The unary predicate `osetp` characterizes ordered lists. It is defined by the following axioms.

```
osetp(Nil) = true;
osetp(ins(x, Nil)) = true;
osetp(ins(x, ins(y, z))) = and(x < y, osetp(ins(y, z)));
```

With this predicate, the conjecture is expressed as follows.

```
osetp(y) = true => in(x, y) = in'(x, y);
```

A particular user specified strategy and the following additional lemmas were necessary for the termination of the proof with SPIKE. The three first lemma are natural, the last one is less intuitive.

```
osetp(y) = true => sorted(y) = true;
osetp(ins(u1, u2)) = true => osetp(u2) = true;
u1 < u2 = true, u2 < u3 = true => u1 < u3 = true;
osetp(ins(u4, u5)) = true , u2 < u4 = true  => in(u2, u5) = false;
```

3.7 Verification of Trace Properties

We have seen in the previous sections how membership constraints can be used in the axioms of $\mathcal{R}$ for the specification of operations on complex data structures, and how our method can handle it. Our procedure can also handle membership constraints in the conjecture. This feature can be used for instance in order to restrict some terms to a particular pattern. It is very useful in the context of the

verification of infinite systems, in order to express that a trace of events belongs to a (regular) set of bad traces.

In [3] we follow this approach for the verification of security properties of cryptographic protocols, using an adaptation of the procedure of this paper in order to deal with specifications which are not necessarily confluent and sufficiently complete. In this section we wont describe in full details the specification of [3] but we shall roughly describe the main lines of the approach. Consider the following conjecture:

$$trace(y) \neq true \, [\![y{:}\llcorner x \lrcorner^{\mathsf{List}}, y{:}\llcorner x \lrcorner^{\mathsf{Bad}}]\!] \tag{11}$$

Here, the membership constraint $y{:}\llcorner x \lrcorner^{\mathsf{List}}$ restricts y to be generated by the non terminal $\llcorner x \lrcorner^{\mathsf{List}}$ of the normal form constrained tree grammar. It means that y is a constructor term in normal form (as in the above example of sorted lists) representing a list of events of a system. The second membership constraint $y{:}\llcorner x \lrcorner^{\mathsf{Bad}}$ further restricts y to belong to a regular tree language representing faulty traces (traces which lead to a state of the system corresponding to a failure, an attack for instance). Finally the clause $trace(y) \neq true$ expresses that y is not a trace of the system. Hence the above conjecture (11) means that every bad trace is not reachable.

The defined function *trace* can be specified using constrained conditional rewrite rules. For instance, in [3], we follow the approach of Paulson [25] for the inductive specification of the messages exchanges of the protocol, and of the actions of the insecure communication environment. Note also that we extends this model with equations specifying the cryptographic operations, like the following non-left-linear equation for the decryption operator *dec* in a symmetric cryptosystem: $dec\big(enc(x, y), y\big) \rightarrow x$. These axioms, sometimes referred as *explicit destructors* equations, permit a strict extension of the verification model (they allow strictly more attacks on protocols) and they are specified as constructor equations of $\mathcal{R}_{\mathcal{C}}$ in our model.

4 Constrained Tree Grammars

Constrained tree grammars have been introduced in [9], in the context of automated induction. The idea of using such formalism for induction theorem proving is also in *e.g.* [5, 12], because it is known that they can generate the languages of normal-forms for arbitrary term rewriting systems.

In this paper, we push the idea one step beyond with a full integration of tree grammars with constraints in our induction procedure. Indeed, constrained tree grammars are used here:

i. as an induction scheme (instead of test-sets), for triggering induction steps by instantiation of subgoals using production rules,
ii. as a decision procedure for checking deletion criteria, including tests like ground irreducibility or validity in restricted cases, as long as emptiness is decidable.

iii. for the definition and treatment of constraints of membership in fixed tree languages, in particular languages of normal forms.

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 $var(v) \cap var(f(u_1, \ldots, u_n)) = \emptyset$.

4.1 Languages of Terms

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 a constrained grammar* $\mathcal{G}$ *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. With the normal grammar of Section 3.4, denoted $\mathcal{G}$ in this example, we have: $L(\mathcal{G}, \llcorner x \lrcorner^{\mathsf{Bool}}) = \{true, false\}$, $L(\mathcal{G}, \llcorner x \lrcorner^{\mathsf{Nat}}) = \{0, s^n(0) \mid n > 0\}$, $L(\mathcal{G}, \llcorner x \lrcorner^{\mathsf{Set}}) = \{\emptyset\}$, $L(\mathcal{G}, \llcorner ins(x_1, x_2) \lrcorner) = \{ins(s^{n_1}(0), ins(\ldots, ins(s^{n_k}(0)))) \mid k \geq 1, n_1 < \ldots < n_k\}$, ◇

Note that every regular tree language L can be generated by a constrained tree grammar following Definitions 1 and 2, with production rules of the form: $\llcorner x \lrcorner^S := f(\llcorner x \lrcorner^{S_1}, \ldots, \llcorner x \lrcorner^{S_1})$ where $S_1, \ldots, S_n, S$ are new sorts representing the non terminals of a regular tree grammar generating L.

The intersection between the language generated by a constrained tree grammar (in some non-terminal) and a regular tree language is generated by a constrained tree grammar. The constrained grammar for the intersection is built with a product construction.

4.2 Languages of Normal Forms

The constrained grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}}) = (Q_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}}), \Delta_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}}))$ defined in Figure 1 generates the language of ground $\mathcal{R}_{\mathcal{C}}$-normal forms. Its construction is a generalization of the one of [11]. 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). For purpose of the the construction of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$, a new sort Red is added to $\mathcal{S}$, (the sort of reducible terms), and hence also a new variable x^{Red}. An example of a constrained grammar for $\mathcal{R}_{\mathcal{C}}$-normal

$$\mathcal{L}(\mathcal{R}_{\mathcal{C}}) = \left\{ \begin{array}{l} u \mid u \text{ is a strict subterm of } l \text{ for some } l \to r \,[\![c]\!] \in \mathcal{R}_{\mathcal{C}} \\ \quad \text{or } u \text{ is a subterm of } l \text{ if } c \text{ is empty} \end{array} \right\}$$

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

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

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

forms constructed this way was given in Section 3.1.

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

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

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

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

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

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

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

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

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

Using the observation that every ground constructor term is generated by $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$, we obtain as a corollary that $t \in L(\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C}), \llcorner x\lrcorner^{\mathsf{Red}})$ iff t is $\mathcal{R}_\mathcal{C}$-reducible.

5 Inference System

In this section, we present an inference system for our inductive theorem proving procedure. Let us first summarize the key steps of our procedure with the following pseudo-algorithm[6]. The complete inference system, introduced by the examples of Section 3, is presented in details in Subsections 5.2, 5.3 and 5.4.

We start with a conjecture (goal) G (a constrained clause) and a rewrite system (with conditions and constraints) $\mathcal{R}$, with a subset $\mathcal{R}_\mathcal{C}$ of constructor constrained (unconditional) rewrite rules.

[6] Note that it is only a simplified version of the procedure, for presentation purpose, in order to give an intuition of how the procedure operates.

1. compute the constrained tree grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$
2. given a goal (or subgoal) C, generate instances of C by using the production rules of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$. We obtain $C_1, \ldots, C_n$.
3. **for each** C_i, do:
   (a) **if** C_i is a tautology or C_i is a constructor clause and can be detected as inductively valid **then** delete it
   (b) **else if** we are in one of the two following cases:
      i. C_i is a constructor clause and is reducible using $\mathcal{R}_{\mathcal{C}}$, **or**
      ii. C_i contains a non-constructor symbol and is reducible using $\mathcal{R}$ and induction hypotheses

      then reduce C_i into C'_i
   (c) **else disproof** (the initial conjecture is not an inductive theorem)
4. **if** 3 did not fail **then** C becomes an *induction hypothesis*
5. **for each** C'_i, do:
   (a) **if** C'_i is a tautology or it is a constructor clause and can be detected as inductively valid or it is subsumed by an axiom or induction hypothesis **then** delete it
   (b) **otherwise** C'_i becomes a new subgoal, **go to** 2.

If every subgoal is deleted, then G is an inductive theorem of $\mathcal{R}$. The procedure may not terminate, and in this case appropriate lemmas should be added by the user in order to achieve termination.

The deletion criteria (steps 3a and 5a) include tautologies, forward subsumption, clauses with an unsatisfiable constraint, and constructor clause and can be detected as inductively valid, under some conditions defined precisely below.The procedure for testing these criteria is based on a reduction to a tree grammar non-emptiness problem (does there exist at least one term generated by a given grammar), using $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$. In particular, it should be noted that we can decide validity this way for clauses C_i which are ground irreducible [19, 21] (a notion central in inductive theorem proving / proof by consistency). It is possible to decide ground irreducibility also by mean of reduction to non-emptiness, following the lines of [11]. In Section 6, we show how such tests can be achieved effectively, providing that $\mathcal{R}$ is ground confluent, for some classes of tree grammar with equality and disequality constraints studied in former works [1, 8, 14, 11]. The extension to other kind of constraints (like *e.g.* ordering constraints) requires algorithms for corresponding classes of tree grammars (see discussions in Sections 7 and 8).

The reductions at step 3b are performed either with standard rewriting or with *ind. contextual rewriting* (case 3(b)ii) or by case analysis, (*partial splitting* in case 3(b)i and *rewrite splitting* in case 3(b)ii). These rules are defined formally in Sections 5.2 and 5.3.

5.1 Induction Ordering

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

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

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

5.2 Simplification Rules for Defined Functions

Our procedure uses the simplification rules for defined symbols presented in Figure 2. The rules in this figure define the relation $\xrightarrow[\mathcal{H}]{}_\mathcal{D}$ for simplifying constrained clauses using $\mathcal{R}_\mathcal{D}$, $\mathcal{R}$ and a given set $\mathcal{H}$ of constrained clauses considered as induction hypotheses.

Inductive Rewriting simplifies goals using the axioms of $\mathcal{R}_\mathcal{D}$ 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 the well-founded ordering $\gg$ on constrained clauses This approach is more general than structural induction which is more restrictive concerning simplification with induction hypotheses (see e.g. [7]). Inductive Contextual Rewriting can be viewed as a generalization of a rule in [29] 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: $\{C \, [\![c]\!]\} \xrightarrow[\mathcal{H}]{}_\mathcal{D} \{C' \, [\![c]\!]\}$
if $C \, [\![c]\!] \xrightarrow[\rho,\sigma]{} C' \, [\![c]\!]$, $l\sigma > r\sigma$ and $l\sigma > \Gamma\sigma$
where $\rho = \Gamma \Rightarrow l \to r \, [\![c]\!] \in \mathcal{R}_\mathcal{D} \cup \{\psi \mid \psi \in \mathcal{H}$ and $C \, [\![c]\!] \gg \psi\}$

Inductive Contextual Rewriting: $\{\Upsilon \Rightarrow C[l\sigma] \, [\![c]\!]\} \xrightarrow[\mathcal{H}]{}_\mathcal{D} \{\Upsilon \Rightarrow C[r\sigma] \, [\![c]\!]\}$
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 $\Gamma \Rightarrow l \to r \, [\![c']\!] \in \mathcal{R}_\mathcal{D}$

Rewrite Splitting: $\{C[t]_p \, [\![c]\!]\} \xrightarrow[\mathcal{H}]{}_\mathcal{D} \{\Gamma_i\sigma_i \Rightarrow C[r_i\sigma_i]_p \, [\![c \wedge c_i\sigma_i]\!]\}_{i \in [1..n]}$
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 $\Gamma_i\sigma_i \Rightarrow l_i\sigma_i \to r_i\sigma_i \, [\![c_i\sigma_i]\!]$, $i \in [1..n]$
are all the instances of rules $\Gamma_i \Rightarrow l_i \to r_i \, [\![c_i]\!] \in \mathcal{R}_\mathcal{D}$ such that $l_i\sigma_i = t$

Inductive Deletion: $\{C \, [\![c]\!]\} \xrightarrow[\mathcal{H}]{}_\mathcal{D} \emptyset$ if $C \, [\![c]\!]$ is a tautology or c is unsatisfiable

Figure 2: Simplification Rules for Defined Functions

5.3 Simplification Rules for Constructors

The simplification rules for constructors are presented in Figure 3, they define the relation $\rightarrow_{\mathcal{C}}$ for simplifying constrained clauses using $\mathcal{R}_{\mathcal{C}}$ and $\mathcal{R}$.

Rewriting simplifies goals with axioms from $\mathcal{R}_{\mathcal{C}}$. Partial Splitting eliminates ground reducible terms in a constrained clause $C \llbracket c \rrbracket$ by adding to $C \llbracket c \rrbracket$ 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: $\{C \llbracket c \rrbracket\} \rightarrow_{\mathcal{C}} \{C' \llbracket c \rrbracket\}$ if $C \llbracket c \rrbracket \xrightarrow[\mathcal{R}_{\mathcal{C}}]{+} C' \llbracket c \rrbracket$ and $C \llbracket c \rrbracket \gg C' \llbracket c \rrbracket$

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

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

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

Figure 3: Simplification Rules for Constructors

5.4 Induction Inference Rules

The main inference system is displayed in Figure 4. 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, according to the definition of term generation in Section 4.1, we shall initiate the process by adding to the conjectures one membership constraint for each variable.

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

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

Simplification, resp. Inductive Simplification, reduces conjectures according to the rules of Section 5.3, resp. 5.2. Inductive Narrowing generates new subgoals by

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

Inductive Simplification: $\dfrac{(\mathcal{E} \cup \{C \,[\![c]\!]\}, \mathcal{H})}{(\mathcal{E} \cup \mathcal{E}', \mathcal{H})}$ if $\{C \,[\![c]\!]\} \xrightarrow[\mathcal{E} \cup \mathcal{H}]{}_{\mathcal{D}} \mathcal{E}'$

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

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

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

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

Figure 4: Induction Inference Rules

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 2 (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 [2] (see Section 3.3) while preserving all the results of the next section. Narrowing is similar and uses the rules of Figure 3 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 $\mathcal{E}$).

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The result of the application of Rewrite Splitting is:

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

Then there exists k such that $\mathcal{R} \models \Gamma_k\sigma_k\delta$ for some $\delta \in Sol(c' \wedge c_k\sigma_k)$. Let $C_k \equiv \Gamma_k\sigma_k \Rightarrow C[r_k\sigma_k]_p\,[\![c' \wedge c_k\sigma_k]\!]$, we have $\mathcal{R} \not\models C_k\delta$, since $\mathcal{R} \models \Gamma_k\sigma_k\delta$, $\mathcal{R} \models t\delta = r_k\sigma_k\delta$, and $\mathcal{R} \not\models C\theta$. On the other hand, $C\theta \gg C_k\delta$ since $\{t\} >^{mul} \Gamma_k\sigma_k$, and $t > r_k\sigma_k$. This contradicts the minimality of $C\theta$.

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

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

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

Corollary 1 (Refutational completeness). *Assume that $\mathcal{R}_\mathcal{C}$ is terminating and that $\mathcal{R}$ is sufficiently complete. Let $\mathcal{D}_0$ be a set of unconstrained clauses and let $\mathcal{E}_0 = decorate(\mathcal{D}_0)$. 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. The proof is the same as for Theorem 1 except that we do no need the Fact 2 since the goals of $\mathcal{E}_0$ are already decorated. Hence we do neither need the hypotheses that $\mathcal{R}_{\mathcal{C}}$ is terminating and that $\mathcal{R}$ is sufficiently complete which where only used for the proof of Fact 2. □

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

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

We shall see in Section 7 some example of applications of Theorem 2 and Corollary 2 to specifications which are not sufficiently complete.

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

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

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

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

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

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

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

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

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

1. for one $i \in [1..n]$, $t_i\tau$ is reducible. In this case, there exists a substitution σ such that $\tau = \sigma\theta$ and $t_i \,[\![c]\!] \vdash^+ t_i\sigma \,[\![c']\!]$ and $t_i\sigma$ contains as a subterm an instance of a left-hand side of rule of $\mathcal{R}_\mathcal{C}$. Therefore, either Rewriting or Partial Splitting can be applied to $t_i\sigma \,[\![c']\!]$. It implies that Narrowing can be applied to $C \,[\![c]\!]$, which is a contradiction.
2. every $t_i\tau$ is irreducible. The term $f(t_1, \ldots, t_n)\tau$ is reducible at root position because f is strongly complete wrt $\mathcal{R}$. Then there exists σ such that $\tau = \sigma\theta$ and $f(t_1, \ldots, t_n) \,[\![c]\!] \vdash^+ f(t_1, \ldots, t_n)\sigma \,[\![c']\!]$ and moreover $f(t_1, \ldots, t_n)\sigma$ is an instance of a left-hand side of rule of $\mathcal{R}_\mathcal{D}$. Therefore, either Inductive rewriting or Rewrite Splitting can be applied. Indeed the application condition of the latter inference is a consequence of the strongly completeness of $\mathcal{R}$. Hence, the inference Inductive Narrowing can be applied to $C \,[\![c]\!]$, which is a contradiction.

In conclusion, the clause $C \,[\![c]\!]$ contains only constructor terms.

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

5.6 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), like in the example of Section 3. Note that this is in contrast with other procedures like [7, 2] where the termination of the whole system $\mathcal{R}$ is required.

If $\mathcal{R}_\mathcal{C}$ is non-terminating then one can apply e.g. the constrained completion technique [23] 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 2. 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 system of Section 3. ◇

6 Decision Procedures for Conditions in Inference Rules

We present a reduction of the conditions in the inference rules of Figures 2, 3, and 4 to emptiness decision problems for tree automata with constraints. We deduce a decision procedure for these tests in the case where the constraints in the specification are limited to syntactic equality and disequality.

We assume here that, like in Theorem 1, the inference system is applied to a set $\mathit{decorate}(\mathcal{D}_0)$ where $\mathcal{D}_0$ is a set of unconstrained clauses.

6.1 Reductions

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

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

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

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

$\Delta(t\,[\![\bigwedge_{i=1}^m x_i{:}\llcorner u_i \lrcorner \wedge d]\!])$ contains all the production rules of $\Delta_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ plus:
$\llcorner t^\circ \lrcorner := g(\llcorner t_1^\circ \lrcorner, \ldots, \llcorner t_m^\circ \lrcorner)\,[\![d]\!]$, if $t = g(t_1, \ldots, t_m)$
and every $\llcorner f^\circ(v_1^\circ, \ldots, v_n^\circ) \lrcorner := f(\llcorner s_1 \lrcorner, \ldots, \llcorner s_n \lrcorner)\,[\![\,]\!]$ such that $f(u_1, \ldots, u_n) \lhd t$,
and $\forall j \leq m$ if $v_j^\circ = x_i$ for some i, then $\llcorner s_j \lrcorner = \llcorner u_i \lrcorner$
if $v_j^\circ \in \mathcal{X} \setminus \{x_1, \ldots, x_m\}$ then $\llcorner s_j \lrcorner \in Q_{\mathrm{NF}}(\mathcal{R}c)$
if $v_j^\circ \notin \mathcal{X}$ then $\llcorner s_j \lrcorner = \llcorner v_j^\circ \lrcorner$

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

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

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

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

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

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

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

Ground (ir)reducibility. The rules Validity, hence Simplification, and Disproof (by negation) check ground irreducibility.

Lemma 6. *Ground reducibility and ground irreducibility decision are reducible to* (EI).

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

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

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

Proof. Let $C \,[\![c]\!]$ be a ground irreducible constructor constrained clause. Let $\tilde{C}$ be the constraint obtained from C by replacement of every equation $s = t$ (resp. disequation $s \neq t$) by the atom $s \approx t$ (resp. $s \not\approx t$). Since $C \,[\![c]\!]$ is ground irreducible and $\mathcal{R}$ is ground-confluent, we have that $C \,[\![c]\!]$ is valid in the initial model of $\mathcal{R}$ iff every substitution $\sigma \in sol(c)$ grounding for C is such that $\sigma \in sol\big(\tilde{C}\big)$. This is equivalent to $L\big(\mathcal{G}(C \,[\![c \wedge \neg\tilde{C}]\!])\big) = \emptyset$. □

6.2 Decision

It remains to give decision procedures for (ED) and (EI). We proceed by reduction to analogous problems on tree automata with (dis)equality constraints [10], for a class of tree grammars defined as follows.

Definition 5. *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 $f(u_1, \ldots, u_n)$.*

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 [10] 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 $u_i \not\approx u_j$ [1],
2. the constraints in c are only disequalities $s_1 \not\approx s_2$ [11],
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) [14],
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) [8]
   (i) s occurs along p in t,
   (ii) $s \approx s'$ or $s' \approx s$ is an atomic constraint of c,
   (iii) s, s' are not brothers in a subterm $f(\ldots, s, \ldots, s', \ldots)$ occurring on p.

Theorem 4. *All the conditions of the simplification rules in Figures 2,3 and the inference rules in Figure 4 are decidable or make recursive call to the procedure itself when $\mathcal{R}$ is ground confluent and, for all $l \to r\ [\![c]\!] \in \mathcal{R}_\mathcal{C}$, for all $s \approx s' \in c$, (resp. all $s \not\approx s' \in c$) s and s' are either variables or strict subterms of l (resp. variables or strict subterms occurring at sibling positions in l).*

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

The algorithms provided in the literature for the emptiness decision for the classes 1 to 4 of tree automata with equality and disequality constraints are all very costly, due to the inherent complexity of the problem. For instance, for the "easiest" class 1, the problem is EXPTIME-complete [10], see also [21, 11] concerning class 2. The problem is however less difficult for deterministic automata (*e.g.*, PTIME for class 1), like the one of Figure 1.

Cleaning algorithms, which may behave better in the average, have been proposed [8] for optimizing emptiness decision. An interesting aspect of the cleaning algorithm is its monotonicity: an incremental change on the automaton in input causes only an incremental change of the intermediate structure constructed by the algorithm for emptiness decision. This should permit to reuse such structures in our setting because all the constrained grammars of Section 6.1 are incrementally obtained from the unique normal form grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$.

Another promising approach for implementation is the use of first-order saturation techniques. It has been studied for solving various decision problem for several classes of tree automata with or without constraints [18, 13, 17].

7 Handling Partial Specifications

The example of sorted lists in Section 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. We treat the case where the specification of defined function is partial in Section 7.1, and the case where axioms for constructors are partial in Section 7.2.

7.1 Partially Defined Functions

Under the condition that the conjecture is decorated, extending a given sufficiently complete specification with additional axioms for defining partial (defined) functions preserves successful derivations.

Theorem 5. *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 $\mathcal{R}'$.*

Proof. The grammars $\mathcal{G}_{\mathrm{NF}}({\mathcal{R}_\mathcal{C}}')$ and $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ are the same. Therefore every inference step wrt $\mathcal{R}$ is also an inference step wrt $\mathcal{R}'$. □

We apply Theorem 5 to a partial extension of the specification of Section 3.

Specification of min for sorted lists. Let us complete the specification of Section 3 with a new defined symbol $min : \mathsf{Set} \to \mathsf{Nat}$ and the following rules of $\mathcal{R}_\mathcal{D}$:

$$min(ins(x, \emptyset)) \to x$$
$$min(ins(x, ins(y, z))) \to min(ins(x, z)) \, [\![x \prec y]\!]$$

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

Proof of two conjectures for min. We shall prove, using our inference system, that the two following constrained and decorated conjectures are inductive theorems of $\mathcal{R}$.

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

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

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

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

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

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

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

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

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

The proof of (13) follows the same steps.

Note that by Theorem 5 the proofs of the decorated conjectures (1.a), (1.b) and (2.a), (2.b) in Section 3 remain valid for the above extended specification.

7.2 Partial Constructors and Powerlists

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). This is possible with our procedure when $\mathcal{R}_{\mathcal{C}}$ 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.

We illustrate this approach in this section with an example of application of Theorem 2 to a non complete specification of powerlists.

Specification of powerlists. A powerlist [24] is a list of length 2^n (for $n \geq 0$) whose elements are stored in the leaves of a balanced binary tree. Kapur gives in [20] a specification of powerlists and some proofs of conjectures with an extension of RRL mentioned in introduction. This example is carried out with an extension of the algebraic specification approach where some partial constructor symbols are restricted by *application conditions*. We propose below another specification of powerlists which contains only constrained rewrite rules, and which can be efficiently handled by our method.

We consider a signature for representing powerlists of natural numbers, with the sorts: $\mathcal{S} = \{\mathsf{Nat}, \mathsf{List}\}$ and the constructor symbols:

$$\mathcal{C} = \big\{0 : \mathsf{Nat}, s : \mathsf{Nat} \to \mathsf{Nat}, v : \mathsf{Nat} \to \mathsf{List}, tie : \mathsf{List} \to \mathsf{List}, \bot : \mathsf{List}\big\}$$

The symbols 0 and s are used to represent the natural numbers in unary notation, v creates a singleton powerlist $v(n)$ of length 1 from a number n, and tie is the concatenation of powerlists. The operator tie is restricted to well balanced constructor terms of the same depth. In order to express this property, we shall consider a constructor rewrite system $\mathcal{R}_\mathcal{C}$ which reduces to $\bot$ every term $tie(s,t)$ which is not well balanced. This way, only the well defined powerlists are $\mathcal{R}_\mathcal{C}$-irreducible. For this purpose, we shall use a new binary constraint predicate $\sim$ defined on constructor terms of sort List as the smallest equivalence such that:

$$\begin{array}{c} v(x) \sim v(y) \quad \text{for all } x, y : \mathsf{Nat} \\ tie(x_1, x_2) \sim tie(y_1, y_2) \quad \text{iff } x_1 \sim x_2 \sim y_1 \sim y_2 \end{array}$$

The constructor TRS $\mathcal{R}_\mathcal{C}$ has one rule constrained by $\sim$:

$$tie(y_1, y_2) \to \bot \, [\![y_1 \not\sim y_2]\!] \quad tie(\bot, y) \to \bot \quad tie(y, \bot) \to \bot$$

Tree grammars with $\sim$-constraints on brother subterms. The normal form tree grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ associated to $\mathcal{R}_\mathcal{C}$ generates the well founded ground constructor terms. Its non-terminals, according to the construction in Section 4.2, are: $\llcorner x \lrcorner^{\mathsf{Nat}}$, $\llcorner x \lrcorner^{\mathsf{List}}$, $\llcorner \bot \lrcorner$, $\llcorner tie(x_1, x_2) \lrcorner$ and its production rules:

$$\begin{array}{l} \llcorner x \lrcorner^{\mathsf{Nat}} := 0 \quad \llcorner x \lrcorner^{\mathsf{Nat}} := s(\llcorner x_2 \lrcorner^{\mathsf{Nat}}) \quad \llcorner x \lrcorner^{\mathsf{List}} := v(\llcorner x \lrcorner^{\mathsf{Nat}}) \quad \llcorner \bot \lrcorner := \bot \\ \llcorner tie(x_1, x_2) \lrcorner := tie(\llcorner x_3 \lrcorner^{\mathsf{List}}, \llcorner x_4 \lrcorner^{\mathsf{List}}) \qquad [\![x_3^{\mathsf{List}} \sim x_4^{\mathsf{List}}]\!] \\ \llcorner tie(x_1, x_2) \lrcorner := tie\big(\llcorner tie(x_3, x_4) \lrcorner, \llcorner tie(x_5, x_6) \lrcorner\big) \; [\![tie(x_3, x_4) \sim tie(x_5, x_6)]\!] \end{array}$$

Note that all the constraints in these production rules are applied to brother subterms. We have omitted in the above list the non-terminal $\llcorner x \lrcorner^{\mathsf{Red}}$, and production rules of the form: $\llcorner x \lrcorner^{\mathsf{Red}} := tie(\llcorner x_1 \lrcorner^{\mathsf{List}}, \llcorner x_2 \lrcorner^{\mathsf{List}})\, [\![x_1^{\mathsf{List}} \not\sim x_2^{\mathsf{List}}]\!]$ or $\llcorner x \lrcorner^{\mathsf{Red}} := tie(\llcorner \bot \lrcorner, \llcorner x_2 \lrcorner^{\mathsf{List}})$.

The emptiness problem is decidable for such constrained tree grammars. This can be shown with an adaptation of the proof in [1] to $\sim$-constraints (instead of equality constraints) or also by an encoding into the visibly tree automata with one memory of [13].

Proof of a conjecture. We add to the specification a defined symbol rev: $\mathcal{D} = \{ rev : \mathsf{List} \to \mathsf{List} \}$ and a defined TRS $\mathcal{R}_\mathcal{D}$:

$$rev(\bot) \to \bot \qquad (\mathsf{r}_0)$$
$$rev(v(y)) \to v(y) \qquad (\mathsf{r}_1)$$
$$rev(tie(y_1, y_2)) \to tie(rev(y_2), rev(y_1)) \qquad (\mathsf{r}_2)$$

The conjecture is:

$$rev(rev(x)) = x \qquad (18)$$

A proof of Conjecture (18) can be found in [20]. We prove (18) by the analysis of several cases, where each case is treated quickly. As explained above, we need

to decorate its variables with non-terminals of the normal form grammar. There are three possibilities:

$$rev(rev(x)) = x \, [\![x{:}_{\llcorner} x_{\lrcorner}^{\mathsf{List}}]\!] \tag{19}$$

$$rev(rev(x)) = x \, [\![x{:}_{\llcorner} \bot_{\lrcorner}]\!] \tag{20}$$

$$rev(rev(x)) = x \, [\![x{:}_{\llcorner} tie(x_1, x_2)_{\lrcorner}]\!] \tag{21}$$

Let us apply the production rules of the grammar to Conjectures (19) and (20) (inference Inductive Narrowing). It returns respectively:

$$rev(rev(v(x))) = x \, [\![x{:}_{\llcorner} x_{\lrcorner}^{\mathsf{Nat}}]\!] \tag{22}$$

$$rev(rev(\bot)) = \bot \tag{23}$$

The subgoals (22) and (23) are reduced by the rules (r_1) and (r_0) of $\mathcal{R}_{\mathcal{D}}$ (Inductive Rewriting for Inductive Narrowing) into the respective tautologies: $v(x) = v(x) \, [\![x{:}_{\llcorner} x_{\lrcorner}^{\mathsf{Nat}}]\!]$ and $\bot = \bot$.

Now, let us apply Inductive Narrowing to Conjecture (21). The application of the production rules of the grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ returns:

$$rev(rev(tie(x_1, x_2))) = tie(x_1, x_2) \quad [\![x_1{:}_{\llcorner} x_3{}_{\lrcorner}^{\mathsf{List}} \wedge x_2{:}_{\llcorner} x_4{}_{\lrcorner}^{\mathsf{List}} \wedge x_3^{\mathsf{List}} \sim x_4^{\mathsf{List}}]\!] \tag{24}$$

$$rev(rev(tie(x_1, x_2))) = tie(x_1, x_2) \quad [\![x_1{:}_{\llcorner} x_3{}_{\lrcorner}^{\mathsf{List}} \wedge x_2{:}_{\llcorner} tie(x_4, x_5)_{\lrcorner} \wedge x_3^{\mathsf{List}} \sim tie(x_4, x_5)]\!] \tag{25}$$

$$rev(rev(tie(x_1, x_2))) = tie(x_1, x_2) \quad [\![x_1{:}_{\llcorner} tie(x_3, x_4)_{\lrcorner} \wedge x_2{:}_{\llcorner} x_5{}_{\lrcorner}^{\mathsf{List}} \wedge tie(x_3, x_4) \sim x_5^{\mathsf{List}}]\!] \tag{26}$$

$$rev(rev(tie(x_1, x_2))) = tie(x_1, x_2) \quad [\![x_1{:}_{\llcorner} tie(x_3, x_4)_{\lrcorner} \wedge x_2{:}_{\llcorner} tie(x_5, x_6)_{\lrcorner} \wedge tie(x_3, x_4) \sim tie(x_5, x_6)]\!] \tag{27}$$

Note that, with (r_2):

$$\begin{aligned} rev(rev(tie(x_1, x_2))) &\to_{\mathcal{R}_{\mathcal{D}}} rev(tie(rev(x_2), rev(x_1))) \\ &\to_{\mathcal{R}_{\mathcal{D}}} tie(rev(rev(x_1)), rev(rev(x_2))) \end{aligned}$$

Hence, the reduction of (24) with the rule (r_2) of $\mathcal{R}_{\mathcal{D}}$ gives:

$$tie(rev(rev(x_1)), rev(rev(x_2))) = tie(x_1, x_2) \quad [\![x_1{:}_{\llcorner} x_3{}_{\lrcorner}^{\mathsf{List}} \wedge x_2{:}_{\llcorner} x_4{}_{\lrcorner}^{\mathsf{List}} \wedge x_3^{\mathsf{List}} \sim x_4^{\mathsf{List}}]\!] \tag{28}$$

ans similarly for (25), (26), and (27).

This later equation (28) can be reduced by Conjecture (21), considered as an induction hypothesis (this is a case of Inductive Rewriting), giving the tautology:

$$tie(x_1, x_2)) = tie(x_1, x_2) \, [\![x_1{:}_{\llcorner} x_3{}_{\lrcorner}^{\mathsf{List}} \wedge x_2{:}_{\llcorner} x_4{}_{\lrcorner}^{\mathsf{List}} \wedge x_3^{\mathsf{List}} \sim x_4^{\mathsf{List}}]\!] \tag{29}$$

The situation is the same for the other reduced equation and this completes the proof of Conjecture (21).

8 Conclusion

A fundamental issue in automatic theorem proving by induction is the computation of a suitable finite description of the set of ground terms in normal form, which can be used as an induction scheme. Normal form constrained tree grammars are perfect induction schemes in the sense that they generate *exactly* the set of constructor terms in normal form. At the opposite, test sets and cover sets are approximated induction schemes when the constructors are not free. They may indeed also represent some reducible ground terms, and therefore may cause the failure (a result of the form "don't know") of an induction proof when constructors are not free. In this case, refutational completeness is not guaranteed. This explains the choice of constrained grammars for the incremental generation of subgoals. Constrained tree grammars are also used (by mean of emptiness test) in order to detect in some cases that constructor subgoals are inductively valid. Moreover, this formalism permits to handle naturally constraint of membership in a fixed regular tree language.

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

Constraints in rules can serve to transform non terminating specifications into terminating ones, for instance in presence of associativity and commutativity axioms (ordering constraints), define ad-hoc evaluation strategies, like e.g. innermost rewriting, directly in the axioms (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 [3]). The treatment of membership constraints permits to express in a natural way, in conjectures, trace properties for the verification of systems. This idea has been applied for the validation and research of attacks (by refutation) on security protocols in a model with explicit destructor functions [3]. These symbols represent operators like projection or decryption whose behaviour is specified with constructor axioms.

Our procedure can handle partial specifications: specifications which are not sufficiently complete and specifications with partial constructor functions in the lines of [20]. Moreover, it preserves the proofs of decorated conjectures made in a sufficiently complete specification when this specification is extended with partial symbols.

The definition of tree grammars with constraints in Section 4 is very general. It embeds some classes of grammars for which the emptiness problem is decidable (see Section 6) and also classes for which this problem is still open. Therefore,

advances in tree automata theory can benefit our approach, and we are planing to study new classes of tree automata with constraints.

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. 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.
2. A. Bouhoula. Automated theorem proving by test set induction. *Journal of Symbolic Computation*, 23(1):47–77, 1997.
3. A. Bouhoula and F. Jacquemard. Verifying regular trace properties of security protocols with explicit destructors and implicit induction. In *Proc. of the workshop FCS-ARSPA*, pages 27–44, 2007.
4. A. Bouhoula and F. Jacquemard. Automated Induction with Constrained Tree Automata. in *Proceedings of the 4th International Joint Conference on Automated Reasoning (IJCAR)*, vol. 5195 of Springer LNCS, pages 539–553, 2008.
5. A. Bouhoula and J.-P. Jouannaud. Automata-driven automated induction. *Information and Computation*, 169(1):1–22, 2001.
6. A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. *Theoretical Computer Science*, 236(1-2):35–132, 2000.
7. A. Bouhoula and M. Rusinowitch. Implicit induction in conditional theories. *Journal of Automated Reasoning*, 14(2):189–235, 1995.
8. A.C. Caron, H. Comon, J-L. Coquidé, M. Dauchet, and F. Jacquemard. Pumping, cleaning and symbolic constraints solving. In *Proc. of the 21st Int. Conf. on Automata, Languages and Programming*, 1994.
9. H. Comon. *Unification et disunification. Théories et applications*. PhD thesis, Institut Polytechnique de Grenoble (France), 1988.
10. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, C. Löding, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. `http://www.grappa.univ-lille3.fr/tata`, 2007.
11. H. Comon and F. Jacquemard. Ground reducibility is exptime-complete. *Information and Computation*, 187(1):123–153, 2003.
12. H. Comon-Lundh. *Handbook of Automated Reasoning*, chapter Inductionless Induction. Number chapter 14. Elsevier, 2001.
13. H. Comon-Lundh, F. Jacquemard, and N. Perrin. Tree automata with memory, visibility and structural constraints. In *Proc. of the 10th Int. Conf. on Found. of Software Science and Comp. Struct. (FoSSaCS'07)*, vol. 4423 of *LNCS*, pages 168–182. Springer, 2007.
14. M. Dauchet, A.-C. Caron, and J.-L. Coquidé. Automata for reduction properties solving. *Journal of Symbolic Computation*, 20, 1995.
15. J. Davis. Finite set theory based on fully ordered lists. In *In 5th International Workshop on the ACL2 Theorem Prover and Its Applications (ACL2 2004)*, 2004. Sets Library Website: `http://www.cs.utexas.edu/users/jared/osets/Web`.

16. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, pages 243–320. MIT Press, 1990.
17. J. Goubault-Larrecq. Deciding $\mathcal{H}_1$ by Resolution. *Information Processing Letters*, 95(3):401–408, 2005.
18. F. Jacquemard, M. Rusinowitch, and L. Vigneron. Tree automata with equality constraints modulo equational theories. *Journal of Logic and Algebraic Programming*, 75(2), pages 182–208, 2008.
19. 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.
20. D. Kapur. Constructors can be partial too. In *Essays in Honor of Larry Wos*. MIT Press, 1997.
21. D. Kapur, P. Narendran, D. Rosenkrantz, and H. Zhang. Sufficient completeness, ground reducibility and their complexity. *Acta Informatica*, 28:311–350, 1991.
22. M. Kaufmann, P. Manolios, and J.S. Moore. Computer-Aided Reasoning: An Approach. Kluwer Academic Publishers, 2000.
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. Jayadev Misra. Powerlist: A structure for parallel recursion. *ACM Transactions on Programming Languages and Systems*, 16(6):1737–1767, 1994.
25. Lawrence C. Paulson. The inductive approach to verifying cryptographic protocol. *Journal of Computer Security*, 6:85–128, 1998.
26. David A. Plaisted. Semantic confluence tests and completion methods. *Information and Control*, 65(2-3):182–215, 1985.
27. C. Sengler. Termination of Algorithms over Non-freely Generated Data Types. In proceedings of the 13th Int. Conf. on Automated Deduction, vol. 1104 of Springer LNCS, pages 121-135, 1996.
28. S. Stratulat. A general framework to build contextual cover set induction provers. *Journal of Symbolic Computation*, 32(4):403–445, 2001.
29. H. Zhang. Implementing contextual rewriting. In *In Proc. 3rd Int. Workshop on Conditional Term Rewriting Systems*, 1992.
30. H. Zhang, D. Kapur, and M. S. Krishnamoorthy. A mechanizable induction principle for equational specifications. In *Proc. 9th Int. Conf. on Automated Deduction*, vol. 310 of Springer LNCS, pages 162–181, 1988.