Automating Sufficient Completeness Check for Conditional and Constrained TRS

Adel Bouhoula, Florent Jacquemard

Automating Sufficient Completeness Check for Conditional and Constrained TRS*

Adel Bouhoula[1] and Florent Jacquemard[2]

[1] École Supérieure des Communications de Tunis, Tunisia. bouhoula@planet.tn
[2] INRIA Futurs & LSV UMR CNRS–ENSC, France. florent.jacquemard@inria.fr

Abstract. We present a procedure for checking sufficient completeness for conditional and constrained term rewriting systems containing axioms for constructors which may be constrained (by *e.g.* equalities, disequalities, ordering, membership...). Such axioms allow to specify complex data structures like *e.g.* sets, sorted lists or powerlists. Our approach is integrated in a framework for inductive theorem proving based on tree grammars with constraints, a formalism which permits an exact representation of languages of ground constructor terms in normal form.
The key technique used in the procedure is a generalized form of narrowing where, given a term, instead of unifying it with left members of rewrite rules, we instantiate it, at selected variables, following the productions of a constrained tree grammar, and test whether it can be rewritten. Our procedure is sound and complete and has been successfully applied to several examples, yielding very natural proofs and, in case of negative answer, a counter example suggesting how to complete the specification. Moreover, it is a decision procedure when the TRS is unconditional but constrained, for a large class of constrained constructor axioms.

Keywords: Conditional and Constrained Rewrite Systems, Sufficient Completeness, Tree Grammars.

1 Introduction

Sufficient completeness [11] is a fundamental property of algebraic specifications. It expresses that some functions are defined on every values by a given a term rewriting system (TRS) $\mathcal{R}$: given a set $\mathcal{C}$ of distinguished operators called *constructors*, used to represent values, every ground term can be rewritten to a constructor term, *i.e.* a term built only from symbols of $\mathcal{C}$. This property is strongly related to inductive theorem proving, and in particular to *ground reducibility*, the property that all ground instances (instances without variables) of a given term are reducible by a given TRS [19, 18].

Sufficient completeness is undecidable in general [12]. Decidability results have been obtained for restricted cases of unconditional TRS [13, 16, 20, 21, 18]. Tree automata with constraints have appeared to be a well suited framework in

* This work has been partially supported by the grant INRIA–DGRSRT 06/I09.

this context of decision of sufficient completeness and related properties; for a survey, see [7]. In the context of conditional specifications, the problem is much harder and the art is less developed (see section on related work below).

In this paper, we present a method for testing sufficient completeness of conditional and constrained rewrite systems with rules for constructors which can be constrained or non-left-linear. Such rules permit the axiomatization of complex data structures like *e.g.* sorted lists or powerlists, as illustrated in Section 7. Our method is based on the incremental construction of a *pattern tree* labeled by constrained terms. Every construction step is defined as a replacement of a non-terminal by a constrained tree grammar which generates the set of constructor terms irreducible by $\mathcal{R}$. The key idea is roughly that, under the assumption that $\mathcal{R}$ is ground convergent, its sufficient completeness is ensured as long as every term of the form $f(t_1, \ldots, t_n)$ can be rewritten (at the top) when f is a defined symbol and $t_1, \ldots, t_n$ are $\mathcal{R}$-irreducible constructor terms. This principle permits to restrict the positions of non-terminals to be replaced, and therefore ensures the termination of the construction.

The criterion for the verification of sufficient completeness is that all the leaves of the pattern tree are *strongly ground reducible* by $\mathcal{R}$. This latter sufficient condition for the ground reducibility by $\mathcal{R}$ requires in particular that the conditions of candidate rules of $\mathcal{R}$ (for reducibility of ground instances) are inductive consequences of $\mathcal{R}$. Therefore, when $\mathcal{R}$ is conditional, our procedure for sufficient completeness verification relies on a method for inductive theorem proving called as an oracle by the inference system. Actually, the procedure has been designed to be fully integrated within a general framework defined in [3] for inductive theorem proving. The procedure of [3] is sound and refutationally complete for the kind of conjectures considered here, and it is also based on normal form constrained tree grammars. Moreover, when $\mathcal{R}$ is unconditional, our procedure for sufficient completeness becomes a decision procedure for a large class of constrained constructor rules.

Let us summarize below a few arguments in favor of our approach. **1.** It handles axioms for constructors even with constraints. **2.** It is *sound* for ground convergent specifications and it is also *complete*, without restriction. Moreover, it does not require a transformation of the given specification in order to get rid of the constructor rules, at the opposite of *e.g.* [4] – see below. **3.** If the specification is not sufficiently complete, the procedure stops and returns as *counter-examples* the patterns along with constraints on which a function is not defined, as a hint for the rewrite rules which must be added to the system in order to make it sufficiently complete. It is also possible to learn the conditions of such a rule from the failure of the strongly ground reducibility test. **4.** Constrained tree grammars are the cement of the *integration* of our procedures for sufficient completeness verification and inductive theorem proving. Moreover, they are crucial for the completeness of the procedure. Indeed, they provide an *exact* finite representation of ground constructor terms in normal form. In comparison, the *cover sets* used *e.g.* in [17, 2] may be over-approximating (*i.e.* they may represent also some reducible terms) in presence of axioms for constructors. **5.** The method

has been successfully tested with several specifications, yielding very natural proofs where other related techniques fail.

The paper is organized as follows. In Section 2, we briefly introduce the basic concepts about constrained and conditional term rewriting. Constrained tree grammars are presented in Section 3, and Section 4 introduces sufficient completeness and the method of [3] for inductive theorem proving. The procedure for checking sufficient completeness is defined in Section 6, as well as decidable subcases, and its correctness and completeness are proved. Finally, Sections 5 and 7 present examples of applications of this procedure to specifications of respectively integers, sorted lists and powerlists.

Related work. A procedure has been proposed in [2] for checking completeness for parametrized conditional specifications. However, the completeness of this procedure assumes that the axioms for defined functions are left-linear and that there are no axioms for constructors. In [4], tree automata techniques are used to check sufficient completeness of specifications with axioms between constructors. This technique has been generalized to membership equational logic [5] in order to support partial conditional specifications with sorts and subsorts and functions domains defined by conditional memberships. The approaches of [4, 5] work by transforming the initial specification in order to get rid of rewrite rules for constructors. However, unlike us, they are limited to constructor rules which are unconstrained and *left-linear*.

Recently a more general framework has been proposed in [14] (as an extension of [5]), allowing a much wider class of MEL specifications to be checked. The system of [14] analyzes MEL specifications in the Maude language and generates a set of proof obligations which, if discharged, guarantee sufficient completeness. The proof obligations are given to the Maude's inductive theorem prover and may need user interaction (see Section 7). Note that the generated proof obligations may be not valid even when the specification is complete. In such case, a transformation of the initial specification may be needed, in order to get rid of the axioms between constructors (see Section 5). Note also that, unlike with our procedure, a failure of the method of [14] does not imply necessarily that the specification is not sufficiently complete, and if it is not, it does not provide a counter-example to help to complete the specification.

The more recent [15] generalizes the framework of [14] in several directions, allowing in particular deduction modulo axioms, and proves a decision result. This result is orthogonal to the one described at the end of Section 6 in this paper, though both rely on tree automata techniques. On one hand, the decidable case of [15] is restricted to left linear rules and sort constraints, on the other hand, this procedure works in presence of equational axioms for associativity and commutativity (AC), which are not supported by our method. We believe it would certainly be worth to study a combination of the automata modulo AC of [15] and the constrained automata used in this paper.

2 Preliminaries

The reader is assumed familiar with the basic notions of term rewriting [10] and mathematical logic. Notions and notations not defined here are standard.

Terms and substitutions. We assume given a many-sorted signature $(\mathcal{S}, \mathcal{F})$ (or simply $\mathcal{F}$, for short) where $\mathcal{S}$ is a set of *sorts* and $\mathcal{F}$ is a finite set of *function symbols*. Each symbol f is given with a profile $f : S_1 \times \ldots \times S_n \to S$ where $S_1, \ldots, S_n, S \in \mathcal{S}$ and n is the *arity* of f. We assume moreover that $\mathcal{F}$ comes in two parts, $\mathcal{F} = \mathcal{C} \uplus \mathcal{D}$ where $\mathcal{C}$ is 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 decorate variables with sort exponent like x^S in order to indicate that x has sort 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 by $\mathcal{T}(\mathcal{F})$ (resp. $\mathcal{T}(\mathcal{C})$). We assume that each sort contains at least one ground term. The sort of a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is denoted by $sort(t)$.

A term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is identified as usual to a function from its set of *positions* (strings of positive integers) $\mathcal{P}os(t)$ to symbols of $\mathcal{F}$ and $\mathcal{X}$. We use Λ to denote the empty string (root position). 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 $t[s]_p$ (p may be omitted when we just want to indicate that s is a subterm of t). We use $var(t)$ to denote the set of variables occurring in t. A term t is *linear* if every variable occurs at most once in t.

A *substitution* is a finite mapping from variables to terms. As usual, we identify substitutions with their morphism extension to terms. A *variable renaming* is a (well) sorted bijective substitution which maps variables to variables. A substitution σ is *grounding* for a term t if the domain of σ contains all the variables of t and the codomain of σ contains only ground terms. We use postfix notation for substitutions application and composition. The most general common instance of some terms $t_1, \ldots, t_n$ is denoted by $\mathrm{mgi}(t_1, \ldots, t_n)$.

Constraints for terms and clauses. We assume given a constraint language $\mathcal{L}$, which is a finite set of predicate symbols with a recursive Boolean interpretation in $\mathcal{T}(\mathcal{C})$. Typically, $\mathcal{L}$ contains the syntactic equality $. \approx .$ (syntactic disequality $. \not\approx .$), some simplification ordering $. \prec .$ like *e.g.* a lexicographic path ordering [10], and membership $x{:}L$ to a fixed regular language $L \subseteq \mathcal{T}(\mathcal{C})$ (like for instance well sorted terms). *Constraints* on the language $\mathcal{L}$ are Boolean combinations of atoms of the form $P(t_1, \ldots, t_n)$ where $P \in \mathcal{L}$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C}, \mathcal{X})$. By convention, an empty combination is interpreted as true.

We may extend the application of substitutions from terms to constraints in a straightforward way, and therefore define a solution for a constraint c as a (constructor) substitution σ grounding for all terms in c and such that $c\sigma$ is interpreted as true. The set of solutions of the constraint c is denoted by $sol(c)$. 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 $f(x,x')\,[\![c \wedge x \approx x']\!]$. A *literal* is an equation $s = t$ or an oriented equation $s \to t$ between two terms. We consider *clauses* of the form $\Gamma \Rightarrow L$ where Γ is a conjunction of literals and L is a literal. We find convenient to see clauses themselves as terms on a signature extended by the predicate symbols $=$ and $\to$, and the connective $\wedge$ and $\Rightarrow$. This way, we can define a *constrained clause* as a constrained term.

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

A term $t\,[\![d]\!]$ rewrites to $s\,[\![d]\!]$ by a rule $\rho \equiv \Gamma \Rightarrow l \to r\,[\![c]\!] \in \mathcal{R}$, denoted by $t\,[\![d]\!] \xrightarrow[\mathcal{R}]{} s\,[\![d]\!]$ if $t|_p = l\sigma$ for some position p and substitution σ, $s = t[r\sigma]_p$, the substitution σ is such that $d \wedge \neg c\sigma$ is unsatisfiable and $u\sigma \downarrow_{\mathcal{R}} v\sigma$ for all $u = v \in \Gamma$, where $u\sigma \downarrow_{\mathcal{R}} v\sigma$ stands for $\exists w,\ u \xrightarrow[\mathcal{R}]{*} w \xleftarrow[\mathcal{R}]{*} v$ and $\xrightarrow[\mathcal{R}]{*}$ (resp. $\xrightarrow[\mathcal{R}]{+}$) denotes the reflexive transitive (resp. transitive) closure of $\xrightarrow[\mathcal{R}]{}$. Note the semantic difference between conditions and constraints in rewrite rules. The validity of the condition is defined wrt the system $\mathcal{R}$ whereas the interpretation of the 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 $\mathcal{R}$-*irreducible*, or an $\mathcal{R}$-*normal form*. A constrained term $t\,[\![c]\!]$ is *weakly reducible* by $\mathcal{R}$ if it contains as a subterm an instance of a left hand side of a rule of $\mathcal{R}$. 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 definitions of *termination*, *(ground) confluence* and *(ground) convergence* are the standard ones [10].

Constructor specification. We assume from now on that every conditional constrained rewrite system $\mathcal{R}$ is of the form $\mathcal{R} = \mathcal{R}_{\mathcal{D}} \uplus \mathcal{R}_{\mathcal{C}}$ where $\mathcal{R}_{\mathcal{D}}$ contains conditional constrained rules of the form $\Gamma \Rightarrow f(\ell_1, \ldots, \ell_n) \to r\,[\![c]\!]$ such that $f \in \mathcal{D}$, $\ell_1, \ldots, \ell_n \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ and $\mathcal{R}_{\mathcal{C}}$ contains constrained rewrite rules with constructor symbols from $\mathcal{C}$ only.

3 Constrained Tree Grammars

Constrained tree grammars permit an exact representation of the set of ground terms irreducible by a given constrained rewrite system. In our approach, as well as in [3], they are used to generate incrementally a relevant set of terms, by means of non-terminal replacement following production rules.

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

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

Term generation, language. Given a constrained tree grammar $\mathcal{G} = (Q, \Delta)$, the production relation on constrained terms $\vdash_{\mathcal{G},x}$, or $\vdash_x$ or $\vdash$ for short when $\mathcal{G}$ is clear from context, is defined by:

$$t[x] \, [\![x{:}\llcorner u \lrcorner \wedge d]\!] \vdash_x t[f(x_1, \ldots, x_n)] \, [\![x_1{:}\llcorner u_1 \lrcorner \wedge \ldots \wedge x_n{:}\llcorner u_n \lrcorner \wedge c \wedge d\sigma]\!]$$

if there exists $\llcorner u \lrcorner := f(\llcorner u_1 \lrcorner, \ldots, \llcorner u_n \lrcorner) \, [\![c]\!] \in \Delta$ such that $f(u_1, \ldots, u_n) = u\sigma$ (we assume that the variables of $u_1, \ldots, u_n$ and c do not occur in the constrained term $t[x] \, [\![x{:}\llcorner u \lrcorner \wedge d]\!]$) and $x_1, \ldots, x_n$ are fresh variables. The reflexive transitive and transitive closures of the relation $\vdash$ are respectively denoted by $\vdash^*$ and $\vdash^+$.

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

The above membership constraints $t{:}\llcorner u \lrcorner$, with $\llcorner u \lrcorner \in Q$, are interpreted by: $sol(t{:}\llcorner u \lrcorner) = \{\sigma \mid t\sigma \in L(\mathcal{G}, \llcorner u \lrcorner)\}$. Note that we can use such constraint for instance to restrict a term to a given sort or any given regular tree language.

Normal form grammar. For every constrained rewrite system $\mathcal{R}_\mathcal{C}$, we can construct a constrained tree grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C}) = (Q_{\mathrm{NF}}(\mathcal{R}_\mathcal{C}), \Delta_{\mathrm{NF}}(\mathcal{R}_\mathcal{C}))$ which generates the language of ground $\mathcal{R}_\mathcal{C}$-normal forms. Intuitively, this construction, which generalizes the one of [8], corresponds to the complementation and completion of a tree grammar for $\mathcal{R}_\mathcal{C}$-reducible terms, where every subset of non-terminals (for the complementation) is represented by the mgi of its elements.

Let $\mathcal{L}(\mathcal{R}_\mathcal{C})$ be the set containing the subterms of the left hand sides of the constrained rules of $\mathcal{R}_\mathcal{C}$ and the strict subterms of the left hand sides of the unconstrained rules of $\mathcal{R}_\mathcal{C}$, and let $Q_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ be the set containing the non-terminals of the form $\llcorner x^S \lrcorner$ for each sort $S \in \mathcal{S}$ and $\llcorner \mathrm{mgi}(t_1, \ldots, t_n) \lrcorner$ where $\{t_1, \ldots, t_n\}$ is a maximal subset of $\mathcal{L}(\mathcal{R}_\mathcal{C})$ such that $t_1, \ldots, t_n$ are unifiable The set of transitions $\Delta_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ contains every $\llcorner t \lrcorner := f(\llcorner u_1 \lrcorner, \ldots, \llcorner u_n \lrcorner) \, [\![\neg c]\!]$ such that $f \in \mathcal{F}$ with profile $S_1 \times \ldots \times S_n \to S$, $\llcorner u_1 \lrcorner, \ldots, \llcorner u_n \lrcorner \in Q_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$, $u_1, \ldots, u_n$ have respective sorts $S_1, \ldots, S_n$, t is the mgi of the set: $\{u \mid \llcorner u \lrcorner \in Q_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ and u matches $f(u_1, \ldots, u_n)\}$, and $c \equiv \bigvee_{l \to r \, [\![e]\!] \in \mathcal{R}_\mathcal{C}, \; f(u_1, \ldots, u_n) = l\theta} e\theta$.

Example 1. Let Int be a sort for integers and assume a set $\mathcal{C}$ of constructor symbols containing $0 : \mathsf{Int}$ and the unary predecessor and successor symbols $p, s : \mathsf{Int} \to \mathsf{Int}$, and let $\mathcal{R}_\mathcal{C} = \{s(p(x)) \to x, \; p(s(x)) \to x\}$.

The constrained tree grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ constructed as above has three non-terminals $\llcorner s(x) \lrcorner$, $\llcorner p(x) \lrcorner$ and $\llcorner x^{\mathsf{Int}} \lrcorner$. The latter non-terminal is denoted by $\llcorner 0 \lrcorner$ below because this non-terminal only generates 0; the

two other non-terminals generate respectively the terms of the form $s^m(0)$ and $p^m(0)$ with $m > 0$. The production rules of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ are[3]: $\llcorner 0 \lrcorner := 0$ $\quad \llcorner s(x) \lrcorner := s(\llcorner 0 \lrcorner)$ $\quad \llcorner p(x) \lrcorner := p(\llcorner 0 \lrcorner)$ $\quad \llcorner s(x) \lrcorner := s\big(\llcorner s(x) \lrcorner\big)$ $\quad \llcorner p(x) \lrcorner := p\big(\llcorner p(x) \lrcorner\big)$ $\diamond$

The proof of the following lemma can be found in [3].

Lemma 1. *The language* $\bigcup_{\llcorner u \lrcorner \in Q_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})} L(\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C}), \llcorner u \lrcorner)$ *is the set of* $\mathcal{R}_\mathcal{C}$*-irreducible terms of* $\mathcal{T}(\mathcal{C})$.

We shall consider below the normal form grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ associated to $\mathcal{R}_\mathcal{C}$ and we call a constrained term $t\,[\![c]\!]$ *decorated* if $c = x_1{:}\llcorner u_1 \lrcorner \wedge \ldots \wedge x_n{:}\llcorner u_n \lrcorner \wedge d$, $\{x_1, \ldots, x_n\} = var(t)$, $\llcorner u_i \lrcorner \in Q_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ and $sort(u_i) = sort(x_i)$ for all $i \in [1..n]$.

4 Sufficient Completeness and Automated Induction

We shall now define formally the problem we are concerned with in this paper and its relations with inductive theorem proving.

Definition 3. *The TRS* $\mathcal{R}$ *is sufficiently complete iff for all* $t \in \mathcal{T}(\mathcal{F})$ *there exists* s *in* $\mathcal{T}(\mathcal{C})$ *such that* $t \xrightarrow[\mathcal{R}]{*} s$.

The procedure we are proposing in Section 6 for checking sufficient completeness is based on an equivalent definition using the notion of sufficient completeness of the defined function symbols.

Definition 4. *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* $s \in \mathcal{T}(\mathcal{C})$ *such that* $f(t_1, \ldots, t_n) \xrightarrow[\mathcal{R}]{+} s$.

Property 1. A TRS $\mathcal{R}$ is sufficiently complete iff every defined symbol $f \in \mathcal{D}$ is sufficiently complete wrt $\mathcal{R}$.

Proof. By induction on the structure of a given term $t \in \mathcal{T}(\mathcal{F})$. $\square$

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

Definition 5. *A constrained clause* $C\,[\![c]\!]$ *is an* inductive theorem *of* $\mathcal{R}$ *(denoted by* $\mathcal{R} \models_{\mathcal{I}nd} C\,[\![c]\!]$*) if for all substitutions* $\sigma \in sol(c)$ *we have* $\mathcal{R} \models C\sigma$.

[3] We use a simplified notation for production rules, for the sake of readability.

Inductive theorem proving with constrained tree grammars. In [3] we develop a new approach for automated inductive theorem proving for the same kind of TRS specifications as in this paper. The procedure of [3] is also based on the normal form constrained tree grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$, which is used for the generation of subgoals during the proof by induction. Therefore, it can be called to discharge proof obligations during the procedure for checking sufficient completeness defined in Section 6. Note that the procedure of [3] is sound and refutationally complete for the decorated conjectures that we shall consider here (see Theorem 2 and corollary 2 in [3]). Because of length restrictions, we cannot present in detail the procedure of [3] here. Let us illustrate its principle on an example.

Example 2. We complete the specification of Example 1 with a sort Bool for Booleans, two constants of $\mathcal{C}$, $\mathit{true}, \mathit{false} : \mathsf{Bool}$ and one binary defined symbol $\leq\colon \mathsf{Int} \times \mathsf{Int} \to \mathsf{Bool}$ in $\mathcal{D}$. Let $\mathcal{R}_{\mathcal{D}}$ be the following set of conditional rules:

$$\begin{array}{lll} 0 \leq 0 \to \mathit{true}, & s(x) \leq y \to x \leq p(y), & 0 \leq x = \mathit{true} \Rightarrow 0 \leq s(x) \to \mathit{true}, \\ 0 \leq p(0) \to \mathit{false}, & p(x) \leq y \to x \leq s(y), & 0 \leq x = \mathit{false} \Rightarrow 0 \leq p(x) \to \mathit{false}. \end{array}$$

We show that the following clause is an inductive theorem of $\mathcal{R}$:

$$0 \leq x_1 = \mathit{true} \, [\![x_1 \colon\!\lfloor s(x) \rfloor]\!] \tag{1}$$

Applying the production rules of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ to (1), we obtain two subgoals (induction step): $0 \leq s(x_1) = \mathit{true} \, [\![x_1 \colon\!\lfloor 0 \rfloor]\!]$ and $0 \leq s(x_1) = \mathit{true} \, [\![x_1 \colon\!\lfloor s(x) \rfloor]\!]$. The first subgoal can be further instantiated by $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ into $0 \leq s(0) = \mathit{true}$, and this equation rewrites by $\mathcal{R}_{\mathcal{D}}$ into the tautology $\mathit{true} = \mathit{true}$. The second subgoal can be simplified into the tautology $\mathit{true} = \mathit{true}$ using the clause (1), which, in this case, is considered as an induction hypothesis. It is possible because (1) is strictly smaller[4] than the second subgoal. ◇

Strong ground irreducibility. A crucial problem in the incremental procedure of Section 6 is to detect whether all the ground instances of a term in construction are $\mathcal{R}_{\mathcal{D}}$-reducible. For this purpose, we use the following sufficient condition for ground reducibility, based on the notion of inductive theorem.

Definition 6. *A constrained term $t \, [\![c]\!]$ is* strongly ground reducible *wrt $\mathcal{R}$ if there exist n rules of $\mathcal{R}_{\mathcal{D}}$ and n substitutions with $n > 0$, (the rules are denoted $\Gamma_i \Rightarrow l_i \to r_i \, [\![c_i]\!]$ and the substitutions σ_i, with $i \in [1..n]$) such that $t = l_i\sigma_i$ for all $i \in [1..n]$, $\neg c \vee c_1\sigma_1 \vee \ldots \vee c_n\sigma_n$ is satisfiable and $\mathcal{R} \models_{\mathcal{I}nd} \Gamma_1\sigma_1 \, [\![c \wedge c_1\sigma_1]\!] \vee \ldots \vee \Gamma_n\sigma_n \, [\![c \wedge c_n\sigma_n]\!]$.*

Note that by definition, every strongly ground reducible constrained term is weakly reducible. Moreover, when $\mathcal{R}$ is ground convergent, every strongly ground reducible constrained term is ground reducible. This is shown in the proof of Theorem 1. The converse is not true.

[4] *smaller* is meant here wrt a well–founded ordering on constrained clauses, see [3] for a formal definition.

5 Example: Integers

Let us continue with Examples 1 and 2. Since $\mathcal{R}$ is ground convergent, in order to check the sufficient completeness of the symbol $\leq$ wrt $\mathcal{R}$, it is sufficient to consider the reductions (under $\mathcal{R} = \mathcal{R}_\mathcal{C} \uplus \mathcal{R}_\mathcal{D}$) of the terms of the form $t_1 \leq t_2$ where t_1 and t_2 are $\mathcal{R}_\mathcal{C}$-irreducible terms of $\mathcal{T}(\mathcal{C})$. By Lemma 1, they are produced by $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ starting from terms of the form $x_1 \leq x_2 \, [\![x_1{:}n_1 \wedge x_2{:}n_2]\!]$ where n_1 and n_2 are non-terminal of $Q_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$. For the sake of readability, we shall denote such a term $n_1 \leq n_2$ in the pattern tree[5] of Figure 1 and the case analysis below.

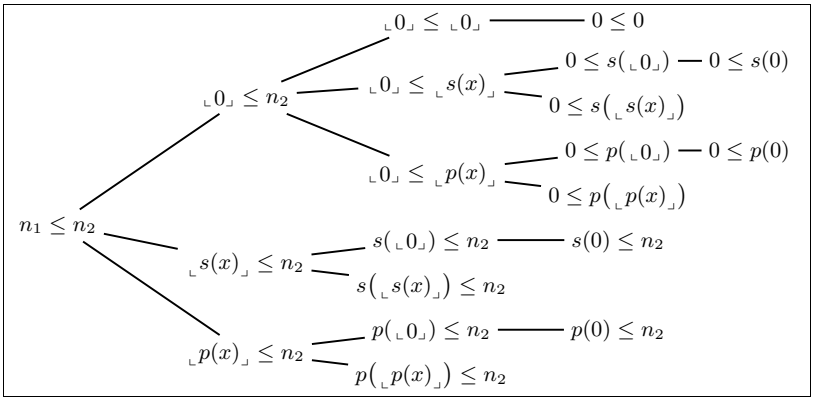

Fig. 1. Sufficient completeness of $\leq$

- $\llcorner 0 \lrcorner \leq \llcorner 0 \lrcorner$ is instantiated by $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ into $0 \leq 0$ which is reducible by $\mathcal{R}_\mathcal{D}$.
- $\llcorner 0 \lrcorner \leq \llcorner s(x) \lrcorner$ is instantiated into $\llcorner 0 \lrcorner \leq s(\llcorner 0 \lrcorner)$ and $\llcorner 0 \lrcorner \leq s(\llcorner s(x) \lrcorner)$. The first term is further instantiated into $0 \leq s(0)$, which is reducible into *true* by $\mathcal{R}_\mathcal{D}$. The second term is instantiated into $0 \leq s(\llcorner s(x) \lrcorner)$, which is strongly ground reducible since $\mathcal{R} \models_{\mathcal{I}nd} 0 \leq x_2 = \mathit{true} \, [\![x_2{:}\llcorner s(x) \lrcorner]\!]$ (see Example 2).
- $\llcorner 0 \lrcorner \leq \llcorner p(x) \lrcorner$: similarly, with $\mathcal{R} \models_{\mathcal{I}nd} 0 \leq x_2 = \mathit{false} \, [\![x_2{:}\llcorner p(x) \lrcorner]\!]$.
- $\llcorner s(x) \lrcorner \leq n_2$ (whatever n_2) is instantiated into $s(\llcorner 0 \lrcorner) \leq n_2$ and $s(\llcorner s(x) \lrcorner) \leq n_2$. Both are instances of the left hand side of an unconditional rule of $\mathcal{R}_\mathcal{D}$.
- $\llcorner p(x) \lrcorner \leq n_2$: the situation is similar.

The proof of the completeness of $\leq$ fails with the method of [2]. Indeed, the following cover set for the sort Int: $\{0, s(x), p(x)\}$ is not relevant because, it does not describe exactly the set of $\mathcal{R}$-irreducible ground constructor terms. For

[5] The tree of Figure 1 does not *stricto sensu* adhere to the definition of Section 6. Two nodes have been added for illustration purposes.

instance $p(s(0))$ is an instance of $p(x)$ but is not irreducible. The methods of [4, 5] can be used for checking the sufficient completeness of $\leq$ since the axioms for constructors are unconstrained and *left-linear*. However, we recall that these procedures do not work directly on the given specification but transform it in order to get rid of the axioms between constructors.

With a direct translation of the above integer specification in Maude syntax, the Maude sufficient completeness checker [14] generates one proof obligation which is not valid[6]. It is possible to prove the sufficient completeness of this specification with [14] using a transformation it into a new specification with free constructors by specifying subsorts for zero, positive and negative integers respectively. A second example with a specification of integers modulo 2 is presented in the long version of this paper, Appendix B, for the interested reader.

6 Verification of Sufficient Completeness

We describe in this section a complete procedure for sufficient completeness verification. It relies on the framework of [3] presented in Section 4.

Pattern trees. The procedure checks the sufficient completeness of each defined symbol $f \in \mathcal{D}$ by the incremental construction of a multi-rooted pattern tree called *pattern tree* of f and denoted by $dtree(f)$. The nodes of $dtree(f)$ are labelled by decorated constrained terms of the form $f(t_1, \ldots, t_n)\,[\![c]\!]$ such that $t_i \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ for every $i \in [1..n]$. Each root of $dtree(f)$ is labelled by a decorated term $f(x_1, \ldots, x_n)\,[\![x_1{:}\llcorner u_{1\lrcorner} \wedge \ldots \wedge x_n{:}\llcorner u_{n\lrcorner}]\!]$ where $x_1, \ldots, x_n$ are distinct variables and $\llcorner u_{1\lrcorner}, \ldots, \llcorner u_{n\lrcorner} \in Q_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$. The successors of any internal node of $dtree(f)$ are determined by the inference rules described in Figure 2.

Inference rules for sufficient completeness. The algorithm presented in Figure 2 for the construction of $dtree(f)$ operates incrementally. It follows the production rules of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ for non-terminal replacement in decorated term labelling the leaves of the tree constructed so far, until the term obtained becomes strongly ground reducible. In order to ensure the termination of the algorithm, the replacements are limited to variables, called *induction variables* whose instantiation is needed in order to trigger a rewrite step.

Definition 7. *The set* $i\mathcal{P}os(f, \mathcal{R})$ *of* induction positions *of* $f \in \mathcal{D}$ *is the set of non-root and non-variable positions of left-hand sides of rules of* $\mathcal{R}_{\mathcal{D}}$ *with the symbol* f *at the root position. The set* $i\mathcal{V}ar(t)$ *of* induction variables *of* $t = f(t_1, \ldots, t_n)$*, with* $f \in \mathcal{D}$ *and* $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C}, \mathcal{X})$*, is the subset of variables of* $var(t)$ *occurring in* t *at positions of* $i\mathcal{P}os(f, \mathcal{R})$ *.*

Intuitively, it is sufficient to consider only induction variables for the application of the production rules of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$, because any ground instance of a term labelling a node in $dtree(f)$ may be only reduced by $\mathcal{R}$ at the root position. Let us now describe the inference rules of Figure 2 in a little more detail.

[6] Joe Hendrix, personal communication.

Instantiation:	$\dfrac{t\,[\![c]\!]}{t'\,[\![c']\!]}$	if $t\,[\![c]\!]$ is not strongly ground reducible where $x \in i\mathcal{V}ar(t\,[\![c]\!])$ and $t\,[\![c]\!] \vdash_x t'\,[\![c']\!]$
Strongly Ground Reducible Leaf:	$\dfrac{t\,[\![c]\!]}{success}$	if $t\,[\![c]\!]$ is strongly ground reducible
Irreducible Leaf:	$\dfrac{t\,[\![c]\!]}{failure(t\,[\![c]\!])}$	if no other rule applies to $t\,[\![c]\!]$

Fig. 2. Inference rules for the construction of a pattern tree

Instantiation applies the production rules of the normal-form grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ to induction variables in the decorated term $t\,[\![c]\!]$.

Strongly Ground Reducible Leaf, following Definition 6, checks, for all rules whose left hand side matches t, the satisfiability of some constraints (this problem is assumed decidable for the constraints considered) and inductive theorem proving, using the procedure of [3] (which is also based on $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$).

Irreducible Leaf produces a failure when none of the two above inferences applies to a leaf $t\,[\![c]\!]$. It means in this case that the symbol f is not sufficiently complete wrt $\mathcal{R}$. The term $t\,[\![c]\!]$ provides an hint on the rule (exactly the left hand side and the constraint of this rule) which must be added to $\mathcal{R}$ in order to complete the specification of f. It is also possible to learn the conditions of such a rule from the failure of the strongly ground reducibility test.

Termination, correctness, completeness. The complete proofs of the following theorems can be found in the long version, Appendix A.

Theorem 1 (Soundness). *Assume that $\mathcal{R}$ is ground convergent. If for all $f \in \mathcal{D}$, all leaves of $dtree(f)$ are success then $\mathcal{R}$ is sufficiently complete.*

The key point of the proof (long version, Appendix A) is that every ground term of the form $f(t_1, \ldots, t_n)$ with $f \in \mathcal{D}$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C})$ is generated by $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ starting from a term labelling a leaf of $dtree(f)$ and hence is $\mathcal{R}_{\mathcal{D}}$-reducible.

As a corollary, since there are only two kinds of leaves, we can conclude that if $\mathcal{R}$ is not sufficiently complete, then the inference system will end with a failure.

Corollary 1 (Refutational Completeness). *Assume that $\mathcal{R}$ is ground convergent. If $\mathcal{R}$ is not sufficiently complete, then there exists $f \in \mathcal{D}$ such that $dtree(f)$ contains a leaf of the form failure.*

Theorem 2 (Completeness). *If $\mathcal{R}$ is sufficiently complete then for each $f \in \mathcal{D}$, all leaves of $dtree(f)$ are success.*

We show (long version, Appendix A) that the existence of a non-strongly ground reducible term in a leaf of $dtree(f)$ contradicts the sufficient completeness of $\mathcal{R}$. As a corollary, we can conclude that if the inference system fails then $\mathcal{R}$ is not sufficiently complete.

Corollary 2 (Soundness of Disproof). *For each $f \in \mathcal{D}$, if there exists a leaf of the form failure in $dtree(f)$ then f is not sufficiently complete wrt $\mathcal{R}$.*

Theorem 3 (Finiteness of pattern trees). *For every $f \in \mathcal{D}$, the size of $dtree(f)$ is bounded.*

It follows from the finiteness of $i\mathcal{P}os(f, \mathcal{R})$ (long version, Appendix A). Note that the finiteness of the pattern trees does not guarantee the termination of the procedure in general, since it relies on the test of strongly ground reducibility.

Decidable case. Strongly ground reducibility, like sufficient completeness and inductive theorem proving is undecidable in general. When $\mathcal{R}$ is unconditional (but constrained), testing strongly ground reducibility (Definition 6) of a constrained term in a pattern tree amounts to pattern matching with left-hand-side of rules of $\mathcal{R}_{\mathcal{D}}$ and checking satisfiability of constraints. Altogether, this problem (strongly ground reducibility) is reducible to emptiness decision for constrained tree grammars (the problem of deciding whether the language of a given grammar is empty or not). We will not detail the reduction here because it is already given, in a similar context, in [3] (Section 6).

Theorem 4. *Sufficient completeness is decidable when $\mathcal{R}$ is unconditional and ground convergent, contains only constraints of equality, disequality, and membership to a regular tree language, and when moreover, for all $l \to r \,[\![c]\!] \in \mathcal{R}_{\mathcal{C}}$, for all $s \approx s' \in c$, each of s and s' is either a variable or a strict subterm of l, and for all $s \not\approx s' \in c$, there is a subterm of l of the form $g(\ldots, s, \ldots, s', \ldots)$.*

The restriction on the constraints correspond to known classes of tree automata with equality and disequality constraints with a decidable emptiness problem (see [7] for a survey). The membership constraints can be treated with a classical product construction.

7 More Examples: Sorted Lists and Powerlists

We consider now a specification of sorted lists without repetition, with the constructor symbols $true, false : \mathsf{Bool}$, $0 : \mathsf{Nat}$, $s : \mathsf{Nat} \to \mathsf{Nat}$, $\emptyset : \mathsf{List}$, $ins : \mathsf{Nat} \times \mathsf{List} \to \mathsf{List}$, and assume two constrained constructor rules in $\mathcal{R}_{\mathcal{C}}$:

$$ins(x, ins(y, z)) \to ins(x, z) \,[\![x \approx y]\!],$$
$$ins(x, ins(y, z)) \to ins(y, ins(x, z)) \,[\![x \succ y]\!]$$

The ordering constraint $\succ$ is interpreted as a reduction ordering total on ground constructor terms. Note that $\mathcal{R}_{\mathcal{C}}$ is terminating thanks to the constraint of the second rule.

Let us complete this signature with the following defined function symbols in $\mathcal{D}$: $\in, \in', co : \mathsf{Nat} \times \mathsf{List} \to \mathsf{Bool}$, $sorted : \mathsf{List} \to \mathsf{Bool}$, and the rules of $\mathcal{R}_\mathcal{D}$:

$$\begin{array}{cc} x \in \emptyset \to \mathit{false} & x \in' \emptyset \to \mathit{false} \\ x_1 \in \mathit{ins}(x_2, y) \to \mathit{true}\,[\![x_1 \approx x_2]\!] & x_1 \in' \mathit{ins}(x_2, y) \to \mathit{true}\,[\![x_1 \approx x_2]\!] \\ x_1 \in \mathit{ins}(x_2, y) \to x_1 \in y\,[\![x_1 \not\approx x_2]\!] & x_1 \in' \mathit{ins}(x_2, y) \to \mathit{false}\,[\![x_1 \prec x_2]\!] \\ x \in' y = x \in y \Rightarrow \mathit{co}(x, y) \to \mathit{true} & x_1 \in' \mathit{ins}(x_2, y) \to x_1 \in' y\,[\![x_1 \succ x_2]\!] \\ \multicolumn{2}{c}{\mathit{sorted}(\mathit{ins}(y, z)) = \mathit{true} \Rightarrow \mathit{sorted}(\mathit{ins}(x, \mathit{ins}(y, z))) \to \mathit{true}\,[\![x \prec y]\!]} \end{array}$$

The constrained tree grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ has the following non-terminals: $Q_{\mathrm{NF}}(\mathcal{R}_\mathcal{C}) = \{ \llcorner x^{\mathsf{Bool}} \lrcorner, \llcorner x^{\mathsf{Nat}} \lrcorner, \llcorner x^{\mathsf{List}} \lrcorner, \llcorner \mathit{ins}(x_1, y_1) \lrcorner \}$, ($\llcorner x^{\mathsf{List}} \lrcorner$ is denoted by $\llcorner \emptyset \lrcorner$ below) and the following set of production rules $\Delta_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$:

$$\begin{array}{l} \llcorner x^{\mathsf{Bool}} \lrcorner := \mathit{true} \mid \mathit{false} \quad \llcorner x^{\mathsf{Nat}} \lrcorner := 0 \mid s(\llcorner x_2^{\mathsf{Nat}} \lrcorner) \quad \llcorner \emptyset \lrcorner := \emptyset, \\ \llcorner \mathit{ins}(x_1, y_1) \lrcorner := \mathit{ins}(\llcorner x^{\mathsf{Nat}} \lrcorner, \llcorner \emptyset \lrcorner) \\ \llcorner \mathit{ins}(x_1, y_1) \lrcorner := \mathit{ins}(\llcorner x^{\mathsf{Nat}} \lrcorner, \llcorner \mathit{ins}(x_2, y_2) \lrcorner)\,[\![x^{\mathsf{Nat}} \prec x_2]\!] \end{array}$$

The pattern tree $\mathit{dtree}(\in)$ described in Figure 3 shows that the function $\in$ is

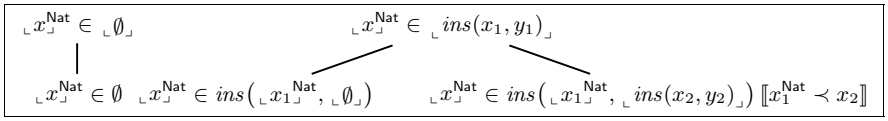

Fig. 3. The pattern tree of $\in$

sufficiently complete. The term $x^{\mathsf{Nat}} \in \emptyset$ is indeed reducible. The two other leaves are also strongly ground reducible since $x_1 \approx x_2 \vee x_1 \not\approx x_2$ is satisfiable. The proof of sufficient completeness of $\in'$ is very similar. In order to prove that the function co is sufficiently complete, we need to show that $\mathcal{R} \models_{\mathcal{I}nd} x \in' y = x \in y$. This theorem can be proved using the method of [3] without any user interaction (see long version, Appendix D).

The pattern tree $\mathit{dtree}(\mathit{sorted})$, (described in Figure 4, Appendix C) contains two *failure* leaves labeled respectively with $\mathit{failure}\big(\mathit{sorted}(\emptyset)\big)$ and $\mathit{failure}\big(\mathit{sorted}(\mathit{ins}(x^{\mathsf{Nat}}, \emptyset))\big)$. The reason is that these terms do not contain induction variables and they are not strongly ground reducible because they do not match a left-hand-side of rule of $\mathcal{R}_\mathcal{D}$. This suggests to complete $\mathcal{R}_\mathcal{D}$ with two rules $\mathit{sorted}(\emptyset) \to \mathit{true}$ and $\mathit{sorted}(\mathit{ins}(x, \emptyset)) \to \mathit{true}$. It is shown in the long version, Appendix C, that the system obtained is sufficiently complete.

A last example, the specification of Misra's powerlists, is presented in the long version, Appendix E.

The methods of [2, 4, 5] cannot be applied to prove the sufficient completeness of $\in$, $\in'$, co and $sorted$ since the axioms for constructors are constrained and *non-left-linear*. We could imagine a straightforward adaptation of

the methods based on cover sets to *constrained cover sets* for sorted lists, like $\{\emptyset, ins(x, \emptyset), ins(x, ins(y, z))\, [\![x < y]\!]\}$. This also fails. The reason is that this representation of $\mathcal{R}_{\mathcal{C}}$-irreducible ground constructor terms is still not exact. For example $ins(0, ins(s(0), ins(0, \emptyset))$ is an instance of $ins(x, ins(y, z))\, [\![x < y]\!]$ but is not irreducible. The Maude sufficient completeness checker has been successfully used for the powerlists [14]. For checking the sufficient completeness of *co*, it generates a proof obligation which cannot be proved automatically by the Maude's inductive theorem prover and therefore must be manually discharged by the user[7].

8 Conclusion

We have proposed a method for testing sufficient completeness of constrained and conditional rewrite systems with constrained rules for constructors. Our procedure uses a tree grammar with constraints which generates the set of ground constructor terms in normal form and is integrated with a method for inductive theorem proving based on the same framework [3]. It is sound for ground convergent TRS and also complete, and has been successfully used manually for checking sufficient completeness of several specifications where related techniques fail. Moreover, in case of disproof, *i.e.* when the specification is not sufficiently complete, our procedure proposes candidates left hand sides and constraints and a hint for conditions of rewrite rules to complete it.

We are planning to implement the procedure presented on this paper, based on a forthcoming system for [3] generalizing Spike [6] and on an efficient library for tree automata with constraints.

At last, following Theorem 1, ground convergence is necessary for the soundness of our method. This property is difficult to establish, especially for conditional constrained rewrite systems, and we are currently developing a technique for checking it using constrained tree grammars in the same framework as in this paper and [3].

Acknowledgements. We wish to kindly thank Joe Hendrix for having processed the above examples with the *Sufficient Completeness Checker* for Maude.

References

1. B. Bogaert and S. Tison. Equality and disequality constraints on direct subterms in tree automata. In *9th Symp. on Theoretical Aspects of Computer Science, STACS*, volume 577 of *LNCS*, pages 161–171. Springer, 1992.
2. A. Bouhoula. Using induction and rewriting to verify and complete parameterized specifications. *Theoretical Computer Science*, 170(1-2):245–276, 1996.
3. A. Bouhoula and F. Jacquemard. Automated induction for complex data structures. Research Report LSV-05-11, Laboratoire Spécification et Vérification, 2005.

[7] personal communication of Joe Hendrix

4. A. Bouhoula and J.-P. Jouannaud. Automata-driven automated induction. *Information and Computation*, 169(1):1–22, 2001.
5. A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. *Theoretical Computer Science*, 236(1-2):35–132, 2000.
6. A. Bouhoula and M. Rusinowitch. Implicit induction in conditional theories. *Journal of Automated Reasoning*, 14(2):189–235, 1995.
7. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. `http://www.grappa.univ-lille3.fr/tata`, 2002.
8. H. Comon and F. Jacquemard. Ground reducibility is exptime-complete. *Information and Computation*, 187(1):123–153, 2003.
9. H. Comon and V. Cortier. Tree automata with one memory set constraints and cryptographic protocols. Theoretical Computer Science 331(1): 143-214, 2005.
10. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, pages 243–320. 1990.
11. J. V. Guttag. *The Specification and Application to Programming of Abstract Data Types*. Phd thesis, University of Toronto, Computer Science Department, 1975. Report CSRG-59.
12. J. V. Guttag. Notes on type abstraction. In Friedrich L. Bauer and Manfred Broy, editors, *Program Construction*, volume 69 of LNCS, pages 593–616. Springer, 1978.
13. J. V. Guttag and J. J. Horning. The algebraic specification of abstract data types. *Acta Inf.*, 10:27–52, 1978.
14. J. Hendrix, M. Clavel, and J. Meseguer. A sufficient completeness reasoning tool for partial specifications. In *Proc. ot the 16th Int. Conf. on Term Rewriting and Applications (RTA)*, vol. 3467 of LNCS, pages 165–174. Springer, 2005.
15. J. Hendrix, H. Ohsaki, and J. Meseguer. Sufficient completeness checking with propositional tree automata. Technical Report UIUCDCS-R-2005-2635 (CS), UILU-ENG-2005-1825 (ENGR), University of Illinois at Urbana-Champaign, 2005.
16. G. Huet and J.-M. Hullot. Proofs by induction in equational theories with constructors. *J. Comput. Syst. Sci.*, 25(2):239–266, 1982.
17. D. Kapur. An automated tool for analyzing completeness of equational specifications. In *Proc. of the Int. Symposium on Software Testing and Analysis (ISSTA)*, special issue of *Software Engineering Notes*, pages 28–43. ACM Press, 1994.
18. D. Kapur, P. Narendran, D. J. Rosenkrantz, and H. Zhang. Sufficient-completeness, ground-reducibility and their complexity. *Acta Informatica*, 28(4):311–350, 1991.
19. D. Kapur, P. Narendran, and H. Zhang. On sufficient-completeness and related properties of term rewriting systems. *Acta Informatica*, 24(4):395–415, 1987.
20. E. Kounalis. Completeness in data type specifications. In B. F. Caviness, editor, *European Conference on Computer Algebra (2)*, volume 204 of LNCS, pages 348–362. Springer, 1985.
21. A. Lazrek, P. Lescanne, and J.-J. Thiel. Tools for proving inductive equalities, relative completeness, and omega-completeness. *Information and Computation*, 84(1):47–70, 1990.
22. J. Misra. Powerlist: A structure for parallel recursion. *ACM Transactions on Programming Languages and Systems*, 16(6):1737–1767, 1994.

Appendix A Proofs of theorems

The proof of the soundness theorem (Theorem 1) uses the following lemma.

Lemma 2. *If $\mathcal{R}$ is ground convergent then $\mathcal{R}_\mathcal{C}$ is ground convergent.*

Theorem 1. Assume that $\mathcal{R}$ is ground convergent. For each $f \in \mathcal{D}$, if all leaves of $dtree(f)$ are *success* then f is sufficiently complete wrt $\mathcal{R}$.

Proof. Assume that for all $f \in \mathcal{D}$ all the leaves of $dtree(f)$ are labeled with *success*. We prove that for all $f(t_1, \ldots, t_m)$ with $f \in \mathcal{D}$ of arity m and $t_1, \ldots, t_m \in \mathcal{T}(\mathcal{C})$, there exists $s \in \mathcal{T}(\mathcal{C})$ such that $f(t_1, \ldots, t_m) \xrightarrow[\mathcal{R}]{}{}^{*} s$. Since, by hypothesis, $\mathcal{R}_\mathcal{C}$ is terminating, we may consider that $t_1 \ldots, t_m$ are $\mathcal{R}_\mathcal{C}$-irreducible (otherwise, they can be normalized under $\xrightarrow[\mathcal{R}_\mathcal{C}]{}$). By Lemma 1, it implies that there exist some non-terminals ${}_\llcorner u_{1\lrcorner}, \ldots, {}_\llcorner u_{m\lrcorner}$ of the grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ such that:

$$f(x_1, \ldots, x_m) \, [\![x_1 : {}_\llcorner u_{1\lrcorner}, \ldots, x_m : {}_\llcorner u_{m\lrcorner}]\!] \vdash^{*} f(t_1, \ldots, t_m) \, [\![c]\!] \tag{2}$$

Note that the first term of the above derivation labels a root node of the pattern tree $dtree(f)$. We proceed by an induction based on the transitive closure of the union of $\xrightarrow[\mathcal{R}]{}$ (it is a well-founded relation by hypothesis) and the subterm relation.

Assume that t is $\mathcal{R}$-irreducible (base case of the induction). Let $s \, [\![d]\!]$ be the first term without induction variables occurring in the above grammar derivation (2), and let τ be the ground substitution of $sol(d)$ such that $s\tau = t$ (τ exists by Lemma 1). We shall show that $s \, [\![d]\!]$ is not strongly ground reducible by $\mathcal{R}$. It implies that the inference Irreducible leaf applies and $dtree(f)$ contains a leaf labeled with **failure**, a contradiction. Indeed, otherwise, by definition, there exist n rules (with $n > 0$) of $\mathcal{R}_\mathcal{D}$ $\Gamma_i \Rightarrow l_i \to r_i \, [\![c_i]\!]$, with $i \in [1..n]$, and n substitutions σ_i, such that $s = l_i\sigma_i$ and $d \wedge \neg c_i\sigma_i$ is unsatisfiable for all $i \in [1..n]$ and $\mathcal{R} \models_{\mathcal{I}nd} \Gamma_1\sigma_1 \, [\![d \wedge c_1\sigma_1]\!] \vee \ldots \vee \Gamma_n\sigma_n \, [\![d \wedge c_n\sigma_n]\!]$.

Since $\tau \in sol(d)$, then for all $i \in [1..n]$, $\tau \in sol(c_i\sigma_i)$ (otherwise, $d \wedge \neg c_i\sigma_i$ would be satisfiable). Therefore, there exists $k \in [1..n]$, such that $\mathcal{R} \models \Gamma_k\sigma_k\tau$. This implies that for each equation $u = v$ in $\Gamma_k\sigma_k\tau$, we have $u \downarrow_\mathcal{R} v$ because $\mathcal{R}$ is ground confluent, hence t can be rewritten by $\Gamma_k \Rightarrow l_k \to r_k \, [\![c_k]\!]$, a contradiction.

Hence t is $\mathcal{R}$-reducible, say $t \xrightarrow[\mathcal{R}]{} t'$. If $t' \in \mathcal{T}(\mathcal{C})$, then we are done. Otherwise, we can apply the induction hypothesis to every minimal (w.r.t. the subterm ordering) subterm of t' headed by a defined symbol. □

Theorem 2. If $\mathcal{R}$ is sufficiently complete then for each $f \in \mathcal{D}$, all leaves of $dtree(f)$ are *success*.

Proof. Assume that $\mathcal{R}$ is sufficiently complete and suppose that there exists a node $t \, [\![c]\!]$ in $dtree(f)$, for some $f \in \mathcal{D}$, to which the inference Irreducible Leaf can be applied. It means, by definition, that $t \, [\![c]\!]$ does not contain any induction variable and is not strongly ground reducible. We show first that $t \, [\![c]\!]$ is weakly reducible by $\mathcal{R}$.

By construction, $t\,[\![c]\!]$ is decorated, and since $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ is clean, there exists $\tau \in sol(c)$ such that for all $x \in var(t)$, $x\tau$ is $\mathcal{R}$-irreducible. Moreover, by construction, $t\tau$ has the form $f(t_1, \ldots, t_n)$ where $f \in \mathcal{D}$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C})$ are all $\mathcal{R}_{\mathcal{C}}$-irreducible. Hence, $t\tau$ is $\mathcal{R}_{\mathcal{D}}$-reducible at root position because $\mathcal{R}$ is sufficiently complete. Therefore, by definition, $t\tau$ is a ground instance of some left-hand side ℓ of a rule $\Gamma \Rightarrow \ell \to r\,[\![c']\!] \in \mathcal{R}_{\mathcal{D}}$, say $t\tau = \ell\theta$. Since by hypothesis t does not contain any induction variable and by definition ℓ is linear, t is an instance of ℓ, say $t = \ell\sigma$, with $\theta = \sigma\tau$ (by definition of induction variables). Hence the following subset $\mathcal{L}$ of $\mathcal{R}_{\mathcal{D}}$ is not empty:

$$\mathcal{L} = \left\{ \Gamma_i \Rightarrow \ell_i \to r_i\,[\![c_i]\!] \;\middle|\; i \in [1..n],\ t = l_i\sigma_i \right\}$$

By hypothesis, $t\,[\![c]\!]$ is not strongly ground reducible by $\mathcal{R}$. It means that one at least of the following properties holds:

$$c \wedge \neg c_1\sigma_1 \wedge \ldots \wedge \neg c_n\sigma_n \text{ is satisfiable} \tag{3}$$

$$\mathcal{R} \not\models_{\mathcal{I}nd} \Gamma_1\sigma_1\,[\![c \wedge c_1\sigma_1]\!] \vee \ldots \vee \Gamma_n\sigma_n\,[\![c \wedge c_n\sigma_n]\!] \tag{4}$$

Assume that (3) is true and let $\delta \in sol(c \wedge \neg c_1\sigma_1 \wedge \ldots \wedge \neg c_n\sigma_n)$. The term $t\delta$ is not reducible at the root position by definition of reducibility.

Assume that (4) is true. For all $k \in [1..n]$ and all ground substitution $\delta \in sol(c \wedge c_k\sigma_k)$, we have $\mathcal{R} \not\models \Gamma_k\sigma_k\delta$. Hence $t\delta$ is not reducible at the root position by a rule of $\mathcal{L}$.

Assume now we are in one of the above case and $t\delta$ is reducible at the root position by a rule $\Gamma \Rightarrow \ell \to r\,[\![d]\!] \in \mathcal{R} \setminus \mathcal{L}$. It means that t is an instance of ℓ, which contradicts the hypothesis that the above rule is not in $\mathcal{L}$.

In conclusion, in all cases, $t\delta$ is not reducible at the root position. But by construction, $t\delta$ has the form $f(s_1, \ldots, s_n)$ where $f \in \mathcal{D}$ and $s_1, \ldots, s_n \in \mathcal{T}(\mathcal{C})$ and are all $\mathcal{R}_{\mathcal{C}}$-irreducible. This contradicts the hypothesis that $\mathcal{R}$ is sufficiently complete. □

Theorem 3. For every $f \in \mathcal{D}$, the size of $dtree(f)$ is bounded.

Proof. The number of rules of $\mathcal{R}_{\mathcal{D}}$ with the function symbol f at the top position is finite. It means that the set $i\mathcal{P}os(f, \mathcal{R})$ is finite too. As a consequence, the size of non-ground terms with induction variables is also bounded, and the height of the pattern tree too, since consecutive grafts in the same branch of the tree are labeled with deeper non-ground constrained terms. □

Appendix B Example: integers modulo

Consider a sort Nat for natural numbers modulo two, with the constructor symbols of $\mathcal{C}$ $0 : \mathsf{Nat}$ and $s : \mathsf{Nat} \to \mathsf{Nat}$, one defined symbol $+$ in $\mathcal{D}$, and let: $\mathcal{R}_{\mathcal{C}} = \{s(s(x)) \to x\,[\![x \approx 0]\!]\}$ and $\mathcal{R}_{\mathcal{D}} = \{x + 0 \to x, x + s(0) \to s(x)\}$. The normal-form grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ has five non-terminals: $\llcorner x^{\mathsf{Nat}} \lrcorner$, $\llcorner 0 \lrcorner$, $\llcorner s(0) \lrcorner$,

$\llcorner s(x) \lrcorner$, and $\llcorner s(s(x)) \lrcorner$, and six production rules (described with the same simplified notation as in Example 1):

$$\begin{array}{lll} \llcorner 0 \lrcorner := 0 & \llcorner s(0) \lrcorner := s(\llcorner 0 \lrcorner) & \llcorner s(x) \lrcorner := s(\llcorner x \lrcorner^{\mathsf{Nat}}) \\ \llcorner x \lrcorner^{\mathsf{Nat}} := 0 & \llcorner s(0) \lrcorner := s(\llcorner x \lrcorner^{\mathsf{Nat}}) & \llcorner s(s(x)) \lrcorner := s(\llcorner s(x) \lrcorner) \, [\![x \not\approx 0]\!] \end{array}$$

note that the non-terminals $\llcorner 0 \lrcorner$ and $\llcorner x \lrcorner^{\mathsf{Nat}}$ actually generate the same language, and that $\llcorner s(s(x)) \lrcorner$ generated no terms at all.

The procedure terminates by attaching *success* to all the leaves of the pattern tree $dtree(+)$, meaning that $\mathcal{R}$ is complete. Let us consider one interesting subtree of $dtree(+)$ with the root: $x_1 + y_1 \, [\![x_1 {:} \llcorner u \lrcorner \wedge y_1 {:} \llcorner s(s(y)) \lrcorner]\!]$, where $\llcorner u \lrcorner$ is any non-terminal. The first inference rule of Figure 2 instantiates it into: $x_1 + s(y_1) \, [\![x_1 {:} \llcorner u \lrcorner \wedge y_1 {:} \llcorner s(y) \lrcorner \wedge y \not\approx 0]\!]$ and $x_1 + s(s(y_1)) \, [\![x_1 {:} \llcorner u \lrcorner \wedge y_1 {:} \llcorner x \lrcorner^{\mathsf{Nat}} \wedge y_1 \not\approx 0]\!]$ and then $x_1 + s(s(0)) \, [\![x_1 {:} \llcorner u \lrcorner \wedge 0 \not\approx 0]\!]$ which is strongly ground reducible because its constraint is unsatisfiable.

Appendix C Example: sufficient completeness of *sorted*

There are two *failure* leaves in the pattern tree $dtree(sorted)$ described in Figure 4, labeled respectively by $failure\big(sorted(\emptyset)\big)$ and $failure\big(sorted(ins(x^{\mathsf{Nat}}, \emptyset))\big)$. The reason is that these terms do not contain any induction variable and that moreover they are not strongly ground reducible, because they do not match a left-hand-side of rule of $\mathcal{R}_{\mathcal{D}}$. This suggests to complete $\mathcal{R}_{\mathcal{D}}$ with two rules $sorted(\emptyset) \to true$ and $sorted(ins(x, \emptyset)) \to true$. The symbol *sorted* is sufficient complete wrt the TRS obtained. Indeed, $failure\big(sorted(\emptyset)\big)$ and

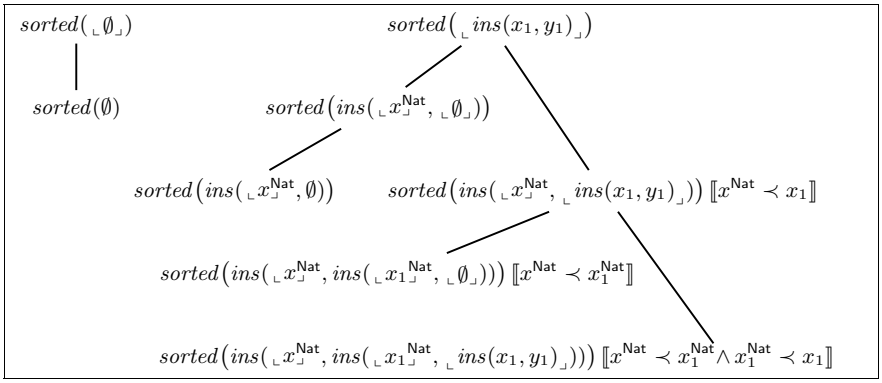

Fig. 4. The pattern tree of *sorted*

$failure\big(sorted(ins(x^{\mathsf{Nat}}, \emptyset))\big)$ are now both reducible by the new rewrite rules.

Moreover, the term: $sorted(ins(\llcorner x^{\mathsf{Nat}}\lrcorner, ins(\llcorner x_1^{\mathsf{Nat}}\lrcorner, \llcorner\emptyset\lrcorner)))\,[\![x^{\mathsf{Nat}} \prec x_1^{\mathsf{Nat}}]\!]$, which is an abbreviation for:

$$sorted(ins(z_1, ins(z_2, z_3)))\,[\![z_1{:}\llcorner x^{\mathsf{Nat}}\lrcorner \wedge z_2{:}\llcorner x_1^{\mathsf{Nat}}\lrcorner \wedge z_3{:}\llcorner\emptyset\lrcorner \wedge x^{\mathsf{Nat}} \prec x_1^{\mathsf{Nat}}]\!]$$

and $sorted(ins(\llcorner x^{\mathsf{Nat}}\lrcorner, ins(\llcorner x_1^{\mathsf{Nat}}\lrcorner, \llcorner ins(x_1, y_1)\lrcorner)))\,[\![x^{\mathsf{Nat}} \prec x_1^{\mathsf{Nat}} \wedge x_1^{\mathsf{Nat}} \prec x_1]\!]$, which is an abbreviation for:

$$sorted(ins(z_1, ins(z_2, z_3)))\left[\!\!\left[\begin{array}{c} z_1{:}\llcorner x^{\mathsf{Nat}}\lrcorner \wedge \;\; z_2{:}\llcorner x_1^{\mathsf{Nat}}\lrcorner \;\; \wedge z_3{:}\llcorner ins(x_1, y_1)\lrcorner \\ \wedge\, x^{\mathsf{Nat}} \prec x_1^{\mathsf{Nat}} \wedge \quad x_1^{\mathsf{Nat}} \prec x_1 \end{array}\right]\!\!\right]$$

are strongly ground reducible since the two following conjectures are inductive theorems of $\mathcal{R}$, and can be proved using the method of [3]:

$$sorted(ins(z_2, z_3)) = true\,[\![z_2{:}\llcorner x^{\mathsf{Nat}}\lrcorner \wedge z_3{:}\llcorner\emptyset\lrcorner]\!]$$
$$sorted(ins(z_2, z_3)) = true\,[\![z_2{:}\llcorner x^{\mathsf{Nat}}\lrcorner \wedge z_3{:}\llcorner ins(x_1, y_1)\lrcorner \wedge x^{\mathsf{Nat}} \prec x_1]\!]$$

Appendix D Example: sufficient completeness of *co*

In order to prove that the function *co* is sufficiently complete, we show that $\mathcal{R} \models_{\mathcal{I}nd} x \in' y = x \in y$, using the method of [3] (without user interaction).

For this purpose, we constrained the variable y in this conjecture to the language of non terminals of the grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$:

$$x \in' \llcorner\emptyset\lrcorner = x \in \llcorner\emptyset\lrcorner \tag{5}$$
$$x \in' \llcorner ins(x_1, y_1)\lrcorner = x \in \llcorner ins(x_1, y_1)\lrcorner \tag{6}$$

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

$$x \in' \emptyset = x \in \emptyset \tag{7}$$
$$x \in' ins(\llcorner x^{\mathsf{Nat}}\lrcorner, \emptyset) = x \in ins(\llcorner x^{\mathsf{Nat}}\lrcorner, \emptyset) \tag{8}$$
$$x \in' ins(\llcorner x^{\mathsf{Nat}}\lrcorner, \llcorner ins(x_1, y_1)\lrcorner) = x \in ins(\llcorner x^{\mathsf{Nat}}\lrcorner, \llcorner ins(x_1, y_1)\lrcorner)\,[\![x^{\mathsf{Nat}} \prec x_1]\!] \tag{9}$$

The clause (7) can be reduced by $\mathcal{R}_{\mathcal{D}}$ to the tautology $false = false$. For (8) we consider a restriction to the cases corresponding to the constraints of the last 3 rules for $\in'$ in $\mathcal{R}_{\mathcal{D}}$ (the rules with $x_1 \in' ins(x_2, y)$ as left member). This technique is called *Rewrite Splitting* in [3], it returns:

$$true = x \in ins(\llcorner x^{\mathsf{Nat}}\lrcorner, \emptyset)\,[\![x \approx x^{\mathsf{Nat}}]\!] \tag{10}$$
$$false = x \in ins(\llcorner x^{\mathsf{Nat}}\lrcorner, \emptyset)\,[\![x \prec x^{\mathsf{Nat}}]\!] \tag{11}$$
$$x \in' \emptyset = x \in ins(\llcorner x^{\mathsf{Nat}}\lrcorner, \emptyset)\,[\![x \succ x^{\mathsf{Nat}}]\!] \tag{12}$$

All these subgoal are reduced by $\mathcal{R}_{\mathcal{D}}$ into tautologies $true = true$ or $false = false$. Similarly, the application of *Rewrite Splitting* to (9) returns:

$$true = x \in ins(\llcorner x^{\mathsf{Nat}}\lrcorner, \llcorner ins(x_1, y_1)\lrcorner)\,[\![x^{\mathsf{Nat}} \prec x_1, x \approx x^{\mathsf{Nat}}]\!] \tag{13}$$
$$false = x \in ins(\llcorner x^{\mathsf{Nat}}\lrcorner, \llcorner ins(x_1, y_1)\lrcorner)\,[\![x^{\mathsf{Nat}} \prec x_1, x \prec x^{\mathsf{Nat}}]\!] \tag{14}$$
$$x \in' \llcorner ins(x_1, y_1)\lrcorner = x \in ins(\llcorner x^{\mathsf{Nat}}\lrcorner, \llcorner ins(x_1, y_1)\lrcorner)\,[\![x^{\mathsf{Nat}} \prec x_1, x \succ x^{\mathsf{Nat}}]\!] \tag{15}$$

The subgoal (13) is reduced by the second rule $\mathcal{R}_{\mathcal{D}}$ for $\in$ (the one with an equality constraint) into the tautology $true = true$. The clause (14) is simplified by Rewrite Splitting with the constrained rules of $\mathcal{R}_{\mathcal{D}}$ for $\in$, into:

$$false = true \, [\![x^{\mathsf{Nat}} \prec x_1, x \prec x^{\mathsf{Nat}}, x \approx x^{\mathsf{Nat}}]\!] \tag{16}$$

$$false = x \in \llcorner ins(x_1, y_1) \lrcorner \, [\![x^{\mathsf{Nat}} \prec x_1, x \prec x^{\mathsf{Nat}}, x \not\approx x^{\mathsf{Nat}}]\!] \tag{17}$$

The subgoal (16) is valid since its constraint is unsatisfiable. The clause (17) cannot be reduced and needs to be further instantiated using the production rules of the normal form grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$. This returns (with variable renaming):

$$false = x \in ins(\llcorner x_2 \lrcorner^{\mathsf{Nat}}, \emptyset) \, [\![x^{\mathsf{Nat}} \prec x_2^{\mathsf{Nat}}, x \prec x^{\mathsf{Nat}}, x \not\approx x^{\mathsf{Nat}}]\!] \tag{18}$$

$$false = x \in ins(\llcorner x_2 \lrcorner^{\mathsf{Nat}}, \llcorner ins(x_2, y_2) \lrcorner) \, [\![x^{\mathsf{Nat}} \prec x_2^{\mathsf{Nat}}, x \prec x^{\mathsf{Nat}}, x \not\approx x^{\mathsf{Nat}}, x_2^{\mathsf{Nat}} \prec x_2]\!] \tag{19}$$

Note that, thanks to the constraints in the production rules of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$, the constraint of (19) implies that both $x_2^{\mathsf{Nat}} \prec x_2$ and $x \prec x_2^{\mathsf{Nat}}$.

The clause (18) can be reduced by $\mathcal{R}_{\mathcal{D}}$ to the tautology $false = false$. The clause (19) can be reduced to the same tautology using the clause (14), which is used in this case as an induction hypothesis.

let us come back to the subgoal (15). The application of Rewrite Splitting (again with the constrained rules of $\mathcal{R}_{\mathcal{D}}$ for $\in$) returns:

$$x \in' \llcorner ins(x_1, y_1) \lrcorner = true \, [\![x^{\mathsf{Nat}} \prec x_1, x \succ x^{\mathsf{Nat}}, x \approx x^{\mathsf{Nat}}]\!] \tag{20}$$

$$x \in' \llcorner ins(x_1, y_1) \lrcorner = x \in \llcorner ins(x_1, y_1) \lrcorner \, [\![x^{\mathsf{Nat}} \prec x_1, x \succ x^{\mathsf{Nat}}, x \not\approx x^{\mathsf{Nat}}]\!] \tag{21}$$

The subgoal (20) is valid since its constraint is unsatisfiable. The last subgoal (21) is reduced by application of (6) (used as induction hypothesis) into the tautology:

$$x \in \llcorner ins(x_1, y_1) \lrcorner = x \in \llcorner ins(x_1, y_1) \lrcorner \, [\![x^{\mathsf{Nat}} \prec x_1, x \succ x^{\mathsf{Nat}}, x \not\approx x^{\mathsf{Nat}}]\!]$$

Appendix E Example: powerlists

Powerlists [22] are lists of 2^n elements (for $n \geq 0$) stored in the leaves of balanced binary trees. Let us consider the following set of constructor symbols in order to represent the powerlists of natural numbers:

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

The symbol v creates a singleton powerlist $v(n)$ 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. Every other term $tie(s, t)$ is reduced to $\bot$ by the following constructor system $\mathcal{R}_{\mathcal{C}}$. Therefore, the well formed powerlists are $\mathcal{R}_{\mathcal{C}}$-irreducible ground terms of sort List. In the definition of $\mathcal{R}_{\mathcal{C}}$, the binary constraint predicate $\sim$ is defined on constructor terms of sort List as the smallest

equivalence such that $v(x) \sim v(y)$ for all x ,y of sort Nat and $tie(x_1, x_2) \sim tie(y_1, y_2)$ iff $x_1 \sim x_2 \sim y_1 \sim y_2$. The TRS $\mathcal{R}_\mathcal{C}$ has one rule constrained by $\sim$:

$$\mathcal{R}_\mathcal{C} = \big\{\, tie(y_1, y_2) \to \bot \,[\![y_1 \not\sim y_2]\!],\ tie(\bot, y) \to \bot,\ tie(y, \bot) \to \bot \,\big\}$$

The tree grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ has non-terminals $\llcorner x\lrcorner^{\mathsf{Nat}}$, $\llcorner x\lrcorner^{\mathsf{List}}$, $\llcorner \bot \lrcorner$ and $\llcorner tie(x_1, x_2)\lrcorner$ and the production rules:

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

Note that all the constraints in these production rules are applied to brother subterms. The emptiness problem is actually 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 one-memory tree automata [9].

We propose a definition of the operator *zip* by the following rules of $\mathcal{R}_\mathcal{D}$:

$$\begin{array}{c}
zip(v(x_1), v(x_2)) \to tie(v(x_1), v(x_2)), \\
zip(tie(x_1, x_2), tie(x_3, x_4)) \to tie(zip(x_1, x_3), zip(x_2, x_4)), \\
zip(v(x_1), tie(x_2, x_3)) \to \bot,\ zip(tie(x_1, x_2), v(x_3)) \to \bot,\ zip(\bot, x) \to \bot,\ zip(x, \bot) \to \bot
\end{array}$$

The sufficient completeness of *zip* can be established with the pattern tree construction. It means in particular that this operator is defined on all well formed powerlists. Let us look at a few cases of $dtree(zip)$. The subtree with root $zip(\llcorner x_1\lrcorner^{\mathsf{List}}, \llcorner x_2\lrcorner^{\mathsf{List}})$ has nodes (without induction variables) of the form $zip(v(\llcorner x_1\lrcorner^{\mathsf{Nat}}), v(\llcorner x_2\lrcorner^{\mathsf{Nat}}))$ which are strongly ground reducible. Hence all the corresponding leaves are labelled with *success*. The situation is the same for the subtrees whose root has one (variable) subterm at least constrained by $\llcorner \bot \lrcorner$, and for the subtrees with root $zip(\llcorner tie(x_1, x_2)\lrcorner, \llcorner x_3\lrcorner^{\mathsf{List}})$ or $zip(\llcorner x_3\lrcorner^{\mathsf{List}}, \llcorner tie(x_1, x_2)\lrcorner)$, or also $zip(\llcorner tie(x_1, x_2)\lrcorner, \llcorner tie(x_3, x_4)\lrcorner)$ (because $\mathcal{R}_\mathcal{D}$ is unconstrained).