Sufficient Completeness Verification for Conditional and Constrained Term Rewriting Systems

Adel Bouhoula, Florent Jacquemard

Sufficient Completeness Verification for Conditional and Constrained TRS[☆]

Adel Bouhoula[a], Florent Jacquemard[b]

[a] *Higher School of Communication of Tunis (Sup'Com), University of Carthage, Tunisia.*
[b] *INRIA Saclay - IdF & LSV, ENS Cachan, 61 av. du pdt Wilson, 94230 Cachan, France.*

Abstract

We present a procedure for checking sufficient completeness of 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 into 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 procedure is presented by an inference system which is shown sound and complete. A precondition of one inference of this system refers to a (undecidable) property called strong ground reducibility which is discharged to the above inductive theorem proving system. We have successfully applied our method to several examples, yielding readable proofs and, in case of negative answer, a counter-example suggesting how to complete the specification. Moreover, we show that it is a decision procedure when the TRS is unconditional but constrained, for an expressive class of constrained constructor axioms.

Keywords: Sufficient Completeness, Conditional and Constrained Term Rewriting, Narrowing, Tree Grammars.

1. Introduction

Sufficient completeness [18] is a fundamental property of algebraic specifications. It expresses that some functions are defined on every value by a given a term rewriting system (TRS) $\mathcal{R}$. More precisely, 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

[☆] This work has been partially supported by the grant INRIA–DGRSRT 10/01.
Email addresses: adel.bouhoula@supcom.rnu.tn (Adel Bouhoula), florent.jacquemard@inria.fr (Florent Jacquemard)

without variables) of a given term are reducible by a given TRS [28, 29, 31]. For instance, a terminating TRS $\mathcal{R}$ is sufficiently complete iff for every non-constructor symbol f, $f(x_1, \ldots, x_n)$ is ground reducible by $\mathcal{R}$.

Sufficient completeness is undecidable in general [19] but decidability results have been obtained for restricted cases of unconditional TRS [20, 23, 31, 8, 24, 33, 29]. Tree automata with constraints have appeared to be a well suited framework for the decision of sufficient completeness and related properties, see e.g. [8, 6, 10] or [7] for a survey. In particular, the decision of ground reducibility is reducible to the problem of emptiness for tree automata with disequality constraints.

In the context of specifications given as TRS with conditions (*i.e.* equational Horn clauses) and constraints, 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 between constructor terms which can be constrained and are not necessarily left-linear. Such rules permit the axiomatization of complex data structures like *e.g.* sorted lists or powerlists, see Section 9. Our method is based on the incremental construction of a finite *pattern tree* for each non-constructor symbol. The pattern trees are labeled by constrained terms and every construction step is defined as a non-terminal replacement by a constrained tree grammar which generates the set of ground constructor terms irreducible by $\mathcal{R}$. Roughly, the idea is to built a finite representation of all the terms of the form $f(t_1, \ldots, t_n)$ such that f is a non-constructor symbol and $t_1, \ldots, t_n$ are ground constructor terms irreducible by $\mathcal{R}$ (if $\mathcal{R}$ is ground convergent, then it is sufficiently complete iff every such term is reducible). We show that it is sufficient in these settings to consider a finite set of positions of non-terminals to be replaced, and therefore that the construction terminates.

The criterion for the verification of sufficient completeness is that all the leaves of the pattern trees are *strongly ground reducible* by $\mathcal{R}$. This sufficient condition for ground reducibility requires in particular that the conditions of candidate rules of $\mathcal{R}$ (for reducing ground instances) are inductive consequences of $\mathcal{R}$ (hence it is undecidable in general for conditional term rewriting systems). Therefore, our procedure for sufficient completeness verification has been integrated with a procedure for inductive theorem proving [2], presented in Section 3.3. The proof obligations generated, when $\mathcal{R}$ is conditional, are discharged to this inductive theorem procedure, which is sound and refutationally complete for the kind of conjectures considered here. Both the procedure of [2] for inductive theorem proving and our procedure for sufficient completeness verification are based on the same framework with constrained tree grammars (Section 3) which provide the glue between both procedures. Moreover, they are crucial for the completeness of the procedure of this paper. Indeed, they provide an *exact* finite representation of ground constructor terms in normal form. In comparison, the *cover sets* used *e.g.* in [26, 1, 5] may be over-approximating (*i.e.* they may represent also some reducible terms) in presence of axioms for constructors.

To sum up, our approach handles axioms for constructors even with constraints, and 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.* [3] – see below. The procedure is based on an inference system which is shown *sound* for ground convergent specifications and it is also *complete*. The assumptions about ground convergence (ground confluence and termination) are discussed in Section 8. One inference requires a test for strong ground reducibility discharged to the inductive theorem proving system, used as an oracle, as explained above 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. The failure of the strong ground reducibility test is also an indication on the conditions missing.

When $\mathcal{R}$ is unconditional, we have a decision procedure for sufficient completeness for an expressive enough class of constrained constructor rules.

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, as well as the method of [2] for inductive theorem proving (Section 3.3). Section 4 introduces sufficient completeness and strong ground reducibility, the sufficient condition used for its decision. After some motivating examples (Section 5), the procedure for checking sufficient completeness is described by an inference system in Section 6, where the correctness and completeness are proved. A decidable subcase is identified in Section 7. In Section 8, we suggest methods for checking the properties of ground confluence and termination, which are needed in the results of Section 6. Finally, the examples of application of this procedure to specifications of integers and integers modulo (Sections 5.1 and 5.2), sorted lists (Sections 9.1-9.3) and powerlists (Section 9.5) show that our method yields very natural proofs on these cases, whereas other related techniques fail.

Related work.. A procedure has been proposed in [1] 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 [3], tree automata techniques are used to check sufficient completeness of specifications with axioms between constructors. This technique has been generalized to membership equational logic [4] in order to support partial conditional specifications with sorts and subsorts and functions domains defined by conditional memberships. The approaches of [3, 4] 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*.

A more general framework has been proposed in [21] (as an extension of [4]), allowing a much wider class of Membership Equational Logic (MEL) specifications to be checked. The system of [21] 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 Maude's

inductive theorem prover and may need user interaction (see the example of sorted lists, Section 9.1). Note that the generated proof obligations can be invalid 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 [21] 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 work [22] generalizes the framework of [21] 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 [22] 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 that it would certainly be worth to study a combination of the automata modulo AC of [22] and the constrained automata used in this paper.

A promising approach is proposed in [17] for proving sufficient completeness without the assumption of confluence and termination of unconditional and unconstrained specifications. Note that if we restrict our technique to constrained and unconditional specifications, we do not need ground confluence anymore, but still the termination of the subset $\mathcal{R}_{\mathcal{D}}$ of non-constructor rules of $\mathcal{R}$ (see Theorem 3). We can relax the condition on termination of $\mathcal{R}_{\mathcal{D}}$ by assuming explicitly that the rewrite rules which are used in order to prove strong ground reducibility of leaves are orientable. With this assumption, our procedure remains correct and complete. Given the complementarity of the two approaches (the expressiveness of our method, with conditions and constraints, and the absence of assumptions in [17]) we think that studying a combination of our techniques and a generalization of [17] could be fruitful for the verification of sufficient completeness for constrained and conditional specifications.

2. Definitions

The reader is assumed familiar with the basic notions of term rewriting [12]. 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*. We denote by $\mathcal{T}(\mathcal{F}, \mathcal{X})$ (resp. $\mathcal{T}(\mathcal{C}, \mathcal{X})$) the set of well-sorted terms over $\mathcal{F}$ (resp. constructor well-sorted terms) with variables in $\mathcal{X}$ and $\mathcal{T}(\mathcal{F})$ (resp. $\mathcal{T}(\mathcal{C})$) its subset of of variable-free terms, or *ground* terms. We assume that each sort contains at least one ground term. We write $var(t)$ for

the set of variables occurring in a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $sort(t)$ for its sort. A term t is *linear* if every variable occurs at most once in t.

The *subterm* of a term t at position p is denoted by $t|_p$ and the result of replacing with s the subterm $t|_p$ of t is denoted by $t[s]_p$ (p may be omitted when we just want to indicate that s is a subterm of t). A *substitution* is a finite mapping from variables to terms. As usual, we identify substitutions with their morphism extension to terms. 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 interpreted over $\mathcal{T}(\mathcal{C})$. Typically, $\mathcal{L}$ may contain the syntactic equality $. \approx .$ or syntactic disequality $. \not\approx .$, some simplification ordering $. \prec .$ like *e.g.* a lexicographic path ordering [12], or membership predicates $x : L$ referring to some fixed subsets of $\mathcal{T}(\mathcal{C})$ defined by a tree grammar (see Section 3). *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 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. It is 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, 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 term rewriting system* or CCTRS. If the rules of $\mathcal{R}$ contain no conditions, $\mathcal{R}$ is called an unconditional constrained TRS.

A term $t \, [\![d]\!]$ rewrites to $s \, [\![d]\!]$ by the above rule $\rho \in \mathcal{R}$ 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 $\xrightarrow[\mathcal{R}]{*}$ denotes the reflexive transitive closure of $\xrightarrow[\mathcal{R}]{}$. Note the semantic difference between conditions and constraints in rewrite rules: the validity of conditions is defined *wrt* $\mathcal{R}$ whereas the interpretation of constraints is fixed and independent from $\mathcal{R}$.

A CCTRS $\mathcal{R}$ is *terminating* if there is no infinite sequence $t_1 \xrightarrow[\mathcal{R}]{} t_2 \xrightarrow[\mathcal{R}]{} \ldots$, and $\mathcal{R}$ 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.

Reducibility and ground reducibility.. If there exists a term $s\,[\![d]\!]$ such that $t\,[\![d]\!] \xrightarrow[\mathcal{R}]{} s\,[\![d]\!]$, then $t\,[\![d]\!]$ is called *reducible* by $\mathcal{R}$. Otherwise $t\,[\![d]\!]$ is called *irreducible* by $\mathcal{R}$, or an $\mathcal{R}$*-normal form*. A constrained term $t\,[\![c]\!]$ is *ground reducible* by $\mathcal{R}$ (resp. *ground irreducible*) if $t\sigma$ is reducible (resp. irreducible) for every irreducible solution σ of c grounding for t.

Constructor specifications and sufficient completeness.. We assume from now on that every CCTRS $\mathcal{R}$ is partitioned into $\mathcal{R} = \mathcal{R}_{\mathcal{D}} \uplus \mathcal{R}_{\mathcal{C}}$ where $\mathcal{R}_{\mathcal{D}}$ contains conditional constrained rules of the form $\Gamma \Rightarrow f(\ell_1, \ldots, \ell_n) \to r\,[\![c]\!]$ with $f \in \mathcal{D}$, $\ell_1, \ldots, \ell_n \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ and $\mathcal{R}_{\mathcal{C}}$ contains constrained rewrite rules with constructor symbols in $\mathcal{C}$ only. A CCTRS $\mathcal{R}$ is *sufficiently complete* iff for all $t \in \mathcal{T}(\mathcal{F})$ there exists s in $\mathcal{T}(\mathcal{C})$ such that $t \xrightarrow[\mathcal{R}]{*} s$.

Inductive theorems.. A clause C is a *deductive theorem* of a CCTRS $\mathcal{R}$ (denoted by $\mathcal{R} \models C$) if it is valid in any model of $\mathcal{R}$. A clause C is an *inductive theorem* of $\mathcal{R}$ (denoted by $\mathcal{R} \models_{\mathcal{I}nd} C$) iff for all substitution σ grounding for C, $\mathcal{R} \models C\sigma$. A constrained clause $C\,[\![c]\!]$ is an *inductive theorem* of a CCTRS $\mathcal{R}$ (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. Constrained Tree Grammars and Inductive Theorem Proving

Constrained tree grammars permit an exact finite representation of the set of ground terms irreducible by a given unconditional and constrained TRS. In our approach, as well as in [2] (see below), they are used to generate incrementally a relevant set of constrained terms, by means of non-terminal replacement following production rules.

3.1. Term languages

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* $\llcorner u \lrcorner := f(\llcorner u_1 \lrcorner, \ldots, \llcorner u_n \lrcorner)\,[\![c]\!]$ *where* $f \in \mathcal{F}$, $\llcorner u \lrcorner$, $\llcorner u_1 \lrcorner, \ldots, \llcorner u_n \lrcorner \in Q$ *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. The constraint $[\![c]\!]$ may be omitted when $c = \mathit{true}$. 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 $\vdash^+$.

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 $\llcorner u \lrcorner$,

$$L(\mathcal{G}, \llcorner u \lrcorner) = \big\{ t \in \mathcal{T}(\mathcal{F}) \;\big|\; x\,[\![x : \llcorner u \lrcorner]\!] \vdash^* t\,[\![c]\!] \text{ and } c \text{ is satisfiable} \big\}.$$

We define the semantics of membership constraints used in production rules, of the form $t : \llcorner u \lrcorner$, with $\llcorner u \lrcorner \in Q$, by: $sol(t : \llcorner u \lrcorner) = \{\sigma \mid t\sigma \in L(\mathcal{G}, \llcorner u \lrcorner)\}$. Note that this allows to use such constraint for instance to restrict a term to a given sort or any given regular tree language.

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 $p, s : \mathsf{Int} \to \mathsf{Int}$. Let $\mathcal{G}_{\mathsf{Int}}$ be a constrained tree grammar with three non-terminals $\llcorner s(x) \lrcorner$, $\llcorner p(x) \lrcorner$ and $\llcorner 0 \lrcorner$ and the production rules

$$\begin{array}{lcl} \llcorner 0 \lrcorner & := & 0 \end{array} \qquad \begin{array}{lcl} \llcorner s(x) \lrcorner & := & s(\llcorner 0 \lrcorner) \\ \llcorner s(x) \lrcorner & := & s\big(\llcorner s(x) \lrcorner\big) \end{array} \qquad \begin{array}{lcl} \llcorner p(x) \lrcorner & := & p(\llcorner 0 \lrcorner) \\ \llcorner p(x) \lrcorner & := & p\big(\llcorner p(x) \lrcorner\big) \end{array}$$

The languages generated by the non-terminals are $L(\mathcal{G}_{\mathsf{Int}}, \llcorner 0 \lrcorner) = \{0\}$, $L(\mathcal{G}_{\mathsf{Int}}, \llcorner s(x) \lrcorner) = \{s^m(0) \mid m > 0\}$, $L(\mathcal{G}_{\mathsf{Int}}, \llcorner p(x) \lrcorner) = \{p^m(0) \mid m > 0\}$. ◇

Example 2. Let Nat be a sort for natural numbers and List be a sort for lists of Nat and assume that the set $\mathcal{C}$ of constructor symbols contains $0 : \mathsf{Nat}$ and $s : \mathsf{Nat} \to \mathsf{Nat}$, $\emptyset : \mathsf{List}$ and $ins : \mathsf{Nat} \times \mathsf{List} \to \mathsf{List}$.

Let $\mathcal{G}_{\mathsf{List}}$ be a constrained tree grammar with four non-terminals: $\llcorner x^{\mathsf{Nat}} \lrcorner$, $\llcorner \emptyset \lrcorner$, $\llcorner ins(x_1, y_1) \lrcorner$, and the following set of production rules

$$\begin{array}{rcl} \llcorner x^{\mathsf{Nat}} \lrcorner & := & 0 \;\big|\; s(\llcorner x^{\mathsf{Nat}} \lrcorner) \\ \llcorner \emptyset \lrcorner & := & \emptyset, \\ \llcorner ins(x_1, y_1) \lrcorner & := & ins(\llcorner x^{\mathsf{Nat}} \lrcorner, \llcorner \emptyset \lrcorner) \\ \llcorner ins(x_1, y_1) \lrcorner & := & ins(\llcorner x^{\mathsf{Nat}} \lrcorner, \llcorner ins(x_2, y_2) \lrcorner)\,[\![x^{\mathsf{Nat}} \prec x_2]\!] \end{array}$$

The constraints in the last production rule refers to the standard ordering relation on natural numbers, represented by terms of $\mathcal{T}(\mathcal{C})$, *i.e.* $s^n(0) \prec s^m(0)$ iff $n < m$. The languages associated are $L(\mathcal{G}_{\mathsf{List}}, \llcorner x^{\mathsf{Nat}} \lrcorner) = \{s^m(0) \mid m \geq 0\}$, $L(\mathcal{G}_{\mathsf{List}}, \llcorner \emptyset \lrcorner) = \{\emptyset\}$, and for $\llcorner ins(x_1, y_1) \lrcorner$, the terms representing sorted lists:

$$L\big(\mathcal{G}_{\mathsf{List}}, \llcorner ins(x_1, y_1) \lrcorner\big) = \{ins(s^{n_1}(0), \ldots, ins(s^{n_k}(0), \emptyset)) \mid 0 < k, n_1 < \ldots < n_k\}.$$

The third above production rule permits to generate singleton list, whereas the last rule permits to generate lists with two or more elements which are sorted. The sorting is ensured by the constraint in the production rule. ◇

3.2. Normal form grammars

For every unconditional and constrained rewrite system $\mathcal{R}_\mathcal{C}$, we can construct a constrained tree grammar $\mathcal{G}_{\rm NF}(\mathcal{R}_\mathcal{C}) = (Q_{\rm NF}(\mathcal{R}_\mathcal{C}), \Delta_{\rm NF}(\mathcal{R}_\mathcal{C}))$ which generates the language of ground $\mathcal{R}_\mathcal{C}$-normal forms. Intuitively, this construction, which generalizes the one of [10], corresponds to the complementation and completion of a constrained tree automaton for terms reducible by $\mathcal{R}_\mathcal{C}$ (such an automaton does essentially pattern matching of left-hand side of rewrite rules), where every subset of non-terminals (for the complementation) is represented by the most general instance (mgi) of its elements.

Let $\mathcal{L}(\mathcal{R}_\mathcal{C})$ be the set containing the strict subterms of the left-hand sides of the rules of $\mathcal{R}_\mathcal{C}$, and let $Q_{\rm NF}(\mathcal{R}_\mathcal{C})$ be the set containing the non-terminals of the form $\llcorner x^S \lrcorner$ for each sort $S \in \mathcal{S}$ and every $\llcorner \text{mgi}(t_1, \ldots, t_n) \lrcorner$ such that $\{t_1, \ldots, t_n\}$ is a subset of unifiable terms of $\mathcal{L}(\mathcal{R}_\mathcal{C})$ of the same sort.

The set of transitions $\Delta_{\rm NF}(\mathcal{R}_\mathcal{C})$ contains every rule

$$\llcorner u \lrcorner := f(\llcorner u_1 \lrcorner, \ldots, \llcorner u_n \lrcorner) \, [\![\neg c]\!]$$

such that $f \in \mathcal{F}$ with profile $S_1 \times \ldots \times S_n \to S$, $\llcorner u \lrcorner, \llcorner u_1 \lrcorner, \ldots, \llcorner u_n \lrcorner \in Q_{\rm NF}(\mathcal{R}_\mathcal{C})$, $u_1, \ldots, u_n$ have respective sorts $S_1, \ldots, S_n$, u is the mgi of the set: $\{v \mid \llcorner v \lrcorner \in Q_{\rm NF}(\mathcal{R}_\mathcal{C})$ and v 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 3. Let us consider the following TRS defined on the signature defined in Example 1, $\mathcal{R}_\mathcal{C} = \{s(p(x)) \to x, \; p(s(x)) \to x\}$.

The above construction applied to $\mathcal{R}_\mathcal{C}$ returns a constrained tree grammar $\mathcal{G}_{\rm NF}(\mathcal{R}_\mathcal{C})$ which is identical to the grammar $\mathcal{G}_{\mathsf{Int}}$ of Example 1 except that the non-terminal $\llcorner 0 \lrcorner$ is now denoted by $\llcorner x^{\mathsf{Int}} \lrcorner$ (recall that this non-terminal only generates 0). More precisely, we have $\mathcal{L}(\mathcal{R}_\mathcal{C}) = \{s(x), p(x)\}$ and $Q_{\rm NF}(\mathcal{R}_\mathcal{C}) = \{\llcorner x^{\mathsf{Int}} \lrcorner, \llcorner s(x) \lrcorner, \llcorner p(x) \lrcorner\}$. Note that $\Delta_{\rm NF}(\mathcal{R}_\mathcal{C})$ also contains some production rules of the form $\llcorner s(x) \lrcorner := s\big(\llcorner p(x) \lrcorner\big) \, [\![\mathit{false}]\!]$ and $\llcorner p(x) \lrcorner := p\big(\llcorner s(x) \lrcorner\big) \, [\![\mathit{false}]\!]$ which are omitted because their application always leads to empty languages. $\diamond$

Example 4. Let us consider the following constrained constructor rules on the signature of Example 2

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

The ordering constraint $\succ$ is interpreted as a reduction ordering total on ground constructor terms (like e.g. a lexicographic path ordering).

The normal form grammar $\mathcal{G}_{\rm NF}(\mathcal{R}_\mathcal{C})$ associated to this constrained TRS is the grammar $\mathcal{G}_{\mathsf{List}}$ of Example 2, where the non-terminal $\llcorner \emptyset \lrcorner$ denotes $\llcorner x^{\mathsf{List}} \lrcorner$ (this non-terminal only generates the empty list $\emptyset$). $\diamond$

The proof of the following property can be found in the long version of [2].

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

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

3.3. Tree grammar based inductive theorem proving

In [2] we propose a new approach for automated inductive theorem proving for the same kind of CCTRS specifications as the ones considered in this paper. The procedure of [2] is based on the normal form constrained tree grammar $\mathcal{G}_{\rm NF}(\mathcal{R}_\mathcal{C})$, which is used to trigger induction steps by the generation of subgoals during the proof by induction and in decision procedures for checking redundancy criteria. It is also called to discharge proof obligations generated by the procedure for checking sufficient completeness defined in Section 6. The procedure of [2] is sound and refutationally complete for the decorated conjectures that we shall consider here.

We shall not present in detail the procedure of [2] here. Very roughly, its principle is to use the above constrained tree grammar $\mathcal{G}_{\rm NF}(\mathcal{R}_\mathcal{C})$ as an induction schema. This grammar permits the generation of subgoals from a conjecture C, by instantiation of variables using the grammar's production rules, triggering induction steps during the proof. All generated subgoals are either deleted, following some criteria, or they are reduced, using axioms or induction hypotheses, or conjectures not yet proved, providing that they are smaller than the goal to be proved. Reduced subgoals become then new conjectures and C becomes an induction hypothesis. The constrained tree grammar $\mathcal{G}_{\rm NF}(\mathcal{R}_\mathcal{C})$ is also used in decision procedures for checking the deletion criteria during induction steps. Let us just illustrate this principle on an example.

Example 5. We complete the specification of Examples 1, 3 with a sort Bool for Booleans, two constants of $\mathcal{C}$, $\mathit{true}, \mathit{false} : \mathsf{Bool}$ and one binary defined symbol $\leq\!: \mathsf{Int} \times \mathsf{Int} \to \mathsf{Bool}$ in $\mathcal{D}$. Let $\mathcal{R}_\mathcal{D}$ be the following set of conditional rules:

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

We show that the following decorated clause (1) is an inductive theorem of $\mathcal{R} = \mathcal{R}_\mathcal{C} \uplus \mathcal{R}_\mathcal{D}$:

$$0 \leq x_1 = \mathit{true}\,[\![x_1 : \llcorner s(x) \lrcorner]\!] \tag{1}$$

Applying the production rules of $\mathcal{G}_{\rm NF}(\mathcal{R}_\mathcal{C})$ to (1), we obtain two subgoals (induction step): $0 \leq s(x_1) = \mathit{true}\,[\![x_1 : \llcorner 0 \lrcorner]\!]$ and $0 \leq s(x_1) = \mathit{true}\,[\![x_1 : \llcorner s(x) \lrcorner]\!]$.

The first subgoal can be further instantiated by $\mathcal{G}_{\rm NF}(\mathcal{R}_\mathcal{C})$ 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 (*wrt* a well–founded ordering on constrained clauses, see [2] for a formal definition) than the second subgoal.

Hence, all the subgoals are reduced into tautologies which are deleted, and the procedure concludes that (1) is an an inductive theorem of $\mathcal{R}$. $\diamond$

4. Sufficient Completeness and Strongly Ground Reducibility

We shall now define some sufficient conditions used for the decision of sufficient completeness (Section 4.1) and relate these properties to inductive theorem proving (Section 4.2).

4.1. Sufficient condition for sufficient completeness

The procedure of Section 6 for checking sufficient completeness is based on the following definition which is equivalent to the definition given in Section 2.

Definition 2. *A function symbol $f \in \mathcal{D}$ is* sufficiently complete *wrt the CCTRS $\mathcal{R}$ iff for all $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C})$, there exists $s \in \mathcal{T}(\mathcal{C})$ such that $f(t_1, \ldots, t_n) \xrightarrow[\mathcal{R}]{+} s$.*

Proposition 2. *A CCTRS $\mathcal{R}$ is sufficiently complete iff every defined symbol $f \in \mathcal{D}$ is sufficiently complete wrt $\mathcal{R}$.*

Proof. The *only if* direction is obvious. We can show the *if* direction by induction on the number of occurrences of symbols of $\mathcal{D}$ in a given term $t \in \mathcal{T}(\mathcal{F})$. $\square$

Let us now state formally a key property for sufficient completeness verification already announced in the introduction.

Proposition 3. *Let $\mathcal{R}$ be a terminating CCTRS. A defined symbol f is sufficiently complete wrt $\mathcal{R}$ iff for all ground constructor terms $t_1, \ldots, t_n$ irreducible by $\mathcal{R}$, $f(t_1, \ldots, t_n)$ is reducible by $\mathcal{R}$.*

Proof. The direction $\Rightarrow$ is immediate. We prove the other direction $\Leftarrow$ by contradiction. Assume that every $f(t_1, \ldots, t_n)$ as in the Proposition is reducible by $\mathcal{R}$ and that $\mathcal{R}$ is not sufficiently complete. Let t be a term of $\mathcal{T}(\mathcal{F}) \setminus \mathcal{T}(\mathcal{C})$ not reducible to a constructor term and minimal wrt the (well founded) rewrite relation $\xrightarrow[\mathcal{R}]{}$. Since t contains a defined symbol, it is reducible by $\mathcal{R}$ into t'. Indeed, let p be a position in t of an innermost occurrence of a defined symbol f in t, and let $f(t_1, \ldots, t_n) = t|_p$ (with $n \geq 0$). The terms $t_1, \ldots, t_n$ are in $\mathcal{T}(\mathcal{C})$, and if they are all irreducible by $\mathcal{R}$, then $t|_p$ is reducible by hypothesis. Either t' is a constructor term or it is a smaller counter-example, and both cases are a contradiction. $\square$

4.2. Strong ground reducibility

The goal of our procedure for sufficient completeness verification (Section 6) is to test the condition of Proposition 3. A key problem in this context is to be able to check that the ground instances of a constrained term are reducible by $\mathcal{R}_{\mathcal{D}}$. For this purpose, we use the following sufficient condition for ground reducibility, based on the notion of inductive validity.

Definition 3. *A constrained term $t\,[\![c]\!]$ is* strongly ground reducible *by $\mathcal{R}$ if there exists n ($n > 0$) rules of $\mathcal{R}_{\mathcal{D}}$ denoted by $\Gamma_i \Rightarrow l_i \to r_i\,[\![c_i]\!]$, and n substitutions denoted by σ_i ($i \in [1..n]$) 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 valid and $\mathcal{R} \models_{\mathcal{I}nd} \Gamma_1\sigma_1\,[\![c \wedge c_1\sigma_1]\!] \vee \ldots \vee \Gamma_n\sigma_n\,[\![c \wedge c_n\sigma_n]\!]$.*

Example 6. Let $\mathcal{R} = \{a \to b, a \to c, b = c \Rightarrow f(x) \to 0\}$. The term $f(x)$ is irreducible by $\mathcal{R}$ but it is strongly ground reducible by $\mathcal{R}$ since $\mathcal{R} \models_{\mathcal{I}nd} b = c$.$\diamond$

Example 7. Let $\mathcal{R} = \{x \in \emptyset \to \mathit{false},\ x_1 \in \mathit{ins}(x_2, y) \to \mathit{true}\,[\![x_1 \approx x_2]\!],\ x_1 \in \mathit{ins}(x_2, y) \to x_1 \in y\,[\![x_1 \not\approx x_2]\!]\}$. The term $x_1 \in \mathit{ins}(x_2, y)$ is strongly ground reducible since the constraint $x_1 \approx x_2 \vee x_1 \not\approx x_2$ is valid. $\diamond$

The following lemma is an immediate consequence of Definition 3.

Lemma 1. *Let $\mathcal{R}$ be a unconditional constrained TRS. Every constrained term strongly ground reducible by $\mathcal{R}$ is reducible by $\mathcal{R}$.*

Lemma 2. *Let $\mathcal{R}$ be a ground confluent CCTRS. Every constrained term strongly ground reducible by $\mathcal{R}$ is ground reducible by $\mathcal{R}$.*

Proof. Let $t\,[\![c]\!]$ be a term strongly ground reducible by $\mathcal{R}$ and let $\sigma \in \mathit{sol}(c)$ be an irreducible solution of c grounding for t. We show that $t\sigma$ is reducible.

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 $t\sigma = l_i\sigma_i$ and $c \wedge \neg c_i\sigma_i$ is unsatisfiable for all $i \in [1..n]$ (this is obviously true since by definition, $\neg c \vee c_1\sigma_1 \vee \ldots \vee c_n\sigma_n$ is valid) and $\mathcal{R} \models_{\mathcal{I}nd} \Gamma_1\sigma_1\,[\![c \wedge c_1\sigma_1]\!] \vee \ldots \vee \Gamma_n\sigma_n\,[\![c \wedge c_n\sigma_n]\!]$.

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

Lemma 2 does not work if $\mathcal{R}$ is not ground confluent.

Example 8. The conditional TRS of Example 7 is not ground confluent. The term $f(x)$, which is strongly ground reducible by $\mathcal{R}$ is not ground reducible by $\mathcal{R}$ since for example $f(a)$ is not reducible by $\mathcal{R}$. $\diamond$

Also, the converse of Lemma 2 is not true.

Example 9. Let $\mathcal{R} = \{\mathit{even}(0) \to \mathit{true}, \mathit{even}(s(0)) \to \mathit{false}, \mathit{even}(s(s(x))) \to \mathit{even}(x)\}$. This unconditional TRS is ground confluent and sufficiently complete. The term $\mathit{even}(x)$ is ground reducible by $\mathcal{R}$ but it is not strongly ground reducible by $\mathcal{R}$. $\diamond$

5. Examples: Integers

In this section, we shall introduce, with two examples, the procedure presented in Section 6 for the verification of sufficient completeness. As mentioned above, the key elements for this procedure are the sufficient condition given in Proposition 3 and the notion of strong ground reducibility of Definition 3. The main idea is that a term of the form $f(t_1, \ldots, t_k)$ where f is a defined symbol and $t_1, \ldots, t_k$ are ground constructor terms irreducible by a CCTRS $\mathcal{R}$, as in Proposition 3, is reducible by $\mathcal{R}$ only if it is reducible at the root position. Hence, in order to check the condition of Proposition 3, it is sufficient to consider the positions of terms $f(t_1, \ldots, t_k)$ close to the root. The procedure of Section 6 aims at covering all the cases of terms of the form $f(t_1, \ldots, t_k)$, with a finite number of tests. It generates incrementally the top part of such terms by non-terminal replacement using the production rules of the normal form grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$, starting from the terms of the form $f(x_1, ..., x_k)\, [\![x_1 : n_1 \wedge \ldots \wedge x_k : n_k]\!]$ where $n_1, ..., n_k$ are non-terminals of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$. All the constrained terms generated are arranged in a derivation tree called *pattern tree*. At every construction step, the procedure checks the term under construction for strong ground reducibility, in order to check the condition of Proposition 3. This test may require a call to the inductive theorem proving procedure of [2]. If the answer to the test is positive, then the construction is stopped for the current constrained term. This corresponds to a successful leave of the pattern tree. The procedure also stops the generation when the top part generated is deep enough to cover all the left-hand sides of rewrite rules of $\mathcal{R}_{\mathcal{D}}$. In this case, if the current tree is not strongly ground reducible, then we have a failure leave in the pattern tree and $\mathcal{R}$ is not sufficiently complete. The failure leaves can be used afterward as a counter-example, in order to analyse which case is not covered by $\mathcal{R}$, and suggests therefore which rules must be added to $\mathcal{R}$ in order to obtain sufficient completeness.

5.1. Integers

Let us continue with Examples 1, 3 and 5. Since $\mathcal{R}$ is ground convergent, following Proposition 3, in order to check the sufficient completeness of the symbol $\leq$ *wrt* $\mathcal{R}$, it is sufficient to consider the reductions of the terms of the form $t_1 \leq t_2$ where t_1 and t_2 are terms of $\mathcal{T}(\mathcal{C})$ irreducible by $\mathcal{R}_{\mathcal{C}}$. By Proposition 1, such terms 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-terminals of $Q_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$. For the sake of readability, we shall denote such a term $n_1 \leq n_2$ below. The multi-rooted tree labeled by constrained terms of Figure 1 is called pattern tree and denoted by $dtree(\leq)$ in Section 6.

Every child in the tree of Figure 1 is obtained from its ancestor by replacement of some non-terminal according to the production rules of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$. This tree covers all the necessary cases for checking sufficient completeness, according to the following case analysis.

- $\llcorner 0 \lrcorner \leq \llcorner 0 \lrcorner$ is instantiated by $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ into $0 \leq 0$ which is reducible by $\mathcal{R}_{\mathcal{D}}$.

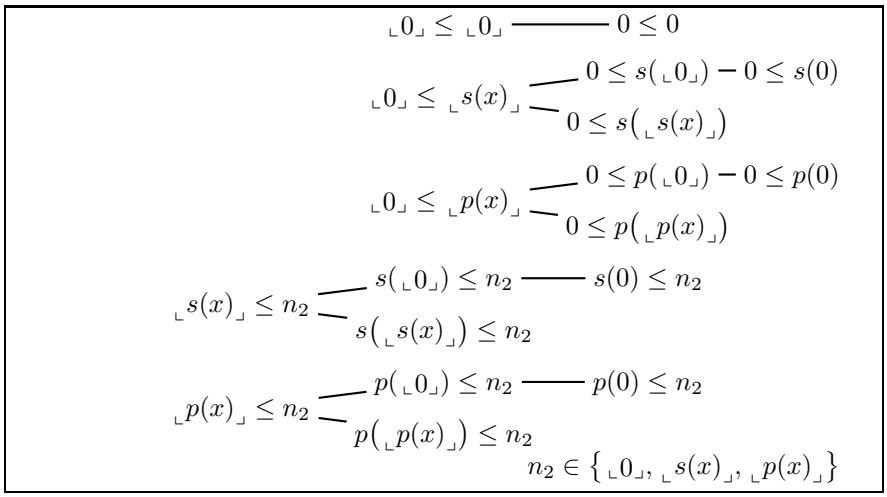

Figure 1: Sufficient completeness of $\leq$

- $\llcorner 0 \lrcorner \leq \llcorner s(x) \lrcorner$ 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 $\mathcal{R} \models_{\mathcal{I}nd} 0 \leq x_1 = \mathit{true}\, [\![x_1 : \llcorner s(x) \lrcorner]\!]$ (see Example 5). Consequently, any further derivation with $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ will result in terms reducible (at the root) by $\mathcal{R}$.
- $\llcorner 0 \lrcorner \leq \llcorner p(x) \lrcorner$: similarly, using $\mathcal{R} \models_{\mathcal{I}nd} 0 \leq x_1 = \mathit{false}\, [\![x_1 : \llcorner p(x) \lrcorner]\!]$.
- $\llcorner s(x) \lrcorner \leq n_2$ (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 [1]. 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 ground constructor terms irreducible by $\mathcal{R}$. For instance $p(s(0))$ is an instance of $p(x)$ but is not irreducible. The methods of [3, 4] 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 [21] generates one proof obligation which is not valid. It is possible to prove the sufficient completeness of this specification with [21] using a transformation into a new specification with free constructors by specifying subsorts for zero, positive and negative integers respectively.

5.2. 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 two non-terminals: $\llcorner x \lrcorner^{\mathsf{Nat}}$ and $\llcorner s(x) \lrcorner$ and three production rules.

$$\llcorner x \lrcorner^{\mathsf{Nat}} := 0 \quad \llcorner s(x) \lrcorner := s(\llcorner x \lrcorner^{\mathsf{Nat}}) \quad \llcorner s(x) \lrcorner := s(\llcorner s(x) \lrcorner)\,[\![x \not\approx 0]\!]$$

The pattern tree *dtree*$(+)$ associated to the defined symbol $+$ is described in Figure 2. All its leaves are strongly ground reducible, like in Section 5.1, meaning that $\mathcal{R}$ is complete.

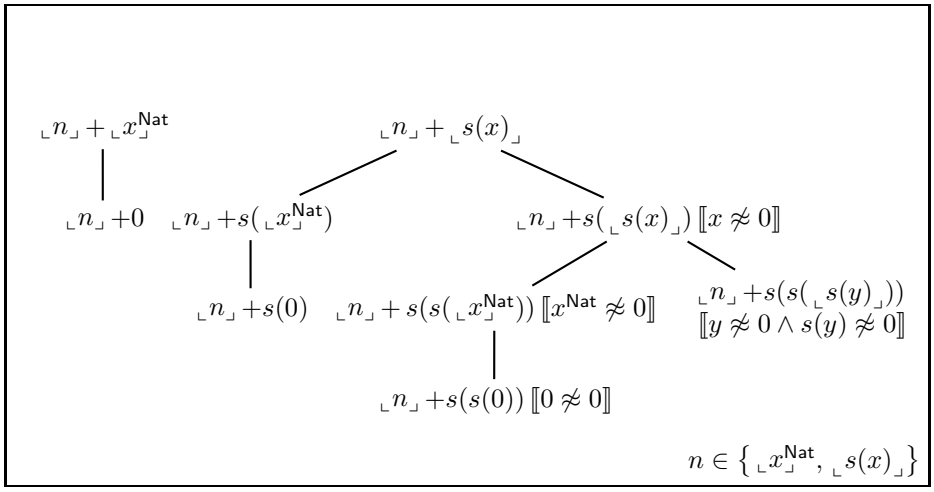

Figure 2: Sufficient completeness of $+$

Let us consider one interesting subtree of *dtree*$(+)$ with the root: $x_1 + y_1\,[\![x_1 : \llcorner n \lrcorner \wedge y_1 : \llcorner s(y) \lrcorner]\!]$, where $\llcorner n \lrcorner$ is any non-terminal. The application of the production rules of the normal form grammar instantiates this term into: $x_1 + s(y_1)\,[\![x_1 : \llcorner n \lrcorner \wedge y_1 : \llcorner x \lrcorner^{\mathsf{Nat}}]\!]$ and $x_1 + s(y_1)\,[\![x_1 : \llcorner n \lrcorner \wedge y_1 : \llcorner s(y) \lrcorner \wedge y \not\approx 0]\!]$. The first term is further instantiated into $x_1 + s(0)\,[\![x_1 : \llcorner n \lrcorner]\!]$ which is strongly ground reducible. The second one is instantiated into $x_1 + s(s(0))\,[\![x_1 : \llcorner u \lrcorner \wedge 0 \not\approx 0]\!]$ whose constraint is not valid (hence it is also strongly ground reducible). The second term is also instantiated into $x_1 + s(s(y))\,[\![x_1 : \llcorner u \lrcorner \wedge y : \llcorner s(x) \lrcorner \wedge x \not\approx 0 \wedge y \not\approx 0]\!]$. This latter term is reducible by $\mathcal{R}_\mathcal{C}$ hence strongly ground reducible.

6. Verification of Sufficient Completeness

In this section, we define formally the procedure for the verification of sufficient completeness of conditional and constrained rewrite systems (Section 6.2)

which was informally described in Section 5. We prove its correctness and completeness (Sections 6.4 and 6.5). This procedure relies on the framework [2] for inductive theorem proving described in Section 3.3. A decidable subcase is presented in Section 7, and the problem of the hypotheses about termination and confluence is discussed in Section 8.

6.1. 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 set of non-terminals of the normal form grammar whose construction is presented in Section 3.2.

6.2. Inference rules for sufficient completeness

The successors of any internal node in $dtree(f)$ are determined by the inference rules described in Figure 3. They follow 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 4. *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. Our procedure is defined by the inference rules given Figure 3.

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 $t\,[\![c]\!]$. The restriction to induction variables ensures the finiteness of the pattern tree, as shown in Theorem 1.

Strongly Ground Reducible Leaf: following Definition 3, this inference checks, for all rules whose left-hand side matches t, (i) the validity of some constraints and (ii) the validity of some inductive theorems, roughly a disjunction of conditions of some rewrite rules of $\mathcal{R}_{\mathcal{D}}$. The verification of point (ii) works by discharging proof obligations of inductive theorems to the procedure of [2], which is also based on $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$. The calls to [2] may be non terminating (inductive validity is not decidable), in this case, it is possible to make it converge by adding some

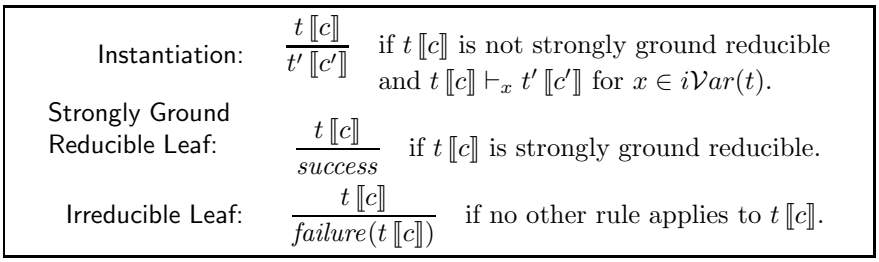

Figure 3: Inference rules for the construction of a pattern tree.

lemmas. However, the number of calls to the procedure of [2] is bounded by the number of leafs in the pattern trees, which is itself bounded, see Theorem 1.

Irreducible Leaf produces a failure when none of the two above inferences applies to a leaf $t\,[\![c]\!]$. This means in this case that the symbol f is not sufficiently complete *wrt* $\mathcal{R}$. The term $t\,[\![c]\!]$ provides a 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 strong ground reducibility test.

6.3. Finiteness of the pattern tree

The size of the pattern tree constructed is always finite.

Theorem 1. *For every CCTRS $\mathcal{R}$ and $f \in \mathcal{D}$, the size of dtree(f) is bounded.*

Proof. It follows from the finiteness of $i\mathcal{P}os(f, \mathcal{R})$. The number of rules of $\mathcal{R}_\mathcal{D}$ with the function symbol f at the top position is finite. This follows from the fact 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 is bounded too, since consecutive grafts in the same branch of the tree are labeled with deeper non-ground constrained terms. □

It follows from Theorem 1 that the termination of the procedure relies only on the test of strong ground reducibility. This property, which depends on inductive validity, is not decidable in general and, as explained above, its proof may require some user interaction. A consequence of Theorem 1 is discussed in Section 7 where a decidable subcase is identified.

6.4. Soundness

The following theorem states the *soundness* of our procedure.

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

PROOF. The key point of the proof 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 reducible by $\mathcal{R}_\mathcal{D}$.

Assume that for all $f \in \mathcal{D}$, all the leaves of $dtree(f)$ are labeled with *success*. We show that f is sufficiently complete wrt $\mathcal{R}$, i.e. that for all $t = f(t_1, \ldots, t_m)$ with $t_1, \ldots, t_m \in \mathcal{T}(\mathcal{C})$, there exists $u \in \mathcal{T}(\mathcal{C})$ such that $t \xrightarrow[\mathcal{R}]{*} u$.

Let us first show that every such t is reducible by $\mathcal{R}$. Since, by hypothesis, $\mathcal{R}_\mathcal{C}$ is terminating, we may consider that $t_1 \ldots, t_m$ are irreducible by $\mathcal{R}_\mathcal{C}$ (otherwise, they can be normalized under $\xrightarrow[\mathcal{R}_\mathcal{C}]{}$). By Proposition 1, it follows that there exists 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 $dtree(f)$. 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 Proposition 1). Since by hypothesis, all the leaves of $dtree(f)$ are labeled with *success*, $s\,[\![d]\!]$ is strongly ground reducible by $\mathcal{R}$. Since $\mathcal{R}$ is ground confluent, it follows by Lemma 2 that t is reducible by $\mathcal{R}$.

We show now that $t \xrightarrow[\mathcal{R}]{*} u \in \mathcal{T}(\mathcal{C})$ by induction based on the transitive closure of the union of $\xrightarrow[\mathcal{R}]{}$ (this is a well-founded relation by hypothesis) and the subterm relation.

The base case of the induction corresponds to t being irreducible by $\mathcal{R}$, and as we have seen above, this never occurs.

For the induction step, we use the above fact that t is reducible by $\mathcal{R}$, say $t \xrightarrow[\mathcal{R}]{} t'$. If $t' \in \mathcal{T}(\mathcal{C})$, then we are done. Otherwise, we apply the induction hypothesis to every maximal (*wrt* the subterm ordering) subterm of t' headed by a defined symbol. □

Since there are only two kinds of leaves, we can state as a corollary the *refutational completeness* of our procedure, *i.e.* that if $\mathcal{R}$ is not sufficiently complete, then the inference system will end with a failure.

Corollary 1. *Let $\mathcal{R}$ be a ground convergent CCTRS. 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.*

When the rules of the system $\mathcal{R}$ contains no conditions (but possibly some constraints), then the assumption of ground confluence in the Soundness Theorem 2 can be dropped, and the assumption of termination can be weaken to the termination of $\mathcal{R}_\mathcal{D}$ only.

Theorem 3. *Let $\mathcal{R}$ be an unconditional and constrained TRS such that $\mathcal{R}_\mathcal{D}$ is terminating. If for all $f \in \mathcal{D}$, all leaves of $dtree(f)$ are success then $\mathcal{R}$ is sufficiently complete.*

PROOF. We use the same schema as in the proof of Theorem 2, except that we use Lemma 1 instead of Lemma 2. Note that, the assumption that $\mathcal{R}$ is ground confluent is not needed here since this TRS is unconditional. □

The following corollary establishes refutational completeness for the same constrained and unconditional case.

Corollary 2 (Refutational Completeness). *Let $\mathcal{R}$ be an unconditional and constrained TRS such that $\mathcal{R}_\mathcal{D}$ is terminating. 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.*

6.5. Completeness

The following theorem establishes the *completeness* of the inference system in Figure 3. Note that it assumes no restriction on the CCTRS $\mathcal{R}$.

Theorem 4. *Let $\mathcal{R}$ be a CCTRS. If $\mathcal{R}$ is sufficiently complete then for each $f \in \mathcal{D}$, all leaves of dtree(f) are success.*

Proof. We show that the existence of a non-strongly ground reducible term in a leaf of *dtree*(f) contradicts the sufficient completeness of $\mathcal{R}$.

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. This means, by definition, that $t \,[\![c]\!]$ does not contain any induction variable and is not strongly ground reducible. We show first that $t \,[\![c]\!]$ contains a subterm which is an instance of a left-hand side of a rule of $\mathcal{R}$.

By construction, $t \,[\![c]\!]$ is decorated, and since $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ is clean (for every non-terminal $\llcorner u \lrcorner \in Q_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$, the language $L\big(\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C}), \llcorner u \lrcorner\big)$ is not empty), there exists $\tau \in sol(c)$ such that for all $x \in var(t)$, $x\tau$ is irreducible by $\mathcal{R}$. 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 irreducible by $\mathcal{R}_\mathcal{C}$. Hence, $t\tau$ is reducible at the root position by $\mathcal{R}_\mathcal{D}$ 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:

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

By hypothesis, $t \,[\![c]\!]$ is not strongly ground reducible by $\mathcal{R}$. This means that at least one 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 $\Gamma \Rightarrow \ell \to r \,[\![d]\!] \in \mathcal{R} \setminus \mathcal{L}$. This 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 irreducible by $\mathcal{R}_\mathcal{C}$. This contradicts the hypothesis that $\mathcal{R}$ is sufficiently complete. □

As a corollary, we conclude the *soundness of disproof* with the procedure: if the inference system fails then $\mathcal{R}$ is not sufficiently complete.

Corollary 3. *Let $\mathcal{R}$ be a CCTRS. 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 $\mathcal{R}$.

7. Decidable case

Sufficient completeness is undecidable for CCTRS in general. Strong ground reducibility, required by the inference Strongly Ground Reducible Leaf, is neither a decidable property, since it relies on the proof of inductive theorems, discharged to the procedure of [2]. The inductive theorems to prove are roughly disjunctions of conditions of rules of $\mathcal{R}_\mathcal{D}$.

When $\mathcal{R}$ is unconditional (but constrained), testing strong ground reducibility (Definition 3) of a constrained term in a pattern tree amounts to do pattern matching with left-hand side of rules of $\mathcal{R}_\mathcal{D}$ and checking validity of constraints. It follows that in the unconditional case, the decision of strong 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).

According to Theorem 1 (finiteness of the pattern tree constructed), the decidability on the unconditional case can be obtained provided that the emptiness problem for the class of tree grammar to which $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_\mathcal{C})$ belongs is decidable. We won't detail the reduction to the emptiness problem here (it is already given, in a similar context, in [2] – Section 6), but rather summarize in the following theorem the conditions ensuring decidability.

Theorem 5. *Sufficient completeness is decidable when $\mathcal{R}$ is an unconditional and constrained TRS, $\mathcal{R}_\mathcal{D}$ is terminating, and $\mathcal{R}$ 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 $s \approx s' \in c$, (resp. all $s \not\approx s' \in c$) s and s' are either variable or strict subterms of l (resp. variables or strict subterms occurring at sibling positions in l).*

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 Cartesian product construction.

8. On ground confluence and termination

The above soundness theorems (Theorems 2 and 3) assume respectively the ground convergence of the CCTRS $\mathcal{R}$ and the termination of $\mathcal{R}_\mathcal{D}$ (providing

that $\mathcal{R}$ is an unconditional and constrained TRS). We discuss in this subsection how these properties can be established.

Ground confluence.. This property guarantees the uniqueness of computations with ground terms. Several works have proposed sufficiency criteria for checking confluence of conditional systems [13, 25]. Ground confluence is undecidable [27] even for equational theories with only unary function symbols. Let us propose below the big lines of two approaches that could be followed in order to prove ground confluence for conditional and constrained rewrite systems in our framework.

A first approach could use the method developed in [32] for checking ground confluence of conditional theories. This technique does not rely on the completion framework. The key idea of this method is to compute all critical pairs between axioms, and then to check each critical pair w.r.t. a sufficient criterion for ground confluence.

In order to use this method in our framework, where rewrite rules are conditional and constrained, we can start by transforming constraints in the rewrite rules into conditions. With appropriate axiomatizations of the constraints (they exists for the constraints used in this paper, namely equalities, disequalities and ordering), this can be performed automatically using a simple syntactical transformation.

Example 10. For example, the following rule for specifying insertion of elements in integer lists (see Section 9)

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

can be transformed into:

$$\begin{array}{rcl} ins(x, ins(x, z)) & \to & ins(x, z), \\ x > y = true \Rightarrow ins(x, ins(y, z)) & \to & ins(y, ins(x, z)) \end{array}$$

where $x > y$ is specified by the following (unconditional) rewrite rules:

$$\begin{array}{rcl} x > 0 & \to & true \\ 0 > x & \to & false \\ s(x) > s(y) & \to & x > y. \end{array}$$

$\diamond$

A second approach would be to use the completion technique proposed in [15] for checking ground confluence of parametric conditional equational specifications. The idea here would be to consider predicates like the above $>$ as parameter functions.

Termination.. Many tools have been developed for automating the proof of *termination* of rewrite systems, and can be used prior to sufficient completeness checking with our procedure. Amongst these systems, let us cite CiME [11], AProVE [16], T$_{\text{T}}$T$_2$ [30], MU-TERM [34] and the Maude Termination Tool [14].

Note that these tools do not support constrained rewrite rules. Therefore, in order to use these systems for checking termination of rewrite systems with conditional and constrained rules, we must, in a preliminary step, transform the constraints in the rewrite rules into conditions, as suggested above.

9. More Examples: Sorted Lists and Powerlists

9.1. Sorted lists

Let us consider the specification of sorted lists without repetition started in Examples 2 and 4. Recall that it is based on the constructor symbols $\mathit{true}, \mathit{false} : \mathsf{Bool}$, $0 : \mathsf{Nat}$, $s : \mathsf{Nat} \to \mathsf{Nat}$, $\emptyset : \mathsf{List}$, $\mathit{ins} : \mathsf{Nat} \times \mathsf{List} \to \mathsf{List}$, and that

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

Note that $\mathcal{R}_{\mathcal{C}}$ is terminating thanks to the constraint of the second rule (the ordering $\succ$ is assumed total on ground terms).

The constrained tree grammar $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$ is the one of Examples 2, extended with two production rules $\llcorner x^{\mathsf{Bool}} \lrcorner := \mathit{true} \mid \mathit{false}$.

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

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

9.2. Sufficient completeness of $\in$

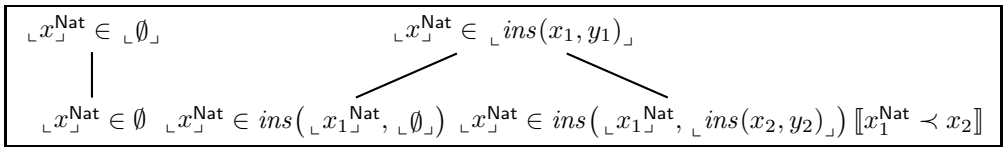

Figure 4: The pattern tree of $\in$

The pattern tree $\mathit{dtree}(\in)$ given in Figure 4 shows that the function $\in$ is sufficiently complete. The term $x^{\mathsf{Nat}} \in \emptyset$ is indeed reducible. The two other leaves are also strongly ground reducible since $x_1 \approx x_2 \vee x_1 \not\approx x_2$ (the disjunction of the constraints of the second and third rules of $\mathcal{R}_{\mathcal{D}}$) is valid.

9.3. Sufficient completeness of sorted

The pattern tree of Figure 5 shows that *sorted* is not sufficiently complete. It contains two *failure* leaves labeled respectively with $sorted(\emptyset)$ and $sorted(ins(x^{\mathsf{Nat}}, \emptyset))$ (we drop the tag *failure* of the inference Irreducible Leaf of Figure 3, for the sake of readability). The reason is that these terms do not contain induction variables and that moreover they are not strongly ground reducible because they do not match a left-hand side of a rule of $\mathcal{R}_{\mathcal{D}}$. This suggests to complete $\mathcal{R}_{\mathcal{D}}$ with two rules $sorted(\emptyset) \to true$ and $sorted(ins(x, \emptyset)) \to true$. We show below that the system obtained is sufficiently complete. Indeed,

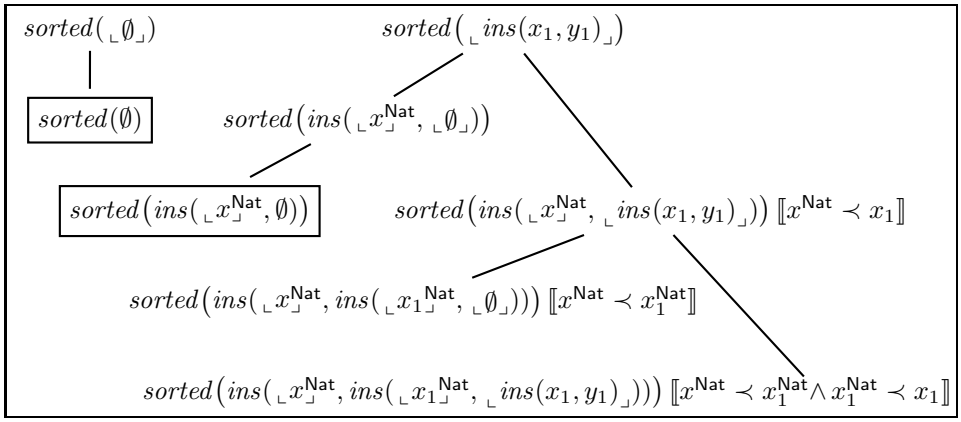

Figure 5: The pattern tree of *sorted* with two *failure* leaves.

$sorted(\emptyset)$ and $sorted(ins(x^{\mathsf{Nat}}, \emptyset))$ are now both reducible by the new rewrite rules. Moreover, the term: $sorted(ins(\llcorner x\lrcorner^{\mathsf{Nat}}, ins(\llcorner x_1\lrcorner^{\mathsf{Nat}}, \llcorner\emptyset\lrcorner))) \; [\![x^{\mathsf{Nat}} \prec x_1^{\mathsf{Nat}}]\!]$, which is an abbreviation for:

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

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

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

are strongly ground reducible since the two following conjectures are inductive theorems of $\mathcal{R}$, and can be proved using the method of [2]:

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

9.4. A more involved proof

Let us consider an alternative specification of a membership operator in $\mathcal{D}$: $\in'$: $\mathsf{Nat} \times \mathsf{List} \to \mathsf{Bool}$, with the rules of $\mathcal{R}_\mathcal{D}$:

$$\begin{array}{rcll} x \in' \emptyset & \to & \mathit{false} & \\ x_1 \in' \mathit{ins}(x_2, y) & \to & \mathit{true} & [\![x_1 \approx x_2]\!] \\ x_1 \in' \mathit{ins}(x_2, y) & \to & \mathit{false} & [\![x_1 \prec x_2, \mathit{ins}(x_2, y) : {\llcorner}\mathit{ins}(x, y){\lrcorner}]\!] \\ x_1 \in' \mathit{ins}(x_2, y) & \to & x_1 \in' y & [\![x_1 \succ x_2]\!] \end{array}$$

Note that in the above third rule, the term $\mathit{ins}(x_2, y)$ is constrained to belong to the language generated by the non-terminal ${\llcorner}\mathit{ins}(x, y){\lrcorner}$. This means that this term must be a ground $\mathcal{R}_\mathcal{C}$-normal form. The proof of the sufficient completeness of $\in'$ is very similar to the above proof for $\in$.

Finally, let us consider a symbol of $\mathcal{D}$, $\mathit{co} : \mathsf{Nat} \times \mathsf{List} \to \mathsf{Bool}$, which will be sufficiently complete iff $\in'$ coincide with $\in$, according to the following rule of $\mathcal{R}_\mathcal{D}$:

$$x \in' y = x \in y \Rightarrow \mathit{co}(x, y) \to \mathit{true}$$

In order to prove that the function co is sufficiently complete, we show in appendix that $\mathcal{R} \models_{\mathcal{I}nd} x \in' y = x \in y$, using the method of [2].

9.5. Powerlists

Powerlists [35] 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:

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

The symbol v creates a singleton powerlist $v(n)$ containing a number n, and *tie* is the concatenation of powerlists. The operator *tie* is restricted to well balanced constructor terms of $\mathcal{T}(\mathcal{C} \setminus \{\bot\})$ of the same depth. Every other term of the form $\mathit{tie}(s, t)$ is reduced to $\bot$ by the following constructor system $\mathcal{R}_\mathcal{C}$. Therefore, the well-formed powerlists are ground terms of sort List irreducible by $\mathcal{R}_\mathcal{C}$.

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 $\mathit{tie}(x_1, x_2) \sim \mathit{tie}(y_1, y_2)$ iff $x_1 \sim x_2 \sim y_1 \sim y_2$. Note in particular that $\bot$ is equivalent by $\sim$ to any other constructor term.

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

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

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

$${\llcorner}x{\lrcorner}^{\mathsf{Nat}} := 0 \qquad {\llcorner}x{\lrcorner}^{\mathsf{Nat}} := s({\llcorner}x_2{\lrcorner}^{\mathsf{Nat}})$$
$${\llcorner}x{\lrcorner}^{\mathsf{List}} := v({\llcorner}x{\lrcorner}^{\mathsf{Nat}}) \qquad {\llcorner}x{\lrcorner}^{\mathsf{List}} := \mathit{tie}({\llcorner}x_1{\lrcorner}^{\mathsf{List}}, {\llcorner}x_2{\lrcorner}^{\mathsf{List}}) \, [\![x_1^{\mathsf{List}} \sim x_2^{\mathsf{List}}]\!] \qquad {\llcorner}x{\lrcorner}^{\mathsf{List}} := \bot$$

Note that all the constraints in these production rules are applied to brother subterms. The emptiness problem can be shown decidable for such constrained tree grammars, with an adaptation of the similar proof for tree automata with equality and disequality constraints between subterms [7], or alternatively with an encoding into tree automata with one memory [9].

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

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

The sufficient completeness of *zip* can be established with the pattern tree construction. This means in particular that this operator is defined on all well-formed powerlists.

The subtree of $dtree(zip)$ with root $zip(\llcorner x_1 \lrcorner^{\mathsf{List}}, \llcorner x_2 \lrcorner^{\mathsf{List}})$ has nodes (without induction variables) of the form

$zip(v(\llcorner x_1 \lrcorner^{\mathsf{Nat}}), v(\llcorner x_2 \lrcorner^{\mathsf{Nat}}))$,

$zip(v(\llcorner x_1 \lrcorner^{\mathsf{Nat}}), tie(\llcorner x_2 \lrcorner^{\mathsf{List}}, \llcorner x_3 \lrcorner^{\mathsf{List}}))\, [\![x_2^{\mathsf{List}} \sim x_3^{\mathsf{List}}]\!]$,

$zip(tie(\llcorner x_1 \lrcorner^{\mathsf{List}}, \llcorner x_2 \lrcorner^{\mathsf{List}}), v(\llcorner x_3 \lrcorner^{\mathsf{Nat}}))\, [\![x_1^{\mathsf{List}} \sim x_2^{\mathsf{List}}]\!]$, and

$zip(tie(\llcorner x_1 \lrcorner^{\mathsf{List}}, \llcorner x_2 \lrcorner^{\mathsf{List}}), tie(\llcorner x_3 \lrcorner^{\mathsf{List}}, \llcorner x_4 \lrcorner^{\mathsf{List}}))\, [\![x_1^{\mathsf{List}} \sim x_2^{\mathsf{List}}, x_3^{\mathsf{List}} \sim x_4^{\mathsf{List}}]\!]$

which are all strongly ground reducible. Hence all the corresponding leaves are labelled with *success*. This subtree also contains nodes of the form $zip(\bot, t)$ or $zip(t, \bot)$ which are reducible by $\mathcal{R}_{\mathcal{D}}$ (hence strongly ground reducible).

9.6. Sufficient completeness proofs with other methods

The methods of [1, 3, 4] cannot be applied to prove the sufficient completeness of $\in$, 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 $\{\emptyset, ins(x, \emptyset), ins(x, ins(y, z))\, [\![y \succ x]\!]\}$. This also fails. The reason is that this representation of ground constructor terms irreducible by $\mathcal{R}_{\mathcal{C}}$ is still not exact. For example $ins(0, ins(s(0), ins(0, \emptyset))$ is an instance of $ins(x, ins(y, z))\, [\![y \succ x]\!]$ but is not irreducible.

The Maude sufficient completeness checker has been successfully used for powerlists [21]. For checking the sufficient completeness of *co* it generates a proof obligation which cannot be proved automatically by Maude's inductive theorem prover and therefore must be manually discharged by the user[1].

[1] Personal communication of Joe Hendrix.

10. 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 [2]. It is sound for ground convergent CCTRS and also complete modulo the above oracle for proving inductive theorems. We show that it is a decision procedure for unconditional and constrained TRS wrt a large class of constrained constructor axioms.

It has been successfully used 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.

As constrained tree grammar serve as a parameter in the procedure, future progress in decision procedures for classes of tree automata with constraints will permit to extend the languages of specifications and constraints handled.

Acknowledgements.

We wish to sincerely thank Joe Hendrix for having processed the above examples of Section 5 and 9 with the *Sufficient Completeness Checker* for Maude, and the reviewers whose useful remarks helped to improve the presentation of the paper.

References

[1] A. Bouhoula. Using induction and rewriting to verify and complete parameterized specifications. *Theoretical Computer Science*, 170(1-2):245–276, 1996.

[2] A. Bouhoula and F. Jacquemard. Automated Induction with Constrained Tree Automata. Proc. of the 4th International Joint Conference on Automated Reasoning (IJCAR), vol. 5195 of Lecture Notes in Computer Science, pages 539–554, Springer-Verlag, 2008.

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

[4] A. Bouhoula, J.-P. Jouannaud, and J. Meseguer. Specification and proof in membership equational logic. *Theoretical Computer Science*, 236(1-2):35–132, 2000.

[5] A. Bouhoula and M. Rusinowitch. Implicit induction in conditional theories. *Journal of Automated Reasoning*, 1995.

[6] M. Dauchet, A.-C Caron and J.-L. Coquidé. Automata for Reduction Properties Solving Journal of Symbolic Computation, 20(2), pages 215–233, 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. Sufficient completeness, term rewriting and anti-unification. In *Proc. of the 8th International Conference on Automated Deduction (CADE)*, vol. 230 of Springer LNCS, pages 128–140, 1986.

[9] H. Comon and V. Cortier. Tree automata with one memory, set constraints and cryptographic protocols. Theoretical Computer Science, 331(1), pages 143-214, Elsevier, 2005.

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

[11] E. Contejean, C. March, B. Monate, and X. Urbain. Proving termination of rewriting with cime. In ext. abstracts of the 6th Int. Workshop on Termination, WST'03, pages 71-73, 2003. see also `http://cime.lri.fr`

[12] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In *Handbook of Theoretical Computer Science, vol. B: Formal Models and Sematics*, pages 243–320. 1990.

[13] N. Dershowitz, M. Okada, and G. Sivakumar. Confluence of conditional rewrite systems. In Proc. 1st International Workshop on Conditional Term Rewriting Systems, vol. 308 of Lecture Notes in Computer Science, Springer-Verlag, pages 31-44, 1987.

[14] F. Durán, S. Lucas and J. Meseguer MTT: The Maude Termination Tool (System Description). In *Proc. of the 4th International Joint Conference on Automated Reasoning (IJCAR 08)*, vol. 5195 of Springer LNCS, pages 313-319, 2008.

[15] H. Ganzinger. Ground term confluence in parametric conditional equational specifications. In Proc. STACS 87, vol. 247 of Lecture Notes in Computer Science, Springer-Verlag, pages 286-298, 1987.

[16] J. Giesl, P. Schneider-Kamp, and R. Thiemann AProVE 1.2: Automatic Termination Proofs in the Dependency Pair Framework In Proc. of the third International Joint Conference on Automated Reasoning (IJCAR '06), vol. 4130 of Lecture Notes in Artificial Intelligence, Springer-Verlag, pages 281-286, 2006.

[17] I. Gnaedig, and H. Kirchner. Computing constructor forms with non terminating rewrite programs. In *Proc. of the 8th ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming* (PPDP '06). ACM Press, pages 121-132, 2006.

[18] J. V. Guttag. *The Specification and Application to Programming of Abstracta Data Types*. Phd thesis, Univ. of Toronto, 1975. Report CSRG-59.

[19] J. V. Guttag. Notes on type abstraction. In *Program Construction*, vol. 69 of LNCS, pages 593–616. Springer, 1978.

[20] J. V. Guttag and J. J. Horning. The algebraic specification of abstract data types. *Acta Informatica*, 10:27–52, 1978.

[21] 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.

[22] 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), Univ. of Illinois at Urbana-Champaign, 2005.

[23] G. Huet and J.-M. Hullot. Proofs by induction in equational theories with constructors. *J. Comput. Syst. Sci.*, 25(2):239–266, 1982.

[24] J.-P. Jouannaud and E. Kounalis. Automatic Proofs by Induction in Equational Theories Without Constructors. In *Proc. of the 1st Symposium on Logic in Computer Science (LICS)*, IEEE Computer Society, pages 358-366, 1986.

[25] S. Kaplan. Simplifying conditional term rewriting systems: Unification, termination and confluence. Journal of Symbolic Computation, 4(3):295-334, 1987.

[26] D. Kapur. An automated tool for analyzing completeness of equational specifications. In *Proc. of the Int. Symp. on Software Testing and Analysis (ISSTA)*, special Issue of *Software Engineering Notes*, pages 28–43. ACM Press, 1994.

[27] D. Kapur, P. Narendan, and F. Otto. On ground confluence of term rewriting systems. Information and Computation, 86(1):14-31, 1990.

[28] D. Kapur, P. Narendran, and H. Zhang. On sufficient-completeness and related properties of term rewriting systems. *Acta Informatica*, 24(4):395–415, 1987.

[29] D. Kapur, P. Narendran, D. J. Rosenkrantz, and H. Zhang. Sufficient-completeness, ground-reducibility and their complexity. *Acta Informatica*, 28(4):311–350, 1991.

[30] M. Korp, C. Sternagel, H. Zankl, and A. Middeldorp. Tyrolean Termination Tool 2. In *Proc. of the Int. Conf. on Rewriting Techniques and Applications (RTA)*, vol. 5595 of Lecture Notes in Computer Science, Springer, pages 295-304, 2009. see also `http://colo6-c703.uibk.ac.at/ttt2`

[31] E. Kounalis. Completeness in data type specifications. In *European Conference on Computer Algebra (2)*, vol. 204 of LNCS, pages 348–362. Springer, 1985.

[32] E. Kounalis and M. Rusinowitch. Studies on the ground convergence property of conditional theories. In Proc. AMAST Workshop in Computing, pages 363-376, Springer-Verlag, 1992.

[33] 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.

[34] S. Lucas. MU-TERM: A Tool for Proving Termination of Context-Sensitive Rewriting. In *Proc. of 15th Int. Conf. on Rewriting Techniques and Applications (RTA) 2004*, vol. 3091 of Springer LNCS, pages 200-209, 2004. see also `http://users.dsic.upv.es/~slucas/csr/termination/muterm`

[35] J. Misra. Powerlist: A structure for parallel recursion. *ACM Transactions on Programming Languages and Systems*, 16(6):1737–1767, 1994.

Appendix A.

In order to complete the proof that the function *co* is sufficiently complete in Section 9.4 we show here that $\mathcal{R} \models_{\mathcal{I}nd} x \in' y = x \in y$, using the method of [2]. Recall that this procedure is also based on the constrained tree grammar $\mathcal{G}_{\rm NF}(\mathcal{R}_{\mathcal{C}})$ of Examples 2 (extended with two production rules $\llcorner x^{\mathsf{Bool}} \lrcorner := \mathit{true}$ and $\llcorner x^{\mathsf{Bool}} \lrcorner := \mathit{false}$).

Let us constrain the variable y in the above conjecture to the language of non-terminals of the grammar $\mathcal{G}_{\rm NF}(\mathcal{R}_{\mathcal{C}})$:

$$x \in' \llcorner \emptyset \lrcorner = x \in \llcorner \emptyset \lrcorner \tag{A.1}$$

$$x \in' \llcorner \mathit{ins}(x_1, y_1) \lrcorner = x \in \llcorner \mathit{ins}(x_1, y_1) \lrcorner \tag{A.2}$$

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

$$x \in' \emptyset = x \in \emptyset \tag{A.3}$$

$$x \in' \mathit{ins}(\llcorner x^{\mathsf{Nat}} \lrcorner, \emptyset) = x \in \mathit{ins}(\llcorner x^{\mathsf{Nat}} \lrcorner, \emptyset) \tag{A.4}$$

$$x \in' \mathit{ins}(\llcorner x^{\mathsf{Nat}} \lrcorner, \llcorner \mathit{ins}(x_1, y_1) \lrcorner) = x \in \mathit{ins}(\llcorner x^{\mathsf{Nat}} \lrcorner, \llcorner \mathit{ins}(x_1, y_1) \lrcorner) \, [\![x^{\mathsf{Nat}} \prec x_1]\!] \tag{A.5}$$

The clause (A.3) can be reduced by $\mathcal{R}_{\mathcal{D}}$ to the tautology $\mathit{false} = \mathit{false}$. For (A.4) 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 $x_1 \in' \mathit{ins}(x_2, y)$ as left member). This technique is called *Rewrite Splitting* in [2], it returns:

$$\mathit{true} \;=\; x \in \mathit{ins}(\llcorner x^{\mathsf{Nat}} \lrcorner, \emptyset) \, [\![x \approx x^{\mathsf{Nat}}]\!] \tag{A.6}$$

$$\mathit{false} \;=\; x \in \mathit{ins}(\llcorner x^{\mathsf{Nat}} \lrcorner, \emptyset) \, [\![x \prec x^{\mathsf{Nat}}]\!] \tag{A.7}$$

$$x \in' \emptyset \;=\; x \in \mathit{ins}(\llcorner x^{\mathsf{Nat}} \lrcorner, \emptyset) \, [\![x \succ x^{\mathsf{Nat}}]\!] \tag{A.8}$$

All these subgoal are reduced by $\mathcal{R}_{\mathcal{D}}$ into tautologies $\mathit{true} = \mathit{true}$ or $\mathit{false} = \mathit{false}$. Similarly, the application of *Rewrite Splitting* to (A.5) returns:

$$\mathit{true} = x \in \mathit{ins}(\llcorner x^{\mathsf{Nat}} \lrcorner, \llcorner \mathit{ins}(x_1, y_1) \lrcorner) \, [\![x^{\mathsf{Nat}} \prec x_1, x \approx x^{\mathsf{Nat}}]\!] \tag{A.9}$$

$$\mathit{false} = x \in \mathit{ins}(\llcorner x^{\mathsf{Nat}} \lrcorner, \llcorner \mathit{ins}(x_1, y_1) \lrcorner) \, [\![x^{\mathsf{Nat}} \prec x_1, x \prec x^{\mathsf{Nat}}]\!] \tag{A.10}$$

$$x \in' \llcorner \mathit{ins}(x_1, y_1) \lrcorner = x \in \mathit{ins}(\llcorner x^{\mathsf{Nat}} \lrcorner, \llcorner \mathit{ins}(x_1, y_1) \lrcorner) \, [\![x^{\mathsf{Nat}} \prec x_1, x \succ x^{\mathsf{Nat}}]\!] \tag{A.11}$$

The subgoal (A.9) is reduced by the second rule $\mathcal{R}_{\mathcal{D}}$ for $\in$ (the one with an equality constraint) into the tautology $\mathit{true} = \mathit{true}$. The clause (A.10) is simplified by Rewrite Splitting with the constrained rules of $\mathcal{R}_{\mathcal{D}}$ for $\in$, into:

$$\mathit{false} = \mathit{true} \, [\![x^{\mathsf{Nat}} \prec x_1, x \prec x^{\mathsf{Nat}}, x \approx x^{\mathsf{Nat}}]\!] \tag{A.12}$$

$$\mathit{false} = x \in \llcorner \mathit{ins}(x_1, y_1) \lrcorner \, [\![x^{\mathsf{Nat}} \prec x_1, x \prec x^{\mathsf{Nat}}, x \not\approx x^{\mathsf{Nat}}]\!] \tag{A.13}$$

The subgoal (A.12) is valid since its constraint is unsatisfiable. The clause (A.13) 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):

$$\mathit{false} = x \in \mathit{ins}(\llcorner x_2 \lrcorner^{\mathsf{Nat}}, \emptyset)\, [\![x^{\mathsf{Nat}} \prec x_2^{\mathsf{Nat}}, x \prec x^{\mathsf{Nat}}, x \not\approx x^{\mathsf{Nat}}]\!] \tag{A.14}$$

$$\mathit{false} = x \in \mathit{ins}(\llcorner x_2 \lrcorner^{\mathsf{Nat}}, \llcorner \mathit{ins}(x_2, y_2) \lrcorner)\, [\![x^{\mathsf{Nat}} \prec x_2^{\mathsf{Nat}}, x \prec x^{\mathsf{Nat}}, x \not\approx x^{\mathsf{Nat}}, x_2^{\mathsf{Nat}} \prec x_2]\!] \tag{A.15}$$

Note that, thanks to the constraints in the production rules of $\mathcal{G}_{\mathrm{NF}}(\mathcal{R}_{\mathcal{C}})$, the constraint of (A.15) implies that both $x_2^{\mathsf{Nat}} \prec x_2$ and $x \prec x_2^{\mathsf{Nat}}$.

The clause (A.14) can be reduced by $\mathcal{R}_{\mathcal{D}}$ to the tautology $\mathit{false} = \mathit{false}$. The clause (A.15) can be reduced to the same tautology using the clause $(A.10)$, which is used in this case as an induction hypothesis.

Let us come back to the subgoal (A.11). The application of Rewrite Splitting (again with the constrained rules of $\mathcal{R}_{\mathcal{D}}$ for $\in$) returns:

$$x \in' \llcorner \mathit{ins}(x_1, y_1) \lrcorner = \mathit{true}\, [\![x^{\mathsf{Nat}} \prec x_1, x \succ x^{\mathsf{Nat}}, x \approx x^{\mathsf{Nat}}]\!] \tag{A.16}$$

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

The subgoal (A.16) is valid since its constraint is unsatisfiable. The last subgoal (A.17) is reduced by application of (A.2) (used as induction hypothesis) into the tautology:

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