HAL
archives-ouvertes.fr

Computing Constructor Forms with Non Terminating Rewrite Programs

Isabelle Gnaedig, Hélène Kirchner

▶ To cite this version:

Isabelle Gnaedig, Hélène Kirchner. Computing Constructor Forms with Non Terminating Rewrite Programs. Symposium on Principles and Practice of Declarative Programming - PPDP'06, Jul 2006, Venise/Italie, ACM, Proceedings of the Eighth ACM-SIGPLAN International Symposium on Principles and Practice of Declarat, 2006. <inria-00112083>

HAL Id: inria-00112083
https://hal.inria.fr/inria-00112083

Submitted on 7 Nov 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Computing Constructor Forms with Non Terminating Rewrite Programs

Isabelle Gnaedig
LORIA-INRIA
615, rue du Jardin Botanique, BP 101,
F-54602 Villers-lès-Nancy Cedex
Isabelle.Gnaedig@loria.fr

Hélène Kirchner
LORIA-CNRS
Campus Scientifique, BP 239,
F-54506 Vandoeuvre-lès-Nancy Cedex
Helene.Kirchner@loria.fr

Abstract

In the context of the study of rule-based programming, we focus in this paper on the property of $\mathcal{C}$-reducibility, expressing that every term reduces to a constructor term on at least one of its rewriting derivations. This property implies completeness of function definitions, and enables to stop evaluations of a program on a constructor form, even if the program is not terminating. We propose an inductive procedure proving $\mathcal{C}$-reducibility of rewriting. The rewriting relation on ground terms is simulated through an abstraction mechanism and narrowing. The induction hypothesis allows assuming that terms smaller than the starting terms rewrite into a constructor term. The existence of the induction ordering is checked during the proof process, by ensuring satisfiability of ordering constraints. The proof is constructive, in the sense that the branch leading to a constructor term can be computed from the proof trees establishing $\mathcal{C}$-reducibility for every term.

Categories and Subject Descriptors F.3.1 [*LOGICS AND MEANINGS OF PROGRAMS*]: Specifying and Verifying and Reasoning about Programs—Logics of programs, Mechanical verification, Specification techniques; F.4.2 [*MATHEMATICAL LOGIC AND FORMAL LANGUAGES*]: Grammars and Other Rewriting Systems; F.4.3 [*MATHEMATICAL LOGIC AND FORMAL LANGUAGES*]: Formal Languages—Algebraic language theory; I.1.3 [*SYMBOLIC AND ALGEBRAIC MANIPULATION*]: Languages and Systems—Evaluation strategies, Substitution mechanisms; I.2.3 [*ARTIFICIAL INTELLIGENCE*]: Deduction and Theorem Proving—Deduction, Inference engines, Mathematical induction; D.3.1 [*PROGRAMMING LANGUAGES*]: Formal Definitions and Theory; D.2.4 [*SOFTWARE ENGINEERING*]: Software/Program Verification—Correctness proofs, Formal methods, Validation

General Terms Algorithms, Languages, Verification

Keywords Sufficient Completeness, Constructor, Abstraction, Narrowing, Ordering Constraint, Termination

1. Introducing the problem

Sufficient completeness plays an important role in algebraic specifications, as well as in rewriting-based programs. In both contexts, terms are built on operators, among them we can distinguish constructors. Constructors are basic symbols, allowing us to describe the values of computations. The other symbols, called defined symbols, represent functions defined on these values. One is naturally led to ask for a specification or a program, whether the functions are sufficiently defined to warrant that to every expression or term corresponds a value.

From the point of view of specifications, where properties are described by equations, sufficient completeness ensures that every term is equivalent to a term built on constructors, called constructor term or constructor form. It allows inductive proofs, in particular by consistency methods [7]. Proof assistants like Coq or PVS include decision or semi-decision procedures based on rewrite rules and rely on complete definitions of functions.

From the point of view of programming, sufficient completeness ensures that a program produces a completely computed form for every data. Studying the property in this context is pertinent since rule-based systems have recently gained considerable interest with the development of efficient compilers. Now, systems like ASF+SDF [25], Maude [5, 12], Cafe-OBJ [17], ELAN [1], or TOM [29], are used for various applications like constraint solving, protocol verification, modeling of biological or chemical systems, and more.

Proving sufficient completeness is undecidable in general. It has already been widely studied, for example in [21, 30, 26, 23, 6, 22, 28, 2, 3], but most of the time, the proposed approaches for proving the property need restrictions like termination and confluence.

The property is strongly related to ground reducibility, which expresses that every ground instance of a term is reducible. Indeed, it is equivalent to ground reducibility of all patterns $f(x_1, \ldots, x_m)$ built on a defined symbol f, provided the rewrite system (RS in short) is terminating, confluent, and the normal form of a constructor term is again a constructor term [22]. Under these conditions, techniques developed for proving ground reducibility hold for sufficient completeness as well [31, 24, 22, 27, 4, 8].

In this paper, we address the problem of sufficient completeness from the programming point of view, and we go beyond the previous usual restrictions.

We observe the case where a program or a RS can be neither confluent, nor terminating, and we study its evaluations. We do not suppose other restrictions used in the domain as the constructor preserving property, or the absence of relation between constructors. The question is to know whether at least one evaluation of a given data gives a completely evaluated result; in other words, whether for every ground term, there exists a rewriting chain that eventually reaches a constructor term, even if the chains do

not converge to a single term, and are infinite. We call this property $\mathcal{C}$-reducibility.

Since several years, we have been studying properties of rewriting in the context of programming, where a rewrite system interpreter or compiler uses a specific strategy for applying rewrite rules, most frequently an innermost strategy that consists in rewriting always at the lowest possible positions, and corresponds to a call-by-value evaluation of functions. We observe rewriting in considering its properties from an inductive point of view. We already have proposed proof procedures for the termination property, in the specific case of rewriting strategies, respectively for the innermost [20], the outermost [14] and local strategies [13]. We also have given a constructive proof technique for weak innermost termination, enabling to reach the terminating branches [15]. Here, $\mathcal{C}$-reducibility is tackled with a similar inductive approach.

It is not based on analyzing the patterns issued from the RS, as do most of existing methods for sufficient completeness, but on observing the rewriting process starting from any term to see whether it is eventually transformed into a constructor term. For that, like for our previous procedures for termination, we reason by induction on the property itself, assuming that the terms t smaller than the starting terms for an induction ordering are $\mathcal{C}$-reducible.

This approach needs a noetherian ordering on terms, used in the induction principle. In contrast to classical induction proofs where the ordering is given, we do not need to define it a priori. We only have to check its existence by ensuring satisfiability of ordering constraints incrementally set along the proof.

By alternatively applying an abstraction mechanism and well-covered narrowing steps, we generate proof trees representing at least one rewriting derivation for every ground term. The proof is constructive, in the sense that the rewriting branch leading to a constructor term can be computed from the proof trees establishing $\mathcal{C}$-reducibility for every term, thus avoiding costly breadth-first computations to reach the constructor form. Although our method differs from previous approaches, we naturally encounter basic notions already used for proving sufficient completeness, as unification, used in [28], or covering properties used in many works in the domain, for example in [26, 28]. In [3], pattern trees are also developed, but under the assumption that the RS is terminating and ground confluent.

Before we detail our procedure, we link $\mathcal{C}$-reducibility to close properties like strong $\mathcal{C}$-reducibility, expressing the existence of a constructor form on every rewriting chain, and their variants in the case of innermost rewriting. Sufficient completeness and ground reducibility are also considered in this comparison. In particular, $\mathcal{C}$-reducibility directly implies sufficient completeness. In addition, we justify that ground reducibility just requires weak termination (expressing that every term has at least one terminating rewriting derivation) to imply sufficient completeness, thus weakening the condition of the well-known theorem of [22]. We also note that as the covering property we require is stronger than ground reducibility, this property directly implies sufficient completeness if we suppose weak termination. Thus, with respect to the sufficient completeness property, the proof technique we propose is interesting:

- obviously when the RS is not even weakly terminating,
- when it is terminating, or just weakly terminating, but we have no technique to prove it. The weak termination proof is difficult to handle in general, and to our knowledge, our proof procedure [15] is the only one existing today. On the following small example, weakly terminating, and which is a classical definition of the booleans, enriched with a rule expressing the double application of *not* on *and*, and deliberately oriented in the divergent direction, it fails:

$$\begin{array}{l} and(1,x) \to x \\ and(0,x) \to 0 \\ or(1,x) \to 1 \\ or(0,x) \to x \\ and(1,x) \to not(not(and(1,x))) \\ not(1) \to 0 \\ not(0) \to 1 \\ not(and(x,y)) \to or(not(x), not(y)). \end{array}$$

- as an alternative of the termination proof, as on the following more realistic example of computation of quotient, which is innermost terminating:

$$\begin{array}{l} quot(0, s(y), s(z)) \to 0 \\ quot(s(x), s(y), z) \to quot(x, y, z) \\ quot(x, 0, s(z)) \to s(quot(x, s(z), s(z))) \\ quot(x, y, 0) \to error \\ quot(error, y, z) \to error \\ quot(x, error, z) \to error \\ quot(x, y, error) \to error. \end{array}$$

In addition, if the RS has no rule with a constructor left-hand side, our proof technique establishes weak termination at the same time. This is the case for the two examples above.

In Section 2, the background is presented. In Section 3, we link known notions of completeness to $\mathcal{C}$-reducibility. Section 4 introduces the basic concepts of the inductive proof mechanism. In Section 5, our procedure is defined with inference rules and a strategy to apply them. In Section 6, we explain how to extract a rewriting chain leading to a constructor term, from the proof of $\mathcal{C}$-reducibility of terms.

2. The background

We assume that the reader is familiar with the basic definitions and notations of term rewriting given for instance in [11]. $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is the set of terms built from a given set $\mathcal{F}$ of function symbols having an *arity* $ar = n \in \mathbb{N}$, and a set $\mathcal{X}$ of variables denoted $x, y \ldots$. In this paper, we only consider finite sets of symbols. $\mathcal{T}(\mathcal{F})$ is the set of ground terms (without variables). Symbols of arity 0 are called *constants*. *Positions* in a term are represented as sequences of integers. The empty sequence ϵ denotes the *top* position. The notation $t|_p$ stands for the subterm of t at position p. To emphasize that u contains subterms $t_j, j \in [1..p]$ respectively at positions $\{i_1..i_p\}$, we write $u[t_j]_{\{i_1..i_p\}}$.

A *substitution* is an assignment from $\mathcal{X}$ to $\mathcal{T}(\mathcal{F}, \mathcal{X})$, written $\sigma = (x \mapsto t) \ldots (y \mapsto u)$. It uniquely extends to an endomorphism of $\mathcal{T}(\mathcal{F}, \mathcal{X})$. We identify a substitution $\sigma = (x \mapsto t) \ldots (y \mapsto u)$ with the finite conjunction of equations $(x = t) \wedge \ldots \wedge (y = u)$. The result of applying σ to a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is written $\sigma(t)$ or σt. The *domain* of σ, denoted $Dom(\sigma)$ is the finite subset of $\mathcal{X}$ such that $\sigma x \neq x$. The *range* of σ, denoted $Ran(\sigma)$ is the set $\bigcup_{x \in Dom(\sigma)} Var(\sigma x)$.

A *ground substitution* or *instantiation* is an assignment from $\mathcal{X}$ to $\mathcal{T}(\mathcal{F})$. The *composition* of substitutions σ_1 followed by σ_2 is denoted $\sigma_2\sigma_1$. Given two substitutions σ_1

and σ_2, we write $\sigma_1 \leq \sigma_2$ iff $\exists\theta$ such that $\sigma_2 = \theta\sigma_1$. We say that σ_1 is a *generalization* of σ_2.

For $\mathcal{F}$, we declare a set $\mathcal{C}$ of *constructors* and a set $\mathcal{D}$ of *defined symbols*. The terms of $\mathcal{T}(\mathcal{C})$ are called *constructor terms*, or more briefly $\mathcal{C}$*-terms* (or $\mathcal{C}$*-forms*). Given a set $\mathcal{R}$ of rewrite rules or term rewriting system on $\mathcal{T}(\mathcal{F},\mathcal{X})$, the rewriting relation induced by $\mathcal{R}$ is denoted by $\rightarrow_{\mathcal{R}}$ ($\rightarrow$ if there is no ambiguity on $\mathcal{R}$). We denote $s \rightarrow^{p,l\rightarrow r,\sigma} t$ (where either p or $l \rightarrow r$ or σ may be omitted) if s rewrites into t at position p with the rule $l \rightarrow r$ and the substitution σ, i.e. $s = s[\sigma l]_p$ and $t = s[\sigma r]_p$. The reflexive transitive closure of the rewriting relation induced by $\mathcal{R}$ is denoted by $\rightarrow^*_{\mathcal{R}}$. Given a term t, we call *normal form* of t any irreducible term u, if it exists, such that $t \rightarrow^*_{\mathcal{R}} u$. If a term t rewrites to a $\mathcal{C}$-form, we write this $\mathcal{C}$-form $t{\downarrow_{\mathcal{C}}}$, and say that it is a $\mathcal{C}$-reduced form for $\mathcal{R}$.

An ordering $\succ$ on $\mathcal{T}(\mathcal{F},\mathcal{X})$ is said to be *noetherian* iff there is no infinite decreasing chain for this ordering. It is *monotone* iff for every pair of terms t, t' of $\mathcal{T}(\mathcal{F},\mathcal{X})$, for every context $f(\ldots\ \ldots)$, $t \succ t'$ implies $f(\ldots t \ldots) \succ f(\ldots t' \ldots)$. It has the subterm property iff for every t of $\mathcal{T}(\mathcal{F},\mathcal{X})$, $f(\ldots t \ldots) \succ t$. It has the constructor subterm property iff for every $g(t_1, \ldots, t_n)$ of $\mathcal{T}(\mathcal{F},\mathcal{X})$ with $g \in \mathcal{C}$, we have $g(t_1, \ldots, t_n) \succ t_1, \ldots, t_n$. Note that if $\succ$ is monotone and has the subterm property, as $\mathcal{F}$ is finite, then it is noetherian [10]. If, in addition, $\succ$ is stable by substitution (for every substitution σ, every pair of terms $t, t' \in \mathcal{T}(\mathcal{F},\mathcal{X})$, $t \succ t'$ implies $\sigma t \succ \sigma t'$), then it is called a *simplification ordering*. A *precedence* is an ordering on $\mathcal{F}$, denoted $>_{\mathcal{F}}$.

A RS $\mathcal{R}$ (innermost) *terminates* iff every (innermost) rewriting derivation (or derivation chain) for $\mathcal{R}$ is finite. It *weakly terminates* if at least one derivation chain starting from every term is finite. It is *confluent* iff for every term t such that $t \rightarrow^* u$ and $t \rightarrow^* v$, there exists t' such that $u \rightarrow^* t'$ and $v \rightarrow^* t'$. It is *constructor preserving* iff for each rule $l \rightarrow r$ of $\mathcal{R}$, whenever $l \in \mathcal{T}(\mathcal{C})$, then $r \in \mathcal{T}(\mathcal{C})$.

3. C-reducibility and related properties

Before we compare them, let us now formally define the properties presented in the introduction.

Definition 1. *Let $\mathcal{R} = \{l_i \rightarrow r_i, i \in [1..m]\}$ be a RS on $\mathcal{T}(\mathcal{F},\mathcal{X})$. The* equational theory associated *to $\mathcal{R}$ is the equational theory induced by the set of equations $\mathcal{E} = \{l_i = r_i, i \in [1..m]\}$. It is also noted $\mathcal{E}$.*

Definition 2. *Let $\mathcal{E}$ be an equational theory on $\mathcal{T}(\mathcal{F},\mathcal{X})$, where $\mathcal{F} = \mathcal{C} \cup \mathcal{D}$. $\mathcal{E}$ is* sufficiently complete *(with respect to $\mathcal{D}$) if for every term $t \in \mathcal{T}(\mathcal{F})$, there is a term $u \in \mathcal{T}(\mathcal{C})$ such that $t =_{\mathcal{E}} u$.*

Definition 3. *Let $\mathcal{R}$ be a RS on $\mathcal{T}(\mathcal{F},\mathcal{X})$. A term $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$ is* ground reducible *if every ground instance of t is reducible with $\mathcal{R}$.*

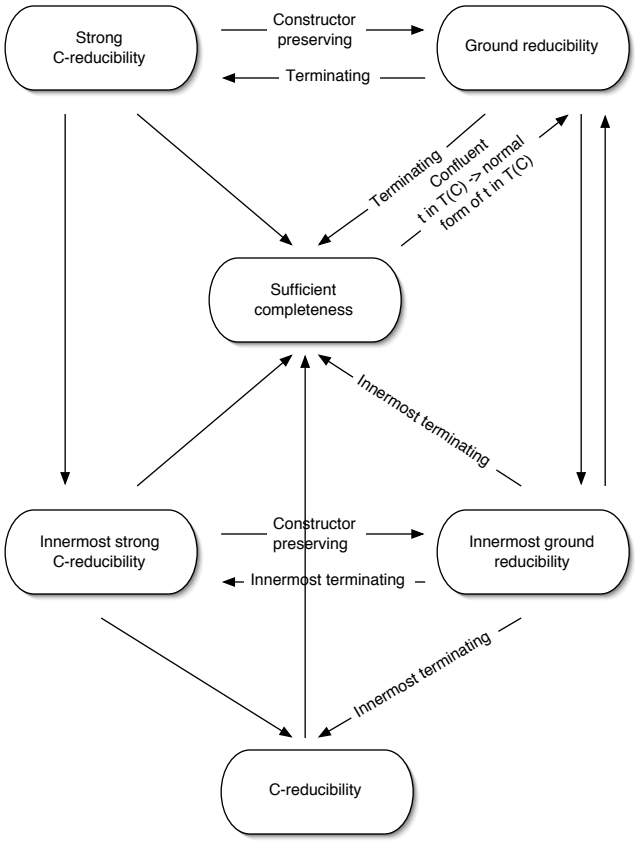
Definition 4. *Let $\mathcal{R}$ be a RS on $\mathcal{T}(\mathcal{F},\mathcal{X})$, where $\mathcal{F} = \mathcal{C} \cup \mathcal{D}$. A ground term is (innermost) (strongly) $\mathcal{C}$-*reducible *iff it reduces into a $\mathcal{C}$-term on at least one of its (resp. all) (innermost) rewriting derivations. The RS $\mathcal{R}$ is said to be (innermost) (strongly) $\mathcal{C}$-*reducing *iff every term t of $\mathcal{T}(\mathcal{F})$ is (innermost) (strongly) $\mathcal{C}$-reducible.*

Let us make some remarks on this property. If $\mathcal{R}$ does not have rules whose left-hand sides are $\mathcal{C}$-terms, the reached $\mathcal{C}$-forms are normal forms.

If there are such rules in $\mathcal{R}$, which is a case of relations between constructors, the $\mathcal{C}$-forms may be reducible. Then, either $\mathcal{R}$ is constructor-preserving and the next terms in the rewriting derivations are still $\mathcal{C}$-terms. Or $\mathcal{R}$ is not constructor-preserving and terms with defined symbols can be introduced after the $\mathcal{C}$-terms in the derivations.

In non terminating cases, $\mathcal{C}$-reducibility allows us to introduce a weak pseudo-termination notion, expressing that the evaluation of every data stops on the first constructor encountered form, ensured to exist on at least one rewriting branch.

Let us now observe the following diagram, summarizing the links between the different properties we consider. "Ground reducibility" stands for ground reducibility of all patterns $f(x_1, \ldots, x_m), f \in \mathcal{D}$; "$\mathcal{C}$-reducibility" stands for $\mathcal{C}$-reducibility of all terms of $\mathcal{T}(\mathcal{F})$. Arrows represent implications between properties; they are labeled by the necessary conditions on the RS required by the implications. All links are quite obvious, except those between ground reducibility and sufficient completeness, given by Theorem 7 in [22].



From the diagram, we immediately set the following theorem, which weakens the terminating condition of the "if" part of Theorem 7 in [22].

Theorem 1. *Let $\mathcal{R}$ be an innermost terminating RS on $\mathcal{T}(\mathcal{F},\mathcal{X})$, where $\mathcal{F} = \mathcal{C} \cup \mathcal{D}$. If every term $f(x_1, \ldots, x_m), f \in$*

$\mathcal{D}, x_1, \ldots, x_m \in \mathcal{X}$ is ground reducible, then the equational theory $\mathcal{E}$ associated to $\mathcal{R}$ is sufficiently complete.

This result can be verified in analyzing the proof of Theorem 7 in [22], where the termination hypothesis can be indeed weakened into innermost termination, and even into weak termination.

Note that sufficient completeness does not imply $\mathcal{C}$-reducibility in every case: for a given term, a $\mathcal{C}$-form reachable by $\mathcal{E}$ can be unreachable by the rewriting relation, as illustrates the following terminating but not confluent example, where a, b, c are constants and c is the only constructor form:

$$a \to b$$
$$a \to c.$$

4. Inductively proving C-reducibility

For proving that a term t of $\mathcal{T}(\mathcal{F})$ is $\mathcal{C}$-reducible, we proceed by induction on $\mathcal{T}(\mathcal{F})$ with a noetherian ordering $\succ$, assuming the property for any t' such that $t \succ t'$. To warrant non emptiness of $\mathcal{T}(\mathcal{F})$ and $\mathcal{T}(\mathcal{C})$, we assume that $\mathcal{C}$ contains at least a constant. The main intuition is to observe rewriting derivations starting from a ground term $t \in \mathcal{T}(\mathcal{F})$ which is any instance of a term $g(x_1, \ldots, x_m) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, for some defined function symbol $g \in \mathcal{D}$, and variables $x_1, \ldots, x_m$. Proving the $\mathcal{C}$-reducibility property on ground terms amounts to prove that at least one derivation leads to a constructor term.

Rewriting derivations are simulated, using a lifting mechanism, by a proof tree developed from $g(x_1, \ldots, x_m)$ on $\mathcal{T}(\mathcal{F}, \mathcal{X})$, for every $g \in \mathcal{D}$, by alternatively using two main concepts, namely narrowing and abstraction. More precisely, narrowing schematizes the rewriting possibilities of terms. Abstraction simulates the reduction of subterms in the derivations until these subterms become $\mathcal{C}$-terms. It expresses the application of the induction hypothesis on these subterms.

The schematization of ground rewriting derivations is achieved through constraints. The nodes of the developed proof trees are composed of a current term of $\mathcal{T}(\mathcal{F}, \mathcal{X})$, and a set of ground substitutions represented by a constraint progressively built along the successive abstraction and narrowing steps. Each node in an abstract tree schematizes a set of ground terms: all ground instances of the current term, that are solutions of the constraint.

The constraint is in fact composed of two kinds of formulas: ordering constraints, set to warrant the validity of the inductive steps, and abstraction constraints combined to narrowing substitutions, which effectively define the relevant sets of ground terms.

As said previously, we consider any term of $\mathcal{T}(\mathcal{F})$ as a ground instance of a term t of $\mathcal{T}(\mathcal{F}, \mathcal{X})$ occurring in a proof tree issued from a reference term t_{ref}.

- first, some subterms $t|_j$ of the current term t of the proof tree are supposed to reduce into a $\mathcal{C}$-term, by the induction hypothesis, if $\theta t_{ref} \succ \theta t|_j$ for the induction ordering $\succ$ and for every θ solution of the constraint associated to t. They are replaced in t by *abstraction variables* X_j representing respectively any of their constructor forms. Reasoning by induction allows us to only suppose the existence of the constructor forms *without explicitly computing them*. If the resulting term only contains constructor symbols and abstraction variables, the initial term is $\mathcal{C}$-reducible. Else,
- the resulting term $u = t[X_j]_{\{i_1,\ldots,i_p\}}$ (where $i_1, \ldots, i_p$ are the abstraction positions in t) is narrowed in all possible ways into terms v, according to the possible instances of the X_j. This corresponds to rewriting ground instances of u (characterized by the constraint associated to u) with all possible rewrite rules.

This technique is inspired from the one we have proposed for proving innermost termination. But in that case, as abstracted subterms were representing innermost normal forms, alternating abstraction and innermost narrowing steps allowed us to simulate all innermost rewriting derivations. Here, abstracted subterms do not generally represent normal forms, and we use standard narrowing, so the process only simulates some possible rewriting derivations, for the standard rewriting relation (without strategy) among which we try to prove the existence of a $\mathcal{C}$-term.

However, we have to ensure that every ground term represented by u is $\mathcal{C}$-reducible, and so we have to prove the existence of at least one branch leading to a $\mathcal{C}$-term for every ground instance of u.

To exhibit such a branch for every ground instance of u containing at least one defined symbol, the narrowing steps have to cover (in a sense we will define later) all possible ground instances of u.

Then $\mathcal{C}$-reducibility of the ground instances of t is reduced to $\mathcal{C}$-reducibility of the ground instances of the terms v. Now, if $\theta t_{ref} \succ \theta v$ for every ground substitution θ that is a solution of the constraint associated to v, by the induction hypothesis, θv is supposed to be $\mathcal{C}$-reducible. Else, the process is iterated on v, until we get a term t' such that either $\theta t_{ref} \succ \theta t'$, or $\theta t'$ is a $\mathcal{C}$-term.

We now introduce some concepts to formalize and automate this mechanism.

4.1 Ordering constraints and abstraction

The induction ordering $\succ$ is constrained along the proof by imposing constraints between terms that must be comparable, each time the induction hypothesis is used in the abstraction mechanism. As we are working with a lifting mechanism on the proof trees with terms of $\mathcal{T}(\mathcal{F}, \mathcal{X})$, we directly work with an ordering $\succ_{\mathcal{P}}$ on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ such that $t_{ref} \succ_{\mathcal{P}} u$ induces $\theta t_{ref} \succ \theta u$, for every θ solution of the constraint associated to u.

So inequalities of the form $t_{ref} > u_1, \ldots, u_m$ are accumulated, which will be called *ordering constraints*. Any ordering $\succ_{\mathcal{P}}$ on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ satisfying them and which is stable by substitution fulfills the previous requirements on ground terms. The ordering $\succ_{\mathcal{P}}$, defined on $\mathcal{T}(\mathcal{F}, \mathcal{X})$, can then be seen as an extension of the induction ordering $\succ$, defined on $\mathcal{T}(\mathcal{F})$. For convenience, the ordering $\succ_{\mathcal{P}}$ will also be written $\succ$.

It is important to note that, for establishing the inductive termination proof, it is sufficient to decide whether there exists such an ordering.

Definition 5. An ordering constraint *is a pair of terms of $\mathcal{T}(\mathcal{F}, \mathcal{X})$ noted $(t > t')$. It is said to be* satisfiable *if there exists an ordering $\succ$, such that for every instantiation θ whose domain contains $\mathcal{V}ar(t) \cup \mathcal{V}ar(t')$, we have $\theta t \succ \theta t'$. We say that $\succ$ satisfies $(t > t')$.*

A conjunction C of ordering constraints is satisfiable if there exists an ordering satisfying all conjuncts. The empty conjunction, always satisfied, is denoted by $\top$.

Satisfiability of a constraint C of this form is undecidable. But a sufficient condition for an ordering $\succ$ to satisfy C

is that $\succ$ is stable by substitution and $t \succ t'$ for every constraint $t > t'$ of C.

The inductive termination proof on ground terms by application of Theorem 2 requires that the noetherian induction ordering has the constructor subterm property. Simplification orderings fulfill such conditions. So in practice, it is sufficient to find a simplification ordering $\succ$ such that $t \succ t'$ for any constraint $t > t'$ of C.

Solving ordering constraints in finding simplification orderings is a well-known problem in rewriting. The simplest and automatable way to proceed is to test simple existing orderings like the subterm ordering, the Recursive Path Ordering, or the Lexicographic Path Ordering. This is often sufficient for the constraints considered here. If these simple orderings are not powerful enough, automatic solvers like Cime2 [9] can provide adequate polynomial orderings.

Other constraints are introduced by the abstraction mechanism. As said previously, to abstract a term t at positions $i_1, \ldots, i_p$, where the $t|_j, j \in \{i_1, \ldots, i_p\}$ are supposed to have a reduced $\mathcal{C}$-form $t|_j\downarrow_{\mathcal{C}}$, we replace the $t|_j$ by abstraction variables X_j representing respectively one of their possible $\mathcal{C}$-forms. Let us define these special variables more formally.

Definition 6. *Let $\mathcal{N}$ be a set of new variables disjoint from $\mathcal{X}$. Symbols of $\mathcal{N}$ are called* $\mathcal{C}$-variables *(or* constructor variables*). Substitutions and instantiations are extended to $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$ in the following way. Let $X \in \mathcal{N}$; for any substitution σ (resp. instantiation θ) such that $X \in Dom(\sigma)$, $\sigma X \in \mathcal{T}(\mathcal{C}, \mathcal{N})$ (resp. $\theta X \in \mathcal{T}(\mathcal{C})$) and then $\mathcal{V}ar(\sigma X) \subseteq \mathcal{N}$.*

Definition 7 (abstraction). *The term $t[t|_j]_{\{i_1,\ldots,i_p\}}$ is said to be* abstracted *into the term u (called* abstraction *of t) at positions $\{i_1, \ldots, i_p\}$ iff $u = t[X_j]_{\{i_1,\ldots,i_p\}}$, where the $X_j, j \in \{i_1, \ldots, i_p\}$ are fresh distinct $\mathcal{C}$-variables.*

$\mathcal{C}$-reducibility on $\mathcal{T}(\mathcal{F})$ is proved by reasoning on terms with abstraction variables, i.e. on terms of $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$. Ordering constraints are extended to pairs of terms of $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$. When subterms t_i are abstracted by X_i, we state constraints on abstraction variables, called *abstraction constraints*, to express that their instances can only be $\mathcal{C}$-terms, rewritten from the corresponding instances of t_i. Initially, they are of the form $t\downarrow_{\mathcal{C}} = X$ where $t \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$, and $X \in \mathcal{N}$, but we will see later on how they will be combined with the substitutions used for the narrowing process.

4.2 Narrowing

After abstraction of the current term t into the term u, if u contains defined symbols, we test whether the ground instances of u are reducible, according to the possible values of the instances of the X_j. This is achieved by narrowing u in all possible ways. We recall the definition of narrowing.

Definition 8 (narrowing). *A term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$* narrows *into a term $t' \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$ at the non-variable position p, using the rule $l \to r \in \mathcal{R}$ with the substitution σ which is written $t \rightsquigarrow^{p,l\to r,\sigma} t'$ iff*

$$\sigma(l) = \sigma(t|_p) \text{ and } t' = \sigma(t[r]_p)$$

where σ is the most general unifier of t at position p and l.

A few remarks can be made on the choice of variables and on the domain of the substitutions generated during the proof process. It is always assumed that there is no variable in common between the rule and the term, i.e. that $Var(l) \cap Var(t) = \emptyset$. This requirement of disjoint variables is easily fulfilled by an appropriate renaming of variables in the rules when narrowing is performed. Observe that for the most general unifier σ used in the above definition, $Dom(\sigma) \subseteq Var(l) \cup Var(t)$ and we can choose $Ran(\sigma) \cap (Var(l) \cup Var(t)) = \emptyset$, thus introducing in the range of σ only fresh variables. Moreover, narrowing is only performed on terms t of $\mathcal{T}(\mathcal{F}, \mathcal{N})$, since an abstracting step is first applied on the reference terms, of the form $g(x_1, \ldots, x_m)$, replacing $x_1, \ldots, x_m \in \mathcal{X}$ by $X_1, \ldots, X_m \in \mathcal{N}$. Then from Definition 6 we infer that for the most general unifiers σ produced during the proof process, all variables of $Ran(\sigma)$ are $\mathcal{C}$-variables.

Notice also that in our process, we are interested in the narrowing substitution applied to the current term t, but not in its definition on the variables of the left-hand side of the rule. So the narrowing substitutions we consider are restricted to the variables of the narrowed term t.

As said before, we only narrow the term u if all narrowing possibilities represent at least one rewriting step for all possible ground instances of u.

Else, the proof process stops with failure. But we cannot conclude that t is not $\mathcal{C}$-reducible, since, although all ground instances of u are not reducible at that step, they may be reduced to a $\mathcal{C}$-form by other rewriting derivations, not observed by the process, as shows the following example. Let $\mathcal{F} = \{f, a, b, c\}$, $\mathcal{C} = \{b, c\}$, $\mathcal{R} = \{a \to b, f(f(b)) \to c, b \to f(b), f(c) \to c\}$. Abstracting the pattern $f(x)$ gives $f(X)$, where X only represents $\mathcal{C}$-forms. Then, $f(X)$ is only narrowed into c for $X = c$. The narrowing substitution $X = f(b)$ is not allowed since $f(b) \notin \mathcal{T}(\mathcal{C}, \mathcal{N})$. However, the other ground instance of $f(X)$, which is $f(b)$, is also $\mathcal{C}$-reducible, but the rewriting chain $f(b) \to f(f(b)) \to c$ is not represented by the abstract-narrow-based process.

Definition 9. *A set of substitutions Σ, is said to* cover a term $u \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$, *or to be* u-covering *iff for any ground instance θu of u, $\exists \sigma \in \Sigma$ such that $Dom(\sigma) \supseteq Var(u)$, and $\theta u = \mu\sigma u$ for some ground substitution μ.*

The following obvious proposition expresses that Definition 9 provides a sufficient condition for the previous requirement on narrowing.

Proposition 1. *Let $\mathcal{R}$ be a RS, u a term of $\mathcal{T}(\mathcal{F}, \mathcal{N})$, and Σ the set of narrowing substitutions of u for $\mathcal{R}$. If Σ is u-covering, then every ground instance αu of u is such that $\alpha u \to^{p,l\to r,\beta\sigma} t'$, for some ground substitution β, and we have $u \rightsquigarrow^{p,l\to r,\sigma} v$ for some v of $\mathcal{T}(\mathcal{F}, \mathcal{N})$, $\beta\sigma = \alpha$ on any variable set $\mathcal{Y} \supseteq Var(u) \cup Dom(\alpha)$, and $t' = \beta v$.*

As a first narrowing step is applied on the patterns $g(X_1, \ldots, X_m), g \in \mathcal{D}$, issued from abstraction of the reference patterns $g(x_1, \ldots, x_m)$, the development of each proof tree at least requires for the narrowing substitutions of $g(X_1, \ldots, X_m)$ to cover $g(X_1, \ldots, X_m)$.

The following two propositions allow us to show that the covering property of the patterns $g(X_1, \ldots, X_m), g \in \mathcal{D}$ is stronger than usual ground reducibility (i.e. of the patterns $g(x_1, \ldots, x_m)$). The first one is an obvious consequence of Proposition 1.

Proposition 2. *Let $\mathcal{R}$ be a RS, $g \in \mathcal{D}$, $X_1, \ldots, X_m \in \mathcal{N}$. Let Σ be the set of narrowing substitutions of $g(X_1, \ldots, X_m)$ with $\mathcal{R}$. If Σ covers $g(X_1, \ldots, X_m)$, then $g(X_1, \ldots, X_m)$ is ground reducible.*

The converse is not true, as shows the following example. Let $\mathcal{F} = \{f, 0, 1\}$, $\mathcal{C} = \{0, 1\}$, and $\mathcal{R} = \{f(0) \to 0, 1 \to 0\}$. The term $f(X)$ is ground reducible since $f(0)$ and $f(1)$ are reducible. However, the set Σ of narrowing substitutions of $f(X)$, equal to $\{\sigma = (X = 0)\}$ is not covering for $f(X)$: $f(1)$ is not a ground instance of $f(0)$.

PROPOSITION 3. *Let $\mathcal{R}$ be a RS on $\mathcal{T}(\mathcal{F}, \mathcal{X})$, $X_1, \ldots, X_m \in \mathcal{N}$, $x_1, \ldots, x_m \in \mathcal{X}$. Then $f(X_1, \ldots, X_m)$ is ground reducible for every $f \in \mathcal{D}$, iff $f(x_1, \ldots, x_m)$ is ground reducible for every $f \in \mathcal{D}$.*

Given a term u, a sufficient condition for a substitution set Σ to be u-covering can be established as follows, provided the variables of the considered term u are $\mathcal{C}$-variables, whose ground instantiations can only be constructor terms. Let P be the set of constructor patterns $\{f(Y_1, \ldots, Y_m) | f \in \mathcal{C}, Y_1, \ldots, Y_m \in \mathcal{N}$, with $ar(f) = m\}$. For $u \in \mathcal{T}(\mathcal{F}, \mathcal{N})$, let Σ_P^u be the set of all possible pattern substitutions of u i.e. the set $\{\sigma_P^u = (X_1 = t_1, \ldots, X_p = t_p) | \{X_1, \ldots, X_p\} = Var(u), t_1, \ldots, t_p \in P\}$.

Then Σ is u-covering if for every $\sigma_P^u \in \Sigma_P^u$, there exists $\sigma \in \Sigma$ such that $\sigma_P^u = \nu\sigma$ for some substitution ν ; in other words, if for every $\sigma_P^u \in \Sigma_P^u$, there exists in Σ a generalization of σ_P^u. This sufficient condition is automatable.

4.3 Cumulating constraints

Abstraction constraints have to be combined with the narrowing substitutions to characterize the ground terms schematized by the current term t in the proof tree. Indeed, a narrowing branch on the current term u with narrowing substitution σ represents a rewriting branch for any ground instance of σu.

In addition, σ has to satisfy the constraints on variables of u, already set in A. So σ, considered as the narrowing constraint attached to the narrowing branch, is added to A. This leads to the introduction of abstraction constraint formulas.

DEFINITION 10. *An* abstraction constrained formula *(ACF in short) is a formula $\bigwedge_i (t_i\downarrow_{\mathcal{C}} = t'_i) \wedge \bigwedge_j (x_j = u_j)$, where $x_j \in \mathcal{X} \cup \mathcal{N}$, $t_i, t'_i, u_j, \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$.*

DEFINITION 11. *An abstraction constrained formula $A = \bigwedge_i (t_i\downarrow_{\mathcal{C}} = t'_i) \wedge \bigwedge_j (x_j = u_j)$ is* satisfiable *iff there exists at least one instantiation θ such that $\bigwedge_i (\theta t_i\downarrow_{\mathcal{C}} = \theta t'_i) \wedge \bigwedge_j (\theta x_j = \theta u_j)$. The instantiation θ is then said to satisfy the ACF A and is called solution of A.*

For a better readability on examples, we can propagate σ into A (by applying σ to A), thus getting instantiated abstraction constraints of the form $t_i\downarrow_{\mathcal{C}} = t'_i$ from initial abstraction constraints of the form $t_i\downarrow_{\mathcal{C}} = X_i$.

An ACF A is attached to each term u in the proof trees; the ground substitutions solutions of A define the instances of the current term u in the proof tree, for which we are observing $\mathcal{C}$-reducibility. When A has no solution, the current node of the proof tree represents no ground term. Such nodes are then irrelevant for the proof. Detecting and suppressing them during a narrowing step allows us to control the narrowing mechanism. So we have the choice between generating only the relevant nodes of the proof tree, by testing satisfiability of A at each step, or stopping the proof on a branch on an irrelevant node, by testing unsatisfiability of A.

Checking satisfiability of A is in general undecidable, but it is often easy in practice to exhibit an instantiation satisfying it. Automatable sufficient conditions are also under study.

Unsatisfiability of A is also undecidable in general, but automatable simple sufficient conditions can be used, as testing whether A contains equalities $t\downarrow_{\mathcal{C}} = u$, where u contains at least a symbol of $\mathcal{D}$.

In the following, we present the procedure dealing with satisfiability of A.

5. Inference rules for the inductive proof

We are now ready to describe the inference rules defining our mechanism. They transform a set T of 3-tuples (U, A, C) where $U = \{t\}$ or $\emptyset$, t is the current term whose $\mathcal{C}$-reducibility has to be proved, A is an abstraction constraint formula, C is a conjunction of ordering constraints.

- The first rule abstracts the current term t at given positions $i_1, \ldots, i_p$. The constraints $t_{ref} > t|_{i_1}, \ldots, t|_{i_p}$ are set, allowing us to suppose, by induction, the existence of $\mathcal{C}$-forms for $t|_{i_1}, \ldots, t|_{i_p}$. Then, $t|_{i_1}, \ldots, t|_{i_p}$ are abstracted into abstraction variables $X_{i_1}, \ldots, X_{i_p}$. The abstraction constraint $t|_{i_1}\downarrow_{\mathcal{C}} = X_{i_1}, \ldots, t|_{i_p}\downarrow_{\mathcal{C}} = X_{i_p}$ is added to the ACF A. We call that rule **Abstract**.

  The abstraction positions are chosen so that the abstraction mechanism captures the greatest possible number of rewriting steps: then we abstract all greatest possible subterms of $t = f(t_1, \ldots, t_m)$. More concretely, we try to abstract $t_1, \ldots, t_m$ and, for each $t_i = g(t'_1, \ldots, t'_n)$ that cannot be abstracted, we try to abstract $t'_1, \ldots, t'_n$, and so on. In the worst case, we are driven to abstract leaves of the term, which are either variables, or constants.

  Note also that it is not useful to abstract subterms if they are in $\mathcal{T}(\mathcal{C}, \mathcal{N})$. Indeed, by Definition 6, every ground instance of such subterms is already a $\mathcal{C}$-term.

- The second rule narrows the resulting term u, if it is not a term of $\mathcal{T}(\mathcal{C}, \mathcal{N})$, in all possible ways in one step, with all possible rewrite rules of the rewrite system $\mathcal{R}$, and all possible substitutions $\sigma_1, \ldots, \sigma_n$, into terms $v_1, \ldots, v_k$, according to Definition 8, if $\{\sigma_1, \ldots, \sigma_n\}$ is u-covering. This step is a branching step, creating as many states as narrowing possibilities. The substitution σ is integrated to A. This is the **Narrow** rule. As $\{\sigma_1, \ldots, \sigma_n\}$ is u-covering, the narrowing step effectively simulates at least one rewriting step for each possible ground instance of u.

- We finally have a **Stop** rule halting the proof process on the current branch of the proof tree, when the ground instances of the current term can be stated as C-reducible. This happens when the whole current term u can be abstracted, i.e. when the induction hypothesis applies on it, or when $u \in \mathcal{T}(\mathcal{C}, \mathcal{N})$.

As said before, ordering constraints have to be satisfied by a noetherian ordering having the constructor subterm property. Any simplification ordering holds. We can make further assumptions on the ordering to enable constraints to be satisfied. In particular, for a Recursive Path Ordering or Lexicographic Path Ordering $\succ$ whose precedence $>_{\mathcal{F}}$ is such that $f >_{\mathcal{F}} c$, $\forall f \in \mathcal{D}$, $\forall c \in \mathcal{C}$, we have $t \succ u$ for $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ containing at least a symbol of $\mathcal{D}$ and $u \in \mathcal{T}(\mathcal{C})$.

Exploiting equalities in A can also help to solve ordering constraints: $t_i\downarrow_{\mathcal{C}} = X_i$ means that θX_i is a $\mathcal{C}$-form obtained by rewriting θt_i, for every ground substitution θ. As we do not abstract terms of $\mathcal{T}(\mathcal{C}, \mathcal{N})$, if $t_i \neq x_i \in \mathcal{X}$, θt_i contains at least a symbol of $\mathcal{D}$, and then $\theta t_i \succ \theta X_i$ for a RPO or an

LPO defined as above. As t_{ref} also contains a symbol of $\mathcal{D}$, we also have $\theta t_{ref} \succ \theta X$ for $X \in \mathcal{N}$.

Before giving the inference rules, let us note that the inductive reasoning can be completed in the following way. When the induction hypothesis cannot be applied to a term u, it may be possible to prove $\mathcal{C}$-reducibility of every ground instance of u by another way. Let $C{-}RED(u)$ be a predicate that is true iff every ground instance of u is $\mathcal{C}$-reducible. In the first and third previous steps of the inductive reasoning, we then associate the alternative predicate $C{-}RED(u)$ to the condition $t > u$. It is true in particular if $u \in \mathcal{T}(\mathcal{C}, \mathcal{N})$, as said in the presentation of the **Stop** rule.

The rules are given in Table 1. They use a reference term $t_{ref} = g(x_1, \ldots, x_m)$, where $x_1, \ldots, x_m \in \mathcal{X}$ and $g \in \mathcal{D}$ (if g is a constant, then $t_{ref} = g$). To prove $\mathcal{C}$-reducibility for every term $t \in \mathcal{T}(\mathcal{F})$ we proceed as follows: for each defined symbol $g \in \mathcal{D}$, we apply the rules using the reference term $t_{ref} = g(x_1, \ldots, x_m)$ on the initial set of 3-tuples $\{(\{t_{ref} = g(x_1, \ldots, x_m)\}, \top, \top)\}$, with a specific strategy S.

The strategy S repeats the following steps: first, apply **Abstract**, and then try **Stop**. Then try all possible applications of **Narrow**. Then, try **Stop** again.

Note that if A is satisfiable, the transformed forms of A by **Abstract** and **Stop** are also satisfiable. Moreover, the first application of **Abstract** generates $A = (\bigwedge_i x_i{\downarrow_{\mathcal{C}}} = X_i)$, always satisfied by the constructor constant supposed to exist in $\mathcal{C}$. Thus with the strategy S, it is useless to prove satisfiability of A in the rules **Abstract** and **Stop**.

The process may not terminate if there is an infinite number of applications of **Abstract** and **Narrow** on the same branch of a derivation tree. Nothing can be said in that case. It stops if no inference rule applies anymore.

A branch of the derivation tree is said to be successful if it is ended by an application of **Stop**, i.e. if its final state is of the form $(\emptyset, A, C)$.

Thus, the inductive $\mathcal{C}$-reducibility proof is successful if there is at least one successful branch in the proof tree, corresponding to each possible ground term. Let us develop this point.

In fact, branching, produced by **Narrow**, can generate different states with narrowing substitutions $\sigma_1, \ldots \sigma_n$. These substitutions can be compared. For σ_i and σ_j, three situations may occur: σ_i is strictly less general than σ_j, which is noted $\sigma_i > \sigma_j$, (or σ_j is strictly less general than σ_i), σ_i and σ_j are equal up to a renaming, or else σ_i and σ_j are incomparable.

States corresponding to substitutions that are more general than other ones then represent a set of ground instances that contains the other ones. So, for proving $\mathcal{C}$-reducibility for all ground instances at a branching point, it is sufficient to prove $\mathcal{C}$-reducibility only for the "most general states".

A branching node in a proof tree can only be a state, on which the Narrow rule applies. Let Σ be the set of narrowing substitutions (possibly with different rewrite rules) at a given branching node. Let Σ_0 be the reduced set from Σ such that $\sigma \in \Sigma_0$ iff $\sigma \in \Sigma$ and $\nexists\, \sigma' \in \Sigma$ such that $\sigma > \sigma'$ on $(Dom(\sigma) \setminus Var(l)) \cup (Dom(\sigma') \setminus Var(l'))$, where l and l' are the left-hand sides of rules respectively used to produce the narrowing substitutions σ and σ'. The set Σ_0 may yet contain equivalent (equal up to a renaming) substitutions which are marked as such. So for any two substitutions in Σ_0, either they are equivalent, or they are incomparable.

A proof tree is *weakly successful* if at each branching node:

- for each class of equivalent substitutions, there exists at least one weakly successful subtree corresponding to a substitution in this class,
- all subtrees corresponding to incomparable substitutions are weakly successful,
- a tree reduced to the state $(\emptyset, A, C)$ is weakly successful.

So the strategy S can be optimized as follows: at each branching point of a proof tree, with set of substitutions Σ, we only develop the subtrees corresponding to Σ_0. Moreover, given two subtrees corresponding to equivalent substitutions, as soon as one of them is weakly successful, the other one is cut.

We write $SUCCESS(g, \succ)$ if the proof tree obtained by application on $(\{g(x_1, \ldots, x_m)\}, \top, \top)$, with the strategy S, of the inference rules whose conditions are satisfied by an ordering $\succ$, is weakly successful.

THEOREM 2. *Let $\mathcal{R}$ be a RS on a set $\mathcal{F}$ of symbols. If there exists a noetherian ordering $\succ$ having the constructor subterm property, such that for each defined symbol g, we have $SUCCESS(g, \succ)$, then $\mathcal{R}$ is $\mathcal{C}$-reducing.*

Recall the important point that the ordering $\succ$ has to be the same for all $g(x_1, \ldots, x_m) \in \mathcal{D}$.

The subtree cut process is similar to the one defined for weak termination, for which a formal description with a complete set of inference rules is given in [16].

Let us come back to the first example of the introduction.

Example: We prove that the RS

$$
\begin{array}{l}
and(1, x) \to x \\
and(0, x) \to 0 \\
or(1, x) \to 1 \\
or(0, x) \to x \\
and(1, x) \to not(not(and(1, x))) \\
not(1) \to 0 \\
not(0) \to 1 \\
not(and(x, y)) \to or(not(x), not(y))
\end{array}
$$

on $\mathcal{T}(\mathcal{F})$, with $\mathcal{F} = \{and : 2, or : 2, not : 1, 1 : 0, 0 : 0\}$, and $\mathcal{C} = \{0, 1\}$, is $\mathcal{C}$-reducing.

Applying the rules on $not(x)$, we get:

$not(x_1)$	$A = \top, \quad C = \top$
Abstract	
$not(X_1)$	$A = (x_1{\downarrow_{\mathcal{C}}} = X_1)$
	$C = (not(x_1) > x_1)$
Narrow	
1	$\sigma_1 = (X_1 = 0)$
	$A = (x_1{\downarrow_{\mathcal{C}}} = 0)$
	$C = (not(x_1) > x_1)$
0	$\sigma_2 = (X_1 = 1)$
	$A = (x_1{\downarrow_{\mathcal{C}}} = 1)$
	$C = (not(x_1) > x_1)$
Stop(*twice*)	
$\emptyset$	$A = (x_1{\downarrow_{\mathcal{C}}} = 0)$
	$C = (not(x_1) > x_1)$
$\emptyset$	$A = (x_1{\downarrow_{\mathcal{C}}} = 1)$
	$C = (not(x_1) > x_1)$

Narrow would also apply with $\sigma_3 = (X_1 = and(X_2, X_3))$, but σ_3 is not licit since $and(X_2, X_3) \notin \mathcal{T}(\mathcal{C}, \mathcal{N})$.

Table 1. Inference rules for $\mathcal{C}$-reducibility

Abstract: $$\frac{\{t\},\ A,\ C}{\{u\},\ A \wedge \bigwedge_{j\in\{i_1,\ldots,i_p\}} t|_j\downarrow_{\mathcal{C}} = X_j,\ C \wedge \bigwedge_{j\in\{i_1,\ldots,i_p\}} H_C(t|_j)}$$

where t is abstracted into u at positions $i_1,\ldots,i_p \neq \epsilon$
if $C \wedge H_C(t|_{i_1}) \ldots \wedge H_C(|t_{i_p})$ is satisfiable

Narrow: $$\frac{\{t\},\ A,\ C}{\{u\},\ A\wedge\sigma,\ C}$$

if $t \rightsquigarrow^{\sigma}_{\mathcal{R}} u$ and $A \wedge \sigma$ is satisfiable

Stop: $$\frac{\{t\},\ A,\ C}{\emptyset,\ A \wedge H_A(t),\ C \wedge H_C(t)}$$

if $(C \wedge H_C(t))$ is satisfiable.

and $H_A(t) = \begin{cases} \top & t \text{ is in } \mathcal{T}(\mathcal{C},\mathcal{N}) \\ t\downarrow_{\mathcal{C}} = X & \text{otherwise.} \end{cases}$

$H_C(t) = \begin{cases} \top & \text{if } C{-}RED(t) \\ t_{ref} > t & \text{otherwise.} \end{cases}$

Moreover $\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$ is $not(X_1)$-covering since for every possible ground instance θX_1 of X_1 (0 or 1), there exists σ in Σ such that $\theta\ not(X_1) = \mu\sigma\ not(X_1)$ for some μ, which is here equal to Id.

Stop applies (twice) since 0 and 1 are terms of $\mathcal{T}(\mathcal{C},\mathcal{N})$.

The ordering constraints are satisfied by any term ordering having the subterm property, for example by a RPO with any precedence.

Now, considering $or(x_1, x_2)$, we get:

$or(x_1,x_2)$	$A = \top \quad C = \top$
Abstract	
$or(X_1,X_2)$	$A = (x_1\downarrow_{\mathcal{C}} = X_1 \wedge x_2\downarrow_{\mathcal{C}} = X_2)$
	$C = (or(x_1,x_2) > x_1, x_2)$
Narrow	
X_2	$\sigma_1 = (X_1 = 0)$
	$A = (x_1\downarrow_{\mathcal{C}} = 0 \wedge x_2\downarrow_{\mathcal{C}} = X_2)$
	$C = (or(X_1,X_2) > x_1, x_2)$
1	$\sigma_2 = (X_1 = 1)$
	$A = (x_1\downarrow_{\mathcal{C}} = 1 \wedge x_2\downarrow_{\mathcal{C}} = X_2)$
	$C = (or(X_1,X_2) > x_1, x_2)$
Stop(*twice*)	
$\emptyset$	$A = (x_1\downarrow_{\mathcal{C}} = 0 \wedge x_2\downarrow_{\mathcal{C}} = X_2)$
	$C = (or(X_1,X_2) > x_1, x_2)$
$\emptyset$	$A = (x_1\downarrow_{\mathcal{C}} = 1 \wedge x_2\downarrow_{\mathcal{C}} = X_2)$
	$C = (or(X_1,X_2) > x_1, x_2)$

$\Sigma = \{\sigma_1, \sigma_2\}$ is $or(X_1, X_2)$-covering.
Stop applies since X_2 and 1 are terms of $\mathcal{T}(\mathcal{C},\mathcal{N})$.

Finally, considering $and(x_1, x_2)$, we get:

$and(x_1,x_2)$	$A = \top \quad C = \top$
Abstract	
$and(X_1,X_2)$	$A = (x_1\downarrow_{\mathcal{C}} = X_1 \wedge x_2\downarrow_{\mathcal{C}} = X_2)$
	$C = (and(x_1,x_2) > x_1, x_2)$
Narrow	
0	$\sigma_1 = (X_1 = 0)$
	$A = (x_1\downarrow_{\mathcal{C}} = 0 \wedge x_2\downarrow_{\mathcal{C}} = X_2)$
	$C = (and(X_1,X_2) > x_1, x_2)$
X_2	$\sigma_2 = (X_1 = 1)$
	$A = (x_1\downarrow_{\mathcal{C}} = 1 \wedge x_2\downarrow_{\mathcal{C}} = X_2)$
	$C = (and(X_1,X_2) > x_1, x_2)$
$not(not(and(1,X_2)))$	$\sigma_3 = (X_1 = 1)$
	$A = (x_1\downarrow_{\mathcal{C}} = 1 \wedge x_2\downarrow_{\mathcal{C}} = X_2)$
	$C = (and(X_1,X_2) > x_1, x_2)$
Stop(*three times*)	
$\emptyset$	$A = (x_1\downarrow_{\mathcal{C}} = 0 \wedge x_2\downarrow_{\mathcal{C}} = X_2)$
	$C = (and(X_1,X_2) > x_1, x_2))$
$\emptyset$	$A = (x_1\downarrow_{\mathcal{C}} = 1 \wedge x_2\downarrow_{\mathcal{C}} = X_2)$
	$C = (and(X_1,X_2) > x_1, x_2))$
$\emptyset$	$A = (x_1\downarrow_{\mathcal{C}} = 1 \wedge x_2\downarrow_{\mathcal{C}} = X_2)$
	$C = (and(X_1,X_2) > x_1, x_2)$

$\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$ is $and(X_1, X_2)$-covering.
Stop applies on the two first branches because 0 and X_2 are terms of $\mathcal{T}(\mathcal{C},\mathcal{N})$, and on the third one because it is issued from the same narrowing substitution as the second one, on which **Stop** applies.

The ordering constraints of the last two proof trees are satisfied by the same ordering as previously.

Note that, as $\mathcal{R}$ does not contain rules whose left-hand side is a $\mathcal{C}$-term, $\mathcal{C}$-reducibility ensures weak termination of the RS.

Now, if we add to $\mathcal{R}$ the relations between constructors

$$1 \to true$$
$$true \to 1$$
$$0 \to false$$
$$false \to 0$$

where $true$ and $false$ are also constructor symbols, we loose the weak termination property. But $\mathcal{C}$-reducibility is proved in the same way than previously, since we have no additional proof tree, and the new proof trees are the same as the previous ones.

Let us now consider the second example of the introduction.

Example:

$$quot(0, s(y), s(z)) \to 0$$
$$quot(s(x), s(y), z) \to quot(x, y, z)$$
$$quot(x, 0, s(z)) \to s(quot(x, s(z), s(z)))$$
$$quot(x, y, 0) \to error$$
$$quot(error, y, z) \to error$$
$$quot(x, error, z) \to error$$
$$quot(x, y, error) \to error$$

with $\mathcal{F} = \{quot : 3, s : 1, 0 : 0, error : 0\}$ and $\mathcal{C} = \{s, 0, error\}$.

So, either we prove innermost termination of the RS, (the dependency pair method [18] works), and we infer sufficient completeness, by Proposition 2 and Theorem 1, in proving coveredness of all patterns $f(X_1, \ldots, X_m), f \in \mathcal{D}$, or we directly apply the $\mathcal{C}$-reducibility proof method. Let us develop the second solution.

Applying the rules on $quot(x_1, x_2, x_3)$, we get:

$quot(x_1, x_2, x_3)$	$A = \top, \quad C = \top$
Abstract	
$quot(X_1, X_2, X_3)$	$A = (x_1\downarrow_{\mathcal{C}} = X_1 \wedge x_2\downarrow_{\mathcal{C}} = X_2 \wedge x_3\downarrow_{\mathcal{C}} = X_3)$ $C = (quot(x_1, x_2, x_3) > x_1, x_2, x_3)$
Narrow	
0	$\sigma_1 = (X_1 = 0 \wedge X_2 = s(X_2') \wedge X_3 = s(X_3'))$ $A = (x_1\downarrow_{\mathcal{C}} = 0 \wedge x_2\downarrow_{\mathcal{C}} = s(X_2') \wedge x_3\downarrow_{\mathcal{C}} = s(X_3'))$ $C = (quot(x_1, x_2, x_3) > x_1, x_2, x_3)$
$quot(X_1', X_2', X_3)$	$\sigma_2 = (X_1 = s(X_1') \wedge X_2 = s(X_2'))$ $A = (x_1\downarrow_{\mathcal{C}} = s(X_1') \wedge x_2\downarrow_{\mathcal{C}} = s(X_2') \wedge x_3\downarrow_{\mathcal{C}} = X_3)$ $C = (quot(x_1, x_2, x_3) > x_1, x_2, x_3)$
$s(quot(X_1, s(X_3'), s(X_3')))$	$\sigma_3 = (X_2 = 0 \wedge X_3 = s(X_3'))$ $A = (x_1\downarrow_{\mathcal{C}} = X_1 \wedge x_2\downarrow_{\mathcal{C}} = 0 \wedge x_3\downarrow_{\mathcal{C}} = s(X_3'))$ $C = (quot(x_1, x_2, x_3) > x_1, x_2, x_3)$
$error$	$\sigma_4 = (X_3 = 0)$ $A = (x_1\downarrow_{\mathcal{C}} = X_1 \wedge x_2\downarrow_{\mathcal{C}} = X_2 \wedge x_3\downarrow_{\mathcal{C}} = 0)$ $C = (quot(x_1, x_2, x_3) > x_1, x_2, x_3)$
$error$	$\sigma_5 = (X_1 = error)$ $A = (x_1\downarrow_{\mathcal{C}} = error \wedge x_2\downarrow_{\mathcal{C}} = X_2 \wedge x_3\downarrow_{\mathcal{C}} = X_3)$ $C = (quot(x_1, x_2, x_3) > x_1, x_2, x_3)$
$error$	$\sigma_6 = (X_2 = error)$ $A = (x_1\downarrow_{\mathcal{C}} = X_1 \wedge x_2\downarrow_{\mathcal{C}} = error \wedge x_3\downarrow_{\mathcal{C}} = X_3)$ $C = (quot(x_1, x_2, x_3) > x_1, x_2, x_3)$
$error$	$\sigma_7 = (X_3 = error)$ $A = (x_1\downarrow_{\mathcal{C}} = X_1 \wedge x_2\downarrow_{\mathcal{C}} = X_2 \wedge x_3\downarrow_{\mathcal{C}} = error)$ $C = (quot(x_1, x_2, x_3) > x_1, x_2, x_3)$

Applying the rule **Narrow** here gives seven branches, following the seven rules of $\mathcal{R}$. Let $u = quot(X_1, X_2, X_3)$. The set $\Sigma = \{\sigma_i, i \in [1..7]\}$ is u-covering, since every σ_P^u of Σ_P^u, where $P = \{0, error, s(X) | X \in \mathcal{N}\}$, has a generalization in Σ.

Then **Stop** applies on all branches, except the third one, for the following reasons. On the first branch and the last four ones, we get $\mathcal{C}$-terms as current terms.

On the second branch, we have $quot(x_1, x_2, x_3) \succ quot(X_1', X_2', X_3)$ for a Lexicographic Path Ordering with any precedence and a left-to-right status for $quot$. Indeed, $\mathcal{R}$ does not contain any rule whose left-hand side is a $\mathcal{C}$-term. Then, if x_1 represents a $\mathcal{C}$-term, it is in normal form, and then $x_1 = x_1\downarrow_{\mathcal{C}}$. In A, we have $x_1\downarrow_{\mathcal{C}} = s(X_1')$. Then $x_1 \succ X_1'$. If x_1 represents a term containing a defined symbol, then $x_1 \succ X_1'$ (see Section 5). In a similar way, we obtain $x_2 \succ X_2'$.

Now, if x_3 represents a term containing a defined symbol, as previously for x_1, we have $x_3 \succ X_3$. If x_3 represents a $\mathcal{C}$-term, we get $x_3 = x_3\downarrow_{\mathcal{C}} = X_3$, so $x_3 \succeq X_3$.

By definition of the LPO, as we have $x_1 \succ X_1'$, we have to verify that $quot(x_1, x_2, x_3) \succ X_2', X_3$. This is true since $x_2 \succ X_2'$ and $x_3 \succeq X_3$.

Then we get:

Stop (**six times**)	
$\emptyset$	$A = (x_1\downarrow_{\mathcal{C}} = 0 \wedge x_2\downarrow_{\mathcal{C}} = s(X_2') \wedge x_3\downarrow_{\mathcal{C}} = s(X_3'))$ $C = (quot(x_1, x_2, x_3) > x_1, x_2, x_3)$

$\emptyset$	$A = (x_1\downarrow_{\mathcal{C}} = s(X_1') \wedge x_2\downarrow_{\mathcal{C}} = s(X_2') \wedge x_3\downarrow_{\mathcal{C}} = X_3 \wedge quot(X_1', X_2', X_3)\downarrow_{\mathcal{C}} = X_4)$ $C = (quot(x_1, x_2, x_3) > x_1, x_2, x_3, quot(X_1', X_2', X_3))$
$s(quot(X_1, s(X_3'), s(X_3')))$	$A = (x_1\downarrow_{\mathcal{C}} = X_1 \wedge x_2\downarrow_{\mathcal{C}} = 0 \wedge x_3\downarrow_{\mathcal{C}} = s(X_3'))$ $C = (quot(x_1, x_2, x_3) > x_1, x_2, x_3)$
$\emptyset$	$A = (x_1\downarrow_{\mathcal{C}} = X_1 \wedge x_2\downarrow_{\mathcal{C}} = X_2 \wedge x_3\downarrow_{\mathcal{C}} = 0)$ $C = (quot(x_1, x_2, x_3) > x_1, x_2, x_3)$
$\emptyset$	$A = (x_1\downarrow_{\mathcal{C}} = error \wedge x_2\downarrow_{\mathcal{C}} = X_2 \wedge x_3\downarrow_{\mathcal{C}} = X_3)$ $C = (quot(x_1, x_2, x_3) > x_1, x_2, x_3)$
$\emptyset$	$A = (x_1\downarrow_{\mathcal{C}} = X_1 \wedge x_2\downarrow_{\mathcal{C}} = error \wedge x_3\downarrow_{\mathcal{C}} = X_3)$ $C = (quot(x_1, x_2, x_3) > x_1, x_2, x_3)$
$\emptyset$	$A = (x_1\downarrow_{\mathcal{C}} = X_1 \wedge x_2\downarrow_{\mathcal{C}} = X_2 \wedge x_3\downarrow_{\mathcal{C}} = error)$ $C = (quot(x_1, x_2, x_3) > x_1, x_2, x_3)$

Now, on the third branch with the term $s(quot(X_1, s(X_3'), s(X_3')))$, **Narrow** applies:

Narrow	
$s(0)$	$\sigma_1 = (X_1 = 0)$ $A = (x_1\downarrow_{\mathcal{C}} = 0 \wedge x_2\downarrow_{\mathcal{C}} = 0 \wedge x_3\downarrow_{\mathcal{C}} = s(X_3'))$ $C = (quot(x_1, x_2, x_3) > x_1, x_2, x_3)$
$s(error)$	$\sigma_2 = (X_1 = error)$ $A = (x_1\downarrow_{\mathcal{C}} = error \wedge x_2\downarrow_{\mathcal{C}} = 0 \wedge x_3\downarrow_{\mathcal{C}} = s(X_3'))$ $C = (quot(x_1, x_2, x_3) > x_1, x_2, x_3)$
$s(quot(X_1', X_3', s(X_3')))$	$\sigma_3 = (X_1 = s(X_1'))$ $A = (x_1\downarrow_{\mathcal{C}} = s(X_1') \wedge x_2\downarrow_{\mathcal{C}} = 0 \wedge x_3\downarrow_{\mathcal{C}} = s(X_3'))$ $C = (quot(x_1, x_2, x_3) > x_1, x_2, x_3)$

$\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$ is covering for $s(quot(X_1, s(X_3'), s(X_3')))$.

Now **Stop** applies on the first two branches above since $s(0)$ and $s(error)$ are $\mathcal{C}$-terms.

Finally, $quot(x_1, x_2, x_3) \succ s(quot(X_1', X_3', s(X_3'))$ since, as $quot \in \mathcal{D}$ and $s \in \mathcal{C}$, we have $quot >_{\mathcal{F}} s$ for the previous LPO. Indeed, in a similar way than previously, if x_1 is a $\mathcal{C}$-term, $x_1 = x_1\downarrow_{\mathcal{C}} = s(X_1') \succ X_1'$. If x_1 contains a defined symbol, $x_1 \succ x_1\downarrow_{\mathcal{C}} = s(X_1') \succ X_1'$. We also get $x_3 \succeq s(X_3') \succ X_3'$.

By definition of the LPO, as we have $x_1 \succ X_1'$, we have to verify that $quot(x_1, x_2, x_3) \succ X_3', s(X_3')$. This is true since $x_3 \succeq s(X_3') \succ X_3'$.

So **Stop** applies on the third branch, which ends the proof.

Other examples can be found in [19], as well as proofs of propositions and theorems.

6. Finding a good derivation chain

Our proof process, as it simulates the rewriting mechanism, gives complete information on a rewriting branch leading to a $\mathcal{C}$-form. It allows us extracting the exact application of rewrite rules that yields a $\mathcal{C}$-form. To rewrite a term, it is enough to follow the rewriting scheme simulated by abstraction and narrowing in the proof trees.

We now formalize the use of the proof trees to compute a $\mathcal{C}$-form for any term.

DEFINITION 12. *Let $\mathcal{R}$ be a RS proved $\mathcal{C}$-reducing with Theorem 2. The strategy tree ST_f associated to $f \in \mathcal{D}$ is the proof tree obtained from the initial state $(\{f(x_1, \ldots, x_m)\}, \top, \top)$.*

DEFINITION 13. *Let $\mathcal{R}$ be a RS proved $\mathcal{C}$-reducing with Theorem 2. Let $ST = \{ST_f | f \in \mathcal{D}\}$ be the set of strategy trees of $\mathcal{R}$ and $s = f(s_1, \ldots, s_m) \in \mathcal{T}(\mathcal{F})$. Rewriting s with respect to ST into a $\mathcal{C}$-form, $\mathcal{C}form_{ST}(s)$, is defined in the following way, where $\mapsto$ denotes a transformation step:*

- *if $f \in \mathcal{C}$, then $\mathcal{C}form_{ST}(f(s_1, \ldots, s_n)) = f(\mathcal{C}form_{ST}(s_1), \ldots, \mathcal{C}form_{ST}(s_n))$,*
- *if $f \in \mathcal{D}$, then reducing s with respect to ST into $\mathcal{C}form_{ST}(s)$ is performed by following the steps in the strategy tree ST_f of f, where t is any term of the transformation chain of s with respect to ST and u is the corresponding term in ST_f:*
  - *if the step is* **Abstract**, *and abstracts u at positions $i_1, \ldots, i_p$,*
    then $t \mapsto t[t_j']_{j \in \{i_1, \ldots, i_p\}}$,
    where $t_j' = \begin{cases} t|_j\downarrow_{\mathcal{C}} & \text{if } C{-}RED(u|_j) \\ \mathcal{C}form_{ST}(t|_j) & \text{otherwise,} \end{cases}$
  - *if the step is* **Narrow** *with $u \rightsquigarrow^{p, l \to r, \sigma} u'$, then $t \mapsto t'$ where t' is defined by $t \to^{p, l \to r, \beta\sigma} t' = \beta u'$, with $\alpha = \beta\sigma$ on $Var(u) \cup Dom(\alpha)$ and $t = \alpha u$,*
  - *if the step is* **Stop**, *then $t \mapsto t'$,*
    where $t' = \begin{cases} t\downarrow_{\mathcal{C}} & \text{if } C{-}RED(u) \\ \mathcal{C}form_{ST}(t) & \text{otherwise.} \end{cases}$

Given a RS $\mathcal{R}$, the previous definition assumes that if the predicate $C{-}RED$ has been used to prove $\mathcal{C}$-reducibility of a particular term u during the proof that $\mathcal{R}$ is $\mathcal{C}$-reducing, one is able to find a $\mathcal{C}$-form for ground instances of u. It is obviously the case if $u \in \mathcal{T}(\mathcal{C}, \mathcal{N})$. Under this assumption, the following theorem holds.

THEOREM 3. *Let $\mathcal{R}$ be a RS proved $\mathcal{C}$-reducing with Theorem 2 and ST its set of strategy trees. Then for any term $t \in \mathcal{T}(\mathcal{F})$, $\mathcal{C}form_{ST}(t)$ is a $\mathcal{C}$-form of t for $\mathcal{R}$.*

7. Conclusion and perspectives

In this paper, we have studied the problem of $\mathcal{C}$-reducibility of ground terms, i.e., reducibility of terms into a constructor form on at least one of their rewriting derivations. This property is interesting from several points of view. It allows us to warrant a completely computed form for every data of rewrite programs, and directly implies sufficient completeness of specifications. In the light of $\mathcal{C}$-reducibility, termination of a program becomes less important, provided one is able to reach a $\mathcal{C}$-form, i.e, to compute a result, which gives a criterion to stop the computations.

Our proof method establishes $\mathcal{C}$-reducibility, and then sufficient completeness, by explicit induction on the $\mathcal{C}$-reducibility property itself. It simulates rewriting by alternatively using abstraction of subterms, and narrowing, provided all possible ground instances of the narrowed terms are covered. It works on the ground term algebra using as induction relation a noetherian ordering having the constructor subterm property.

It does not require confluence, nor restrictions like absence of relation between constructors or the constructor preserving property. It does not need any termination property either. When there is no constructor left-hand side of rule, it even provides a proof of weak termination. It is constructive in the sense that a computation branch leading to a constructor form can be deduced from the proof trees.

Thanks to the power of induction, the ordering constraints generated by the proof process are often simple and satisfied by the subterm ordering, or by an usual ordering like the Recursive Path Ordering. If not, they can be delegated to automatic ordering constraint solvers. Verifying unsatisfiability of A is also automatable, with the given sufficient conditions cited in Section 4.3. So we can have an automatic implementation of the proof procedure establishing $\mathcal{C}$-reducibility. As the proof technique is close to the rewriting process itself, it seems to be promising to extend it to conditional, typed or constrained rewriting.

References

[1] P. Borovanský, C. Kirchner, H. Kirchner, P.-E. Moreau, and C. Ringeissen. An Overview of ELAN. In C. Kirchner and H. Kirchner, editors, *Proceedings of the 2nd International Workshop on Rewriting Logic and its Applications*, Electronic Notes in Theoretical Computer Science, Pont-à-Mousson (France), Sept. 1998. Elsevier Science Publishers B. V. (North-Holland).

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

[3] A. Bouhoula and F. Jaquemard. Automatic verification of. sufficient completeness for. specifications of complex data structures. Technical Report RR-LSV-05-17, INRIA, 2005.

[4] A.-C. Caron, J.-L. Coquide, and M. Dauchet. Encompassment properties and automata with constraints. In C. Kirchner, editor, *Proceedings 5th Conference on Rewriting Techniques and Applications, Montreal (Canada)*, volume 690 of *Lecture Notes in Computer Science*, pages 328–342. Springer-Verlag, 1993.

[5] M. Clavel, S. Eker, P. Lincoln, and J. Meseguer. Principles of Maude. In J. Meseguer, editor, *Proceedings of the 1st International Workshop on Rewriting Logic and its Applications*, volume 5 of *Electronic Notes in Theoretical Computer Science*, Asilomar, Pacific Grove, CA, USA, September 1996. North Holland.

[6] H. Comon. Sufficient completeness, term rewriting system and anti-unification. In J. Siekmann, editor, *Proceedings 8th International Conference on Automated Deduction, Oxford (UK)*, volume 230 of *Lecture Notes in Computer Science*, pages 128–140. Springer-Verlag, 1986.

[7] H. Comon. Inductionless induction. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 14, pages 913–962. Elsevier Science, 2001.

[8] H. Comon and F. Jacquemard. Ground reducibility is EXPTIME-complete. In *Proc. 12th IEEE Symp. Logic in Computer Science*, pages 26–34. IEEE Comp. Soc. Press, 1997.

[9] E. Contejean, C. Marché, B. Monate, and X. Urbain. CiME version 2, 2000. http://cime.lri.fr/.

[10] N. Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17:279–301, 1982.

[11] N. Dershowitz and J.-P. Jouannaud. *Handbook of Theoretical Computer Science*, volume B, chapter 6: Rewrite Systems, pages 244–320. Elsevier Science Publishers B. V. (North-Holland), 1990. Also as: Research report 478, LRI.

[12] M. C. et al. Maude: specification and programming in rewriting logic. *Theoretical Computer Science*, 285:187–243, August 2002.

[13] O. Fissore, I. Gnaedig, and H. Kirchner. Termination of rewriting with local strategies. In M. P. Bonacina and B. Gramlich, editors, *Selected papers of the 4th International Workshop on Strategies in Automated Deduction*, volume 58 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers B. V. (North-Holland), 2001.

[14] O. Fissore, I. Gnaedig, and H. Kirchner. Outermost ground termination. In *Proceedings of the 4th International Workshop on Rewriting Logic and Its Applications*, volume 71 of *Electronic Notes in Theoretical Computer Science*, Pisa, Italy, September 2002. Elsevier Science Publishers B. V. (North-Holland).

[15] O. Fissore, I. Gnaedig, and H. Kirchner. A proof of weak termination providing the right way to terminate. In *1st International Colloquium on THEORETICAL ASPECTS OF COMPUTING*, volume 3407 of *Lecture Notes in Computer Science*, pages 356–371, Guiyang, China, September 2004. Springer-Verlag.

[16] O. Fissore, I. Gnaedig, and H. Kirchner. Proving weak termination also provides the right way to terminate - Extended version. Technical report, LORIA, Nancy (France), March 2004. Available at http://www.loria.fr/∼gnaedig/PAPERS/REPORTS/wt-extended-2004.ps.

[17] K. Futatsugi and A. Nakagawa. An overview of CAFE specification environment – an algebraic approach for creating, verifying, and maintaining formal specifications over networks. In *Proceedings of the 1st IEEE Int. Conference on Formal Engineering Methods*, 1997.

[18] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Improving dependency pairs. In *Proceedings of the 10th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR '03)*, volume 2850 of *Lecture Notes in Artificial Intelligence*, pages 165–179, Almaty, Kazakhstan, 2003. Springer-Verlag.

[19] I. Gnaedig and H. Kirchner. Computing Constructor Forms with Non Terminating Rewrite Programs - Extended version. Technical report, LORIA, Nancy (France), 2006. Available at http://www.loria.fr/∼gnaedig/PAPERS/REPORTS/comp-extended-2006.pdf.

[20] I. Gnaedig, H. Kirchner, and O. Fissore. Induction for innermost and outermost ground termination. Technical Report A01-R-178, LORIA, Nancy (France), September 2001.

[21] G. Huet and J.-M. Hullot. Proofs by induction in equational theories with constructors. *Journal of Computer and System Sciences*, 25(2):239–266, Oct. 1982. Preliminary version in Proceedings 21st Symposium on Foundations of Computer Science, IEEE, 1980.

[22] J.-P. Jouannaud and E. Kounalis. Automatic proofs by induction in theories without constructors. *Information and Computation*, 82:1–33, 1989.

[23] D. Kapur, P. Narendran, and H. Zhang. Proof by induction using test sets. In *Proceedings 8th International Conference on Automated Deduction, Oxford (UK)*, volume 230 of *Lecture Notes in Computer Science*, pages 99–117. Springer-Verlag, 1986.

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

[25] P. Klint. A meta-environment for generating programming environments. *ACM Transactions on Software Engineering and Methodology*, 2:176–201, 1993.

[26] E. Kounalis. Completeness in data type specifications. In B. Buchberger, editor, *Proceedings EUROCAL Conference, Linz (Austria)*, volume 204 of *Lecture Notes in Computer Science*, pages 348–362. Springer-Verlag, 1985.

[27] E. Kounalis. Testing for the ground (co-)reducibility property in term-rewriting systems. *Theoretical Computer Science*, 106:87–117, 1992.

[28] A. Lazrek, P. Lescanne, and J.-J. Thiel. Tools for proving inductive equalities, relative completeness and ω-completeness. *Information and Computation*, 84(1):47–70, Jan. 1990.

[29] P.-E. Moreau, C. Ringeissen, and M. Vittek. A Pattern Matching Compiler for Multiple Target Languages. In G. Hedin, editor, *12th Conference on Compiler Construction, Warsaw (Poland)*, volume 2622 of *LNCS*, pages 61–76. Springer-Verlag, May 2003.

[30] T. Nipkow and G. Weikum. A decidability result about sufficient completeness of axiomatically specified abstract data types. In *6th GI Conference*, volume 145 of *Lecture Notes in Computer Science*, pages 257–268. Springer-Verlag, 1983.

[31] D. Plaisted. Semantic confluence tests and completion methods. *Information and Control*, 65:182–215, 1985.