Computing Constructor Forms with Non Terminating Rewrite Programs - Extended version -
Isabelle Gnaedig, Hélène Kirchner

To cite this version:
Isabelle Gnaedig, Hélène Kirchner. Computing Constructor Forms with Non Terminating Rewrite Programs - Extended version -. [Research Report] 2006, pp.17. inria-00113146

HAL Id: inria-00113146
https://inria.hal.science/inria-00113146
Submitted on 10 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
- Extended version -

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 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 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 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 [24], Maude [5, 12], Cafe-OBJ [17], ELAN [1], or TOM [28], 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 [20, 29, 25, 22, 6, 21, 27, 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 [21]. Under these conditions, techniques developed for proving ground reducibility hold for sufficient completeness as well [30, 23, 21, 26, 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 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 proper-
ties from an inductive point of view. We already have pro-
posed proof procedures for the termination property, in the
specific case of rewriting strategies, respectively for the in-
nermost [19], 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, 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 complete-
ness, 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 it-
self, assuming that the terms t smaller than the starting
terms for an induction ordering are 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 rep-
resenting at least one rewriting derivation for every ground
term. The proof is constructive, in the sense that the rewrit-
ing branch leading to a constructor term can be computed
from the proof trees establishing C-reducibility for every
term, thus avoiding costly breadth-first computations to re-
ach the constructor form. Although our method differs from
previous approaches, we naturally encounter basic notions
already used for proving sufficient completeness, as unifica-
tion, used in [27], or covering properties used in many works
in the domain, for example in [25, 27]. 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 C-reducibility to
close properties like strong C-reducibility, expressing the exis-
tence 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, 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 deriv-
a tion) to imply sufficient completeness, thus weakening the
condition of the well-known theorem of [21]. 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 knowl-
edge, our proof procedure [15] is the only one existing
today. On the following small example, weakly terminat-
ing, 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:

\[
\begin{align*}
\text{and}(1, x) & \rightarrow x \\
\text{and}(0, x) & \rightarrow 0 \\
\text{or}(1, x) & \rightarrow 1 \\
\text{or}(0, x) & \rightarrow x \\
\text{not}(\text{not}(\text{and}(1, x))) & \rightarrow \text{not}(0) \\
\text{not}(1) & \rightarrow 0 \\
\text{not}(0) & \rightarrow 1 \\
\text{not}(\text{and}(x, y)) & \rightarrow \text{or}(\text{not}(x), \text{not}(y)).
\end{align*}
\]

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

\[
\begin{align*}
\text{quot}(0, s(y), s(z)) & \rightarrow 0 \\
\text{quot}(s(x), s(y), z) & \rightarrow \text{quot}(x, y, z) \\
\text{quot}(x, 0, s(z)) & \rightarrow s(\text{quot}(x, s(z), s(z))) \\
\text{quot}(x, y, 0) & \rightarrow \text{error} \\
\text{quot}(\text{error}, y, z) & \rightarrow \text{error} \\
\text{quot}(x, \text{error}, z) & \rightarrow \text{error} \\
\text{quot}(x, y, \text{error}) & \rightarrow \text{error}.
\end{align*}
\]

In addition, if the RS has no rule with a constructor left-
hand side, our proof technique establishes weak termina-
tion 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 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 C-reducibility of terms.

2. The background

We assume that the reader is familiar with the basic def-
initions and notations of term rewriting given for instance in
[11]. \( T(\mathcal{F}, \mathcal{X}) \) is the set of terms built from a given set
\( \mathcal{F} \) of function symbols having an \( \text{arity} \) \( ar = n \in \mathbb{N} \),
and a set \( \mathcal{X} \) of variables denoted \( x, y, \ldots \). In this paper, we only
consider finite sets of symbols. \( T(\mathcal{F}) \) 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 \( \epsilon \) denotes the top
position. The notation \( t(i_p) \) stands for the subterm of \( t \)
at position \( p \). To emphasize that \( u \) contains subterms \( \tau \), \( j \in [1..p] \) respectively
at positions \( \{i_1, \ldots, i_p\} \), we write \( u[\tau]_{(i_1, \ldots, i_p)} \).

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

A ground substitution or instantiation is an assignment
from \( \mathcal{X} \) to \( T(\mathcal{F}) \). The composition of substitutions \( \sigma_1 \) fol-

lowed by $\sigma_2$ is denoted $\sigma_2 \sigma_1$. Given two substitutions $\sigma_1$ and $\sigma_2$, we write $\sigma_1 \leq \sigma_2$ iff $\exists \theta$ such that $\sigma_2 = \theta \sigma_1$. We say that $\sigma_1$ is a generalization of $\sigma_2$.

For $\mathcal{F}$, we declare a set $\mathcal{C}$ of constructors and a set $\mathcal{D}$ of defined symbols. The terms of $T(\mathcal{C})$ 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 $T(\mathcal{F}, \mathcal{X})$, the rewriting relation induced by $\mathcal{R}$ is denoted by $\rightarrow_\mathcal{R}$ ($\rightarrow$ if there is no ambiguity on $\mathcal{R}$). We denote $s \rightarrow_{p.l.r.}^* \sigma t$ (where either $p$ or $l \rightarrow r$ or $\sigma$ may be omitted) if $\sigma$ rewrites into $t$ at position $p$ with the rule $l \rightarrow r$ and the substitution $\sigma$, 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 $\mathcal{R}$ normal form if $t$ has no rules whose left-hand sides are constructor terms (or constructor $\mathcal{C}$-terms). Given a $\mathcal{R}$-reducible term $t$, we write this $\mathcal{C}$-form, $t \rightarrow_\mathcal{C}$, and say that it is a $\mathcal{C}$-reduced form for $\mathcal{R}$.

An ordering $\succ$ on $T(\mathcal{F}, \mathcal{X})$ is said to be noetherian if there is no infinite decreasing chain for this ordering. It is monotone iff for every pair of terms $t, t'$ of $T(\mathcal{F}, \mathcal{X})$, for every context $f(\ldots)$, $t \succ t'$ implies $f(t_1, \ldots) \succ f(t'_1, \ldots)$. It has the subterm property iff for every $t$ of $T(\mathcal{F}, \mathcal{X})$, $t \succ t'$ implies $f(t_1, \ldots) \succ f(t'_1, \ldots)$. It has the subterm property whenever $\mathcal{X}$ is a constructor and $\mathcal{R}$ 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 $\sigma$, every pair of terms $t, t'$ of $T(\mathcal{F}, \mathcal{X})$, $t \succ t'$ implies at $\sigma t', \sigma t'$), then it is called a simplification ordering. A precedence is an ordering on $\mathcal{F}$, denoted $\succ_\mathcal{F}$.

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 if for each rule $l \rightarrow r$ of $\mathcal{R}$, whenever $l \in T(\mathcal{C})$, then $r \in T(\mathcal{C})$.

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 $T(\mathcal{F}, \mathcal{X})$. 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 $T(\mathcal{F}, \mathcal{X})$, where $\mathcal{F} = \mathcal{C} \cup \mathcal{D}$. $\mathcal{E}$ is sufficiently complete (with respect to $\mathcal{D}$) if for every term $t \in T(\mathcal{F})$, there is a term $u \in T(\mathcal{C})$ such that $t \equiv_\mathcal{E} u$.

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

**Definition 4.** Let $\mathcal{R}$ be a RS on $T(\mathcal{F}, \mathcal{X})$, where $\mathcal{F} = \mathcal{C} \cup \mathcal{D}$. A 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) $\mathcal{C}$-reducible if every term $t$ of $T(\mathcal{F})$ 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 the 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 $T(\mathcal{F})$. 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 [21].
For proving that a term \( t \) of \( T(F) \) is \( C \)-reducible, we proceed by induction on \( T(F) \) with a noetherian ordering \( \succ \), assuming the property for any \( t' \) such that \( t \succ t' \). To warrant non emptiness of \( T(F) \) and \( T(C) \), we assume that \( C \) contains at least a constant. The main intuition is to observe rewriting derivations starting from a ground term \( t \in T(F) \) which is any instance of a term \( g(x_1, \ldots, x_m) \in T(F,X) \), for some defined function symbol \( g \in D \), and variables \( x_1, \ldots, x_m \). Proving the \( C \)-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 \( T(F,X) \), for every \( g \in D \), 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 \( C \)-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 \( T(F,X) \), 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 \( T(F) \) as a ground instance of a term \( t \) of \( T(F,X) \) occurring in a proof tree issued from a reference term \( t_{ref} \).

- first, some subterms \( t_i \) of the current term \( t \) of the proof tree are supposed to reduce into a \( C \)-term, by the induction hypothesis, if \( \theta_{ref} \succ \theta t_i \) for the induction ordering \( \succ \) and for every \( \theta \) 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 \( C \)-reducible. Else,

- the resulting term \( u = (X_j)_{i1, \ldots, i_p} \) (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 \( C \)-term.

However, we have to ensure that every ground term represented by \( u \) is \( C \)-reducible, and so we have to prove the existence of at least one branch leading to a \( C \)-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 \( C \)-reducibility of the ground instances of \( t \) is reduced to \( C \)-reducibility of the ground instances of the terms \( v \). Now, if \( \theta_{ref} \succ \theta v \) for every ground substitution \( \theta \) that is a solution of the constraint associated to \( v \), by the induction hypothesis, \( \theta v \) is supposed to be \( C \)-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 \( C \)-term.

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

## 4. Inductively proving \( C \)-reducibility

### 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 \( T(F,X) \), we directly work with an ordering \( \succ_p \) on \( T(F,X) \) such that \( t_{ref} \succ_p u \) induces \( \theta_{ref} \succ \theta u \), for every \( \theta \) solution of the constraint associated to \( u \).

So inequalities of the form \( t_{ref} \succ u_1, \ldots, u_m \) are accumulated, which will be called ordering constraints. Any ordering \( \succ_p \) on \( T(F,X) \) satisfying them and which is stable by substitution fulfills the previous requirements on ground terms.

The ordering \( \succ_p \), defined on \( T(F,X) \), can then be seen as an extension of the induction ordering \( \succ \), defined on \( T(F) \).

For convenience, the ordering \( \succ_p \) 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 \( T(F,X) \) noted \((t \succ t')\). It is said to be satisfiable if there exists an ordering \( \succ \), such that for every instantiation \( \theta \) whose domain contains \( \text{Var}(t) \cup \text{Var}(t') \), we have \( \theta t \succ \theta t' \). We say that \( \succ \) satisfies \((t \succ t')\).

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 $t \succ t'$ 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 $t \succ t'$ 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 Cine2 [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 $t_{i_j}, j \in \{i_1, \ldots, i_p\}$ are supposed to have a reduced $C$-form $\sigma_{t_{i_j}}$, we replace the $t_{i_j}$ by abstraction variables $X_{i_j}$ representing respectively one of their possible $C$-forms. Let us define these special variables more formally.

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

**Definition 7 (abstraction).** The term $t[\{t_{i_1}\}[1_{i_1}] \ldots [i_p]]$ is said to be abstracted into the term $u$ (called abstraction of $t$) at positions $\{i_1, \ldots, i_p\}$ iff $u = t[X_{i_1}]_{[1_{i_1}] \ldots [i_p]}$, where the $X_{i_j}, j \in \{i_1, \ldots, i_p\}$ are fresh distinct $C$-variables.

$C$-reducibility on $T(F)$ is proved by reasoning on terms with abstraction variables, i.e. on terms of $T(F, X \cup N)$. Ordering constraints are extended to pairs of terms of $T(F, X \cup N)$. When subterms $t_{i_j}$ are abstracted by $X_{i_j}$, we state constraints on abstraction variables, called abstraction constraints, to express that their instances can only be $C$-terms, rewritten from the corresponding instances of $t_{i_j}$. Initially, they are of the form $t_{i_j} = X$ where $t \in T(F, X \cup N)$, and $X \in N$, 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 $X_{i_j}$. This is achieved by narrowing $u$ in all possible ways. We recall the definition of narrowing.

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

\sigma(l) = \sigma(t[p])$ and $t' = \sigma(t[r_p])$

where $\sigma$ 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 $\sigma$ 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 $\sigma$ only fresh variables. Moreover, narrowing is only performed on terms $t$ of $T(F, N)$, since an abstracting step is first applied on the reference terms, of the form $g(x_1, \ldots, x_m)$, replacing $x_1, \ldots, x_m \in X$ by $X_1, \ldots, X_m \in N$. Then from Definition 6 we infer that for the most general unifiers $\sigma$ produced during the proof process, all variables of $Ran(\sigma)$ are $C$-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 $C$-reducible, since, although all ground instances of $u$ are not reducible at that step, they may be reduced to a $C$-form by other rewriting derivations, not observed by the process, as shows the following example. Let $F = \{f(a, b, c), c = (b, c), R = \{a \rightarrow b, f(f(b)) \rightarrow c, b \rightarrow f(b), f(c) \rightarrow c\}$. Abstracting the pattern $f(x)$ gives $f(X)$, where $X$ only represents $C$-forms. Then, $f(X)$ is only narrowed into $c$ for $X = c$. The narrowing substitution $X = f(b)$ is not allowed since $f(b) \notin T(C, N)$. However, the other ground instance of $f(X)$, which is $f(b)$, is also $C$-reducible, but the rewriting chain $f(b) \rightarrow f(f(b)) \rightarrow c$ is not represented by the abstract-narrow-based process.

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

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

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

As a first narrowing step is applied on the patterns $g(x_1, \ldots, X_m), g \in D$, 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 $g(x_1, \ldots, X_m), g \in D$ is stronger than usual ground reducibility (i.e. of the pattern $g(x_1, \ldots, x_m)$). The first one is an obvious consequence of Proposition 1.

**Proposition 2.** Let $R$ be a $RS$, $g \in D$, $X_1, \ldots, X_m \in N$. Let $\Sigma$ be the set of narrowing substitutions of $g(x_1, \ldots, X_m)$ with $R$. If $\Sigma$ 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 \( F = \{ f, 0, 1 \} \), \( C = \{ 0, 1 \} \), and \( R = \{ f(0) \rightarrow 0, 1 \rightarrow 0 \} \). The term \( f(X) \) is ground reducible since \( f(0) \) and \( f(1) \) are reducible. However, the set \( \Sigma \) of narrowing substitutions of \( f(X) \), equal to \( \{ \sigma \in \{ X = 0 \} \} \) is not covering for \( f(X) \): \( f(1) \) is not a ground instance of \( f(0) \).

**Proposition 3.** Let \( \Sigma \in \text{Narrow} \) on \( T(F, X) \), \( X_1, \ldots, X_m \in N \), \( x_1, \ldots, x_m \in X \). Then \( f(X_1, \ldots, X_m) \) is ground reducible for every \( f \in D \), iff \( f(x_1, \ldots, x_m) \) is ground reducible for every \( f \in D \).

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

Then \( \Sigma \) is \( u \)-covering if for every \( \sigma^p \in \Sigma^p \), there exists \( \sigma \in \Sigma \) such that \( \sigma^p v = \sigma v^\prime \) for some substitution \( v \); in other words, if for every \( \sigma^p \in \Sigma^p \), there exists in \( \Sigma \) a generalization of \( \sigma^p \). 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 \( \sigma \) represents a rewriting branch for any ground instance of \( \sigma u \).

In addition, \( \sigma \) has to satisfy the constraints on variables of \( u \), already set in \( A \). So \( \sigma \), 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 \( \bigwedge \{ t_{i_C} = t_i' \} \land \bigwedge \{ x_i = u_i \} \), where \( x_i \in X \cup N \), \( t_i, t_i', u_i \in T(F, X \cup N) \).

**Definition 11.** An abstraction constrained formula \( A = \bigwedge \{ t_{i_C} = t_i' \} \land \bigwedge \{ x_j = u_j \} \) is satisfiable iff there exists at least one instantiation \( \theta \) such that \( \bigwedge \{ (\theta x_i = \theta x_i') \land \bigwedge \{ (\theta x_j = \theta u_j) \} \}. \) The instantiation \( \theta \) is then said to satisfy the ACF \( A \) and is called solution of \( A \).

For a better readability on examples, we can propagate \( \sigma \) into \( A \) (by applying \( \sigma \) to \( A \)), thus getting instantiated abstraction constraints of the form \( t_{i_C} = t_i' \) from initial abstraction constraints of the form \( t_{i_C} = X_i \).

An ACF 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 \( C \)-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 sat-

ifying 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 \( t_{i_C} = u \), where \( u \) contains at least a symbol of \( D \).

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 of 3-tuples \((U, A, C)\) where \( U = \{ t \} \) or \( \emptyset \), \( t \) is the current term whose \( C \)-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 \( t_1, \ldots, t_p \). The constraints \( t_{i_C} > t_{i_1}, \ldots, t_{i_p} \) are set, allowing us to suppose, by induction, the existence of \( C \)-forms for \( t_{1, \ldots, t_p} \). Then, \( t_{i_1}, \ldots, t_{i_p} \) are abstracted into abstraction variables \( Y_{i_1}, \ldots, Y_{i_p} \). The abstraction constraint \( t_{i_1} = X_{i_1}, \ldots, t_{i_p} = X_{i_p} \) 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 \( t_i = g(t_{i_1}, \ldots, t_{i_n}) \) that cannot be abstracted, we try to abstract \( t_{i_1}, \ldots, t_{i_n} \), 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 \( T(C, N) \). Indeed, by Definition 6, every ground instance of such subterms is already a \( C \)-term.

- The second rule narrows the resulting term \( u \), if it is not a term of \( T(C, N) \), in all possible ways in one step, with all possible rewrite rules of the rewrite system \( R \), and all possible substitutions \( \sigma_1, \ldots, \sigma_m \), into terms \( v_1, \ldots, v_k \), according to Definition 8, if \( \{ \sigma_1, \ldots, \sigma_m \} \) is \( u \)-covering.

  This step is a branching step, creating as many states as narrowing possibilities. The substitution \( \sigma \) is integrated to \( A \). This is the **Narrow** rule. As \( \{ \sigma_1, \ldots, \sigma_m \} \) 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 \( u \in T(C, N) \).

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 \( \succ \) is such that \( f \succ c \), \( \forall f \in D, \forall c \in C \), we have \( t \succ u \) for \( t \in T(F, X) \) containing at least a symbol of \( D \) and \( u \in T(C) \).

Exploiting equalities in \( A \) can also help to solve ordering constraints: \( t_{i_C} = X_i \) means that \( \theta X_i \) is a \( C \)-form obtained by rewriting \( \theta t_i \), for every ground substitution \( \theta \). As we do not abstract terms of \( T(C, N) \), if \( t_i \neq x_i \in X \), \( \theta t_i \) contains at least a symbol of \( D \), and then \( \theta t_i \succ \theta X_i \) for a RPO or an
LPO defined as above. As \( t_{\text{ref}} \) also contains a symbol of \( D \), we also have \( \theta t_{\text{ref}} \succ \theta X \) 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 \( C \)-reducibility of every ground instance of \( u \) by another way. Let \( C - \text{RED}(u) \) be a predicate that is true iff every ground instance of \( u \) is \( C \)-reducible. In the first and third previous steps of the inductive reasoning, we then associate the alternative predicate \( C - \text{RED}(u) \) to the condition \( t > u \). It is true in particular if \( u \in \mathcal{T}(C, \mathcal{N}) \), as said in the presentation of the \textbf{Stop} rule.

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

The strategy \( S \) repeats the following steps: first, apply \textbf{Abstract}, and then try \textbf{Stop}. Then try all possible applications of \textbf{Narrow}. Then, try \textbf{Stop} again.

Note that if \( A \) is satisfiable, the transformed forms of \( A \) by \textbf{Abstract} and \textbf{Stop} are also satisfiable. Moreover, the first application of \textbf{Abstract} generates \( A = (\land, x_{1 \in C} = x_1) \), always satisfied by the constructor constant supposed to exist in \( C \). Thus with the strategy \( S \), it is useless to prove satisfiability of \( A \) in the rules \textbf{Abstract} and \textbf{Stop}.

The process may not terminate if there is an infinite number of applications of \textbf{Abstract} and \textbf{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 \textbf{Stop}, i.e. if its final state is of the form \( \emptyset, A, C \).

Thus, the inductive \( C \)-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 \textbf{Narrow}, can generate different states with narrowing substitutions \( \sigma_1, \ldots, \sigma_n \). These substitutions can be compared. For \( \sigma_i \) and \( \sigma_j \), three situations may occur: \( \sigma_i \) is strictly less general than \( \sigma_j \), which is noted \( \sigma_i > \sigma_j \), or \( \sigma_i \) is strictly less general than \( \sigma_j \), \( \sigma_i \) and \( \sigma_j \) are equal up to a renaming, or else \( \sigma_i \) and \( \sigma_j \) 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 \( C \)-reducibility for all ground instances at a branching point, it is sufficient to prove \( C \)-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 \( \Sigma \) be the set of narrowing substitutions (possibly with different rewrite rules) at a given branching node. Let \( \Sigma_0 \) be the reduced set from \( \Sigma \) such that \( \sigma \in \Sigma_0 \) iff \( \sigma \in \Sigma \) and \( \beta \sigma' \in \Sigma \) such that \( \sigma > \sigma' \) on \( (\text{Dom}(\sigma) \setminus \text{Var}(l')) \cup (\text{Dom}(\sigma') \setminus \text{Var}(l')) \), where \( l \) and \( l' \) are the left-hand sides of rules respectively used to produce the narrowing substitutions \( \sigma \) and \( \sigma' \). The set \( \Sigma_0 \) may yet contain equivalent (equal up to a renaming) substitutions which are marked as such. So for any two substitutions in \( \Sigma_0 \), either they are equivalent, or they are incomparable.

A proof tree is \textit{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 \( \Sigma \), we only develop the subtrees corresponding to \( \Sigma_0 \). Moreover, given two subtrees corresponding to equivalent substitutions, as soon as one of them is weakly successful, the other one is cut.

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

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

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

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

\[
\begin{align*}
\text{and}(1, x) & \rightarrow x \\
\text{and}(0, x) & \rightarrow 0 \\
\text{or}(1, x) & \rightarrow 1 \\
\text{or}(0, x) & \rightarrow x \\
\text{and}(1, x) & \rightarrow \text{not}(\text{not}(\text{and}(1, x))) \\
\text{not}(1) & \rightarrow 0 \\
\text{not}(0) & \rightarrow 1 \\
\text{not}(\text{and}(x, y)) & \rightarrow \text{or}(\text{not}(x), \text{not}(y))
\end{align*}
\]

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

Applying the rules on \( \text{not}(x) \), we get:

\[
\begin{align*}
\text{not}(x_1) & \rightarrow A = \top, \quad C = \top \\
\text{Abstract} \quad \text{not}(X_1) & \rightarrow A = (x_{1 \in C} = x_1) \\
& \quad C = (\text{not}(x_1) > x_1)
\end{align*}
\]

\[
\begin{align*}
\text{Narrow} \quad \sigma_1 = (X_1 = 0) & \rightarrow A = (x_{1 \in C} = 0) \\
& \quad C = (\text{not}(x_1) > x_1) \\
\sigma_2 = (X_1 = 1) & \rightarrow A = (x_{1 \in C} = 1) \\
& \quad C = (\text{not}(x_1) > x_1)
\end{align*}
\]

\[
\begin{align*}
\text{Stop} (\text{twice}) & \rightarrow A = (x_{1 \in C} = 0) \\
& \quad C = (\text{not}(x_1) > x_1) \\
& \rightarrow A = (x_{1 \in C} = 1) \\
& \quad C = (\text{not}(x_1) > x_1)
\end{align*}
\]

\[
\begin{align*}
\text{Narrow} & \rightarrow A = (X_1 = \text{and}(X_2, X_3)) \\
& \quad \text{but } \sigma_3 \text{ is not licit since } \text{and}(X_2, X_3) \notin \mathcal{T}(C, \mathcal{N}).
\end{align*}
\]
Moreover \( \Sigma = \{ \sigma_1, \sigma_2, \sigma_3 \} \) is not\( (X_1) \)-covering since for every possible ground instance \( \theta X_1 \) of \( X_1 \) (0 or 1), there exists \( \sigma \) in \( \Sigma \) such that \( \theta \) not\( (X_1) = \mu \sigma \) not\( (X_1) \) for some \( \mu \), which is here equal to \( Id \).

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

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:

\[
\begin{align*}
\text{Abstract:} & \quad A = \top, \quad C = \top \\
\text{or}(x_1, x_2) & \quad \text{or}(x_1, x_2) \\
\text{Narrow:} & \quad \sigma_1 = (X_1 = 0) \\
& \quad A = (x_1 \downarrow_c = 0 \wedge x_2 \downarrow_c = X_2) \\
& \quad C = (or(x_1, x_2) > x_1, x_2) \\
& \quad \sigma_2 = (X_1 = 1) \\
& \quad A = (x_1 \downarrow_c = 1 \wedge x_2 \downarrow_c = X_2) \\
& \quad C = (or(x_1, x_2) > x_1, x_2) \\
\text{Stop(twice)} & \quad \emptyset \\
& \quad A = (x_1 \downarrow_c = 0 \wedge x_2 \downarrow_c = X_2) \\
& \quad C = (or(x_1, x_2) > x_1, x_2)
\end{align*}
\]

\( \Sigma = \{ \sigma_1, \sigma_2 \} \) is \( or(X_1, X_2) \)-covering.

**Stop** applies since \( X_2 \) and 1 are terms of \( T(\mathcal{C}, \mathcal{N}) \).

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

\[
\begin{align*}
\text{and}(x_1, x_2) & \quad A = \top, \quad C = \top \\
\text{Abstract:} & \quad A = (x_1 \downarrow_c = X_1 \wedge x_2 \downarrow_c = X_2) \\
& \quad C = (and(x_1, x_2) > x_1, x_2) \\
\text{Narrow:} & \quad \sigma_1 = (X_1 = 0) \\
& \quad A = (x_1 \downarrow_c = 0 \wedge x_2 \downarrow_c = X_2) \\
& \quad C = (and(X_1, X_2) > x_1, x_2) \\
& \quad \sigma_2 = (X_1 = 1) \\
& \quad A = (x_1 \downarrow_c = 1 \wedge x_2 \downarrow_c = X_2) \\
& \quad C = (and(X_1, X_2) > x_1, x_2) \\
\text{Stop(three times)} & \quad \emptyset \\
& \quad A = (x_1 \downarrow_c = 0 \wedge x_2 \downarrow_c = X_2) \\
& \quad C = (and(X_1, X_2) > x_1, x_2)
\end{align*}
\]

\( \Sigma = \{ \sigma_1, \sigma_2, \sigma_3 \} \) is \( and(X_1, X_2) \)-covering.

**Stop** applies on the two first branches because 0 and \( X_2 \) are terms of \( T(\mathcal{C}, \mathcal{N}) \), 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 \( R \) 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 \( R \) the relations between constructors

\[
\begin{align*}
1 & \rightarrow \text{true} \\
\text{true} & \rightarrow 1 \\
0 & \rightarrow \text{false} \\
\text{false} & \rightarrow 0
\end{align*}
\]

where \text{true} and \text{false} are also constructor symbols, we lose the weak termination property. But \( C \)-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:**

\[
\begin{align*}
\text{quot}(0, s(y), s(z)) & \rightarrow 0 \\
\text{quot}(s(x), s(y), z) & \rightarrow \text{quot}(x, y, z) \\
\text{quot}(x, 0, s(z)) & \rightarrow \text{quot}(x, s(z), s(z)) \\
\text{quot}(x, y, 0) & \rightarrow \text{error} \\
\text{quot}(\text{error}, y, z) & \rightarrow \text{error} \\
\text{quot}(x, \text{error}, z) & \rightarrow \text{error} \\
\text{quot}(x, y, \text{error}) & \rightarrow \text{error}
\end{align*}
\]

with \( F = \{ \text{quot} : 3, s : 1, 0 : 0, \text{error} : 0 \} \) and \( C = \{ s, 0, \text{error} \} \).

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 \( C \)-reducibility proof method. Let us develop the second solution.

Applying the rules on \( \text{quot}(x_1, x_2, x_3) \), we get:

\[
\begin{align*}
\text{quot}(x_1, x_2, x_3) & \quad A = \top, \quad C = \top \\
\text{Abstract} & \quad \text{quot}(X_1, X_2, X_3) = (X_1 \mid_c = X_1 \land X_2 \mid_c = X_2 \\
& \land X_3 \mid_c = X_3) \\
& \quad C = (\text{quot}(x_1, x_2, x_3) > x_1, x_2, x_3) \\
\text{Narrow} & \quad 0 \\
& \quad \sigma_1 = (X_1 = 0 \land X_2 = s(X'_2) \\
& \quad \land X_3 = s(X'_3)) \\
& \quad A = (x_1 \mid_c = 0 \land x_2 \mid_c = s(X'_2) \\
& \quad \land x_3 \mid_c = s(X'_3)) \\
& \quad C = (\text{quot}(x_1, x_2, x_3) > x_1, x_2, x_3) \\
\text{quot}(X'_1, X'_2, X_3) & \quad 0 \\
& \quad \sigma_2 = (X_1 = s(X'_1) \land X_2 = s(X'_2) \\
& \quad A = (x_1 \mid_c = s(X'_1) \\
& \quad \land x_2 \mid_c = s(X'_2) \land x_3 \mid_c = X_3) \\
& \quad C = (\text{quot}(x_1, x_2, x_3) > x_1, x_2, x_3)
\end{align*}
\]

Applying the rule **Narrow** here gives seven branches, following the seven rules of \( R \). Let \( u = \text{quot}(X_1, X_2, X_3) \).

The set \( \Sigma = \{ \sigma_i, i \in [1, 7] \} \) is u-covering, since every \( \sigma_i \) of \( \Sigma \), where \( P = \{ 0, \text{error}, s(X) \mid X \in \mathcal{N} \} \), has a generalization in \( \Sigma \).

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 \( C \)-terms as current terms.

On the second branch, we have \( \text{quot}(x_1, x_2, x_3) \ntriangleright \text{quot}(X'_1, X'_2, X_3) \) for a Lexicographic Path Ordering with any precedence and a left-to-right status for \( \text{quot} \). Indeed, \( R \) does not contain any rule whose left-hand side is a \( C \)-term. Then, if \( x_1 \) represents a \( C \)-term, it is in normal form, and then \( x_1 = x_1 \mid_c \). In \( A \), we have \( x_1 \mid_c = s(X'_1) \). Then \( x_1 \ntriangleright X'_1 \). If \( x_1 \) represents a term containing a defined symbol, then \( x_1 \ntriangleright X'_1 \) (see Section 5). In a similar way, we obtain \( x_2 \ntriangleright X'_2 \).

Now, if \( x_3 \) represents a term containing a defined symbol, as previously for \( x_1 \), we have \( x_3 \ntriangleright X'_3 \). If \( x_3 \) represents a \( C \)-term, we get \( x_3 = x_3 \mid_c = X_3 \), so \( x_3 \ntriangleright X_3 \).

By definition of the LPO, as we have \( x_1 \ntriangleright X'_1 \), we have to verify that \( \text{quot}(x_1, x_2, x_3) \ntriangleright X'_2, X_3 \). This is true since \( x_2 \ntriangleright X'_2 \) and \( x_3 \ntriangleright X_3 \).

Then we get:

\[
\begin{align*}
\text{Stop (six times)} & \quad 0 \\
& \quad A = (x_1 \mid_c = 0 \land x_2 \mid_c = s(X'_2) \\
& \land x_3 \mid_c = s(X'_3)) \\
& \quad C = (\text{quot}(x_1, x_2, x_3) > x_1, x_2, x_3)
\end{align*}
\]
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 C-form. It allows us extracting the exact application of rewrite rules that yields a 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 C-form for any term.

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

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

- if \( f \in C \), then \( Cform(s) = f(Cform(s_1), \ldots , Cform(s_m)) \),
- if \( f \in D \), then reducing \( s \) with respect to \( ST \) into \( Cform(s) \) is performed by following the steps in the strategy tree \( ST_f \) of \( f \), where \( f \) 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 \( t \mapsto t[i'_j] \in \{i_1, \ldots , i_p\} \),
  - where \( t'_j \) = \[ t[i'_j] \leftrightarrow Cform(t[i'_j]) \] if \( C\text{-RED}(u) \), otherwise,
  - if the step is **Narrow** with \( u \mapsto u' \) \( \beta \), then \( t \mapsto t' \) where \( t' \) is defined by \( t \mapsto t' \) \( \beta \) \( \alpha \), with \( \alpha = \beta \sigma \) on \( Var(u) \cup Dom(u) \) and \( t \mapsto t' \) otherwise,
  - if the step is **Stop**, then \( t \mapsto t' \),
  - where \( t' \) = \[ t[i'_j] \leftrightarrow Cform(t[i'_j]) \] if \( C\text{-RED}(u) \), otherwise.

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

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

7. Some additional examples

**Example:**

The following RS is neither terminating, nor even innermost terminating:

\[
\begin{align*}
f(x) & \rightarrow \text{if}(x, c, f(true)) \\
is\text{if}(true, x, y) & \rightarrow x \\
is\text{if}(\text{false}, x, y) & \rightarrow y \\
is\text{if}(c, x, y) & \rightarrow c
\end{align*}
\]
with $\mathcal{F} = \{ f : 3, s : 1, f : 1, c : 0, \text{true} : 0, \text{false} : 0 \}$ and $\mathcal{C} = \{ c, \text{true}, \text{false} \}$. Applying the rules on $f(x_1)$, we get:

$$ f(x_1) \quad A = \top, \quad C = \top $$

**Abstract**

$f(X_1)$

$$ A = (x_1|c = X_1) $$

$$ C = (f(x_1) > x_1) $$

**Narrow**

$\text{if}(X_1, c, f(\text{true}))$

$$ \sigma = \text{Id} $$

$$ A = (x_1|c = X_1) $$

$$ C = (f(x_1) > x_1) $$

**Narrow**

$c$

$$ \sigma_1 = (X_1 = \text{true}) $$

$$ A = (x_1|c = \text{true}) $$

$$ C = (f(x_1) > x_1) $$

$$ f(\text{true})$

$$ \sigma_2 = (X_1 = \text{false}) $$

$$ A = (x_1|c = \text{false}) $$

$$ C = (f(x_1) > x_1) $$

$$ c$

$$ \sigma_3 = (X_1 = c) $$

$$ A = (x_1|c = c) $$

$$ C = (f(x_1) > x_1) $$

$\text{if}(X_1, c, \text{if}(\text{true}, c, f(\text{true})))$

$$ \sigma_4 = \text{Id} $$

$$ A = (x_1|c = X_1) $$

$$ C = (f(x_1) > x_1) $$

The first **Narrow** is covering for $f(X_1)$ and the second, for $\text{if}(X_1, c, f(\text{true})).$

As the three first branches above are issued from narrowing substitutions, which are less general that the substitution of the fourth branch, they can be cut. We then only keep the last state, on which **Narrow** (covering for $\text{if}(X_1, c, \text{if}(\text{true}, c, f(\text{true})))$) is applied again, which gives the following states.

$c$

$$ \sigma_1 = (X_1 = \text{true}) $$

$$ A = (x_1|c = \text{true}) $$

$$ C = (f(x_1) > x_1) $$

$\text{if}(\text{true}, c, f(\text{true}))$

$$ \sigma_2 = (X_1 = \text{false}) $$

$$ A = (x_1|c = \text{false}) $$

$$ C = (f(x_1) > x_1) $$

$c$

$$ \sigma_3 = (X_1 = c) $$

$$ A = (x_1|c = c) $$

$$ C = (f(x_1) > x_1) $$

$\text{if}(X_1, c, c)$

$$ \sigma_4 = \text{Id} $$

$$ A = (x_1|c = X_1) $$

$$ C = (f(x_1) > x_1) $$

$\text{if}(X_1, c, \text{if}(\text{true}, c, f(\text{true})))$

$$ \sigma_5 = \text{Id} $$

$$ A = (x_1|c = X_1) $$

$$ C = (f(x_1) > x_1) $$

Again, we can cut the three first branches. **Stop** applies on the fourth state since we have $f(x_1) \succ \text{if}(X_1, c, c)$ for a RPO with the precedence $f \succ \text{id}, f, c$. Indeed, we have $f(x_1) \succ c$, and $f(x_1) \succ X_1$ following the remarks of Section 5.

Then, **Stop** applies on the three branches, which ends the proof.

$$ \emptyset \quad A = (x_1|c = \text{true}, x_2|c = X_2, x_3|c = X_3) $$

$$ C = (f(x_1, x_2, x_3) > x_1, x_2, x_3) $$

$$ \emptyset \quad A = (x_1|c = \text{false}, x_2|c = X_2, x_3|c = X_3) $$

$$ C = (f(x_1, x_2, x_3) > x_1, x_2, x_3) $$

$$ \emptyset \quad A = (x_1|c = c, x_2|c = X_2, x_3|c = X_3) $$

$$ C = (f(x_1, x_2, x_3) > x_1, x_2, x_3) $$

**Example:** The following RS is perhaps terminating, but it cannot be proved with a path ordering:

$$ \text{length}(\text{nil}) \rightarrow 0 $$

$$ \text{length}(\text{cons}(x,y)) \rightarrow \text{length}(1\text{y}) $$

$$ \text{length}(x) \rightarrow \text{length}(1\text{x}) $$

$$ \text{length}(0) \rightarrow \text{s}(0) $$

$$ \text{length}(s(x)) \rightarrow \text{length}(x) $$

with $\mathcal{F} = \{ \text{cons} : 2, \text{length} : 1, \text{length}1 : 1, s : 1, 0 : 0, \text{nil} : 0 \}$ and $\mathcal{C} = \{ 0, s, \text{cons}, \text{nil} \}$. Applying the rules on $\text{length}(x_1)$, we get:

$$ \text{length}(x_1) \quad A = \top, \quad C = \top $$

**Abstract**

$\text{length}(X_1)$

$$ A = (x_1|c = X_1) $$

$$ C = (\text{length}(x_1) > x_1) $$
Narrow 0
\[ \sigma = (X_1 = nil) \]
\[ A = (x_1 \downarrow_c = nil) \]
\[ C = (\text{length}(x_1) > x_1) \]
\[ s(0) \]
\[ \sigma = (X_1 = 0) \]
\[ A = (x_1 \downarrow_c = 0) \]
\[ C = (\text{length}(x_1) > x_1) \]
\[ \text{Stop} \]
\[ s(\text{length}(X_2)) \]
\[ \sigma = (X_1 = s(X_2)) \]
\[ A = (x_1 \downarrow_c = s(X_2)) \]
\[ C = (\text{length}(x_1) > x_1) \]
\[ s(\text{length}(X_4)) \]
\[ \sigma = (X_1 = \text{cons}(X_3, X_4)) \]
\[ A = (x_1 \downarrow_c = \text{cons}(X_3, X_4)) \]
\[ C = (\text{length}(x_1) > x_1) \]

The first narrowing step is covering for \text{length}(X_1), and the second one for \text{length}(X_3).

Now, Stop applies four times, like in the previous proof tree, but with \text{length}(x_1) as reference term instead of \text{length}(X_1). With the additional precedence \text{length}(1) \succ s, we get \text{length}(x_1) \succ s(\text{length}(X_2)) and \text{length}(x_1) \succ s(\text{length}(X_4)).

\[ \text{Stop} \]
\[ \emptyset \]
\[ A = (x_1 \downarrow_c = \text{nil}) \]
\[ C = (\text{length}(x_1) > x_1) \]
\[ \emptyset \]
\[ A = (x_1 \downarrow_c = 0) \]
\[ C = (\text{length}(x_1) > x_1) \]
\[ \emptyset \]
\[ A = (x_1 \downarrow_c = s(X_2), s(\text{length}(X_2)) \downarrow_c = X_3) \]
\[ C = (\text{length}(x_1) > x_1, s(\text{length}(X_2))) \]
\[ \emptyset \]
\[ A = (x_1 \downarrow_c = \text{cons}(X_3, X_4), s(\text{length}(X_4)) \downarrow_c = X_6) \]
\[ C = (\text{length}(x_1) > x_1, s(\text{length}(X_4))) \]

Example:

\[ \text{incr}(\text{nil}) \rightarrow \text{nil} \]
\[ \text{incr}(\text{cons}(x, l)) \rightarrow \text{cons}(s(x), \text{incr}(l)) \]
\[ \text{adx}(\text{nil}) \rightarrow \text{nil} \]
\[ \text{adx}(\text{cons}(x, l)) \rightarrow \text{inc}(\text{cons}(x, \text{adx}(l))) \]
\[ \text{nats} \rightarrow \text{adx}(\text{zeros}) \]
\[ \text{zeros} \rightarrow \text{cons}(0, \text{zeros}) \]
\[ \text{head}(\text{cons}(x, l)) \rightarrow x \]
\[ \text{tail}(\text{cons}(x, l)) \rightarrow l \]
\[ \text{adx}(0) \rightarrow \text{err} \]
\[ \text{adx}(s(x)) \rightarrow \text{err} \]
\[ \text{ incr}(0) \rightarrow \text{err} \]
\[ \text{ incr}(s(x)) \rightarrow \text{err} \]
\[ \text{tail}(\text{nil}) \rightarrow \text{nil} \]
\[ \text{tail}(0) \rightarrow \text{err} \]
\[ \text{tail}(s(x)) \rightarrow \text{err} \]
\[ \text{head}(\text{nil}) \rightarrow \text{nil} \]
\[ \text{head}(0) \rightarrow \text{err} \]
\[ \text{head}(s(x)) \rightarrow \text{err} \]

with \( F = \{ \text{cons} : 2, \text{incr} : 1, \text{adx} : 1, \text{head} : 1, \text{tail} : 1, s : 1, \text{nats} : 0, \text{zeros} : 0, 0 : 0, \text{nil} : 0, \text{err} : 0 \} \) and \( C = \{ 0, s, \text{cons}, \text{nil}, \text{err} \} \). Applying the rules on \text{incr}(x_1), we get:
In this paper, we have studied the problem of $C$-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 $C$-reducibility, termination of a program becomes less important, provided one is able to reach a $C$-form, i.e., to compute a result, which gives a criterion to stop the computations.

Our proof method establishes $C$-reducibility, and then sufficient completeness, by explicit induction on the $C$-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 $C$-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.

## 8. Conclusion and perspectives

In this paper, we have studied the problem of $C$-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 $C$-reducibility, termina-

### References


[8] H. Comon and F. Jacquemard. Ground reducibility is


Proof of Theorem 2

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

Proof. Let $\text{quot}(x_1, \ldots, X_m)$, $f \in D$, and $t = f(t_1, \ldots, t_m)$ a ground instance of $\text{quot}(x_1, \ldots, X_m)$. The proof is made by structural induction on $t$.

- Either $t$ is a constant $a$, and is ground reducible by hypothesis,
- or $t$ is not a constant, and:
  - either there exists a subterm $t_i$ of $t$ of the form $g(u_1, \ldots, u_p)$ with $g \in D$. By induction hypothesis, $t_i$ is reducible, and then $t$ is,
  - or every subterm $t_i$ of $t$ is of the form $g(u_1, \ldots, u_p)$, with $g \in C$, and we conclude thanks to the hypothesis.

The converse property is obvious.

Proof of Theorem 2

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

Proof.

We prove by induction on $T(F)$ that every ground instance $\theta \text{quot}(x_1, \ldots, x_m)$ of any term $\text{quot}(x_1, \ldots, x_m) \in T(F, X)$ is C-reducible. The induction ordering is constrained along the proof. At the beginning, it has at least to be noetherian and to have the constructor subterm property. Such an ordering always exists on $T(F)$ (for instance the embedding relation). Let us denote it $\succ$.

If $f$ is a constructor, by constructor subterm property of $\succ$, we have $\theta \text{quot}(x_1, \ldots, x_m) = \theta \text{quot}(x_1, \ldots, x_m)$, $\theta \text{quot}(x_1, \ldots, x_m) \succ \text{quot}(x_1, \ldots, \theta x_m)$. Then, by induction hypothesis, we suppose that $\theta x_1, \ldots, \theta x_m$ is C-reducible. Let $\theta x_1 \in C, \ldots, \theta x_m \in C$ be respectively any of their $C$-forms. It remains to prove that $\theta \text{quot}(x_1, \ldots, x_m)$ is $C$-reducible. This is the case since it is a $C$-term.

If $f$ is not a constructor, let us denote it $g$ and prove that $\theta \text{quot}(x_1, \ldots, x_m)$ is C-reducible for any $\theta$ satisfying $(A_0 = T, C_0 = T)$, if application of the inference rules with the strategy $S$ on $((\{g(x_1, \ldots, x_m)\}, T, T))$, terminates on a weakly successful proof tree. Let us denote $g(x_1, \ldots, x_m)$ by $t_{ref}$ in the sequel of the proof.

To each step of the procedure characterized by a state $s = (\{t\}, A, C)$, we associate the set of ground terms $G = \{\alpha \mid \alpha$ satisfies $A\}$, that is the set of ground instances represented by $s$. Inference rules Abstract and Narrow transform $((\{t\}, A, C))$ into $((\{t'\}, A', C'))$ to which is associated $G' = \{\beta' \mid \beta$ satisfies $A'\}$. We then prove the following result: for each $\alpha$ such that $\alpha \in G$, there exists a $\beta$ such that $\beta'$ is in $G'$ and such that C-reducibility of $\beta'$ implies C-reducibility of $\alpha$.

- Either Abstract is applied, so the current term $t$ becomes $t' = t[x_j]_{j \in \{i_1, \ldots, i_p\}}$, where $i_1, \ldots, i_p$ are the abstraction positions.

According to the condition of application of Abstract at positions $j \in \{i_1, \ldots, i_p\}$, each term $t_j$ such that:

- either $C - \text{RED}(t_j)$ is true, and then by definition of the predicate $C - \text{RED}$, $\alpha t|_j$ is C-reducible;
- or $C \land t_{ref} > t_j$ is satisfiable and then, by induction hypothesis, $\alpha t|_j$ is C-reducible.

Then let us define $\beta = \alpha \cup \bigcup_{j \in \{i_1, \ldots, i_p\}} X_j = \alpha t|_j \in C$.

Clearly $\beta$ satisfies $A'$ provided the $X_j$ are not in $\text{Dom}(\alpha)$, which is the case since they are fresh variables, and $\beta t' = \alpha(t_j|_j \in C \cup \{i_1, \ldots, i_p\}$, so $\alpha \rightarrow \beta t'$. Therefore C-reducibility of $\beta t'$, for any possible $C$-form $\alpha t|_j \in C$ of $\alpha$, implies C-reducibility of $\alpha t$.

- Or Narrow is applied on $((\{f(u_1, \ldots, u_m)\}, A, C)$, with a t-covering set $S$ of narrowing substitutions. If the strategy $S$ is optimized as explained in Section 5, then $S$ is reduced to $S_0$, which is also t-covering.

For any $\gamma$ satisfying $A$, as $S$ is t-covering, and by Proposition 2, there exists a rewriting step $\alpha \rightarrow \beta t'$ such that $t \rightarrow \beta t'$, and we conclude thanks to the hypothesis.

We then proceed by crossed proof with the following lemma.

We proceed by crossed proof with the following lemma.

Proof of Theorem 3

We proceed by crossed proof with the following lemma.

**Lemma 1.** Let $s = f(s_1, \ldots, s_m) \in T(F)$, and $t \rightarrow t'$ be any step of a transformation chain of $s$ with respect to $ST$ until $C$-form $s_f$. If this step is determined relatively to a step $u \rightarrow u'$ of the strategy tree $ST_f$, then $t$ is a ground instance of $u$. 
Proof. Let \( s = f(s_1, \ldots, s_m) \in T(\mathcal{F}) \) be a term and \( t \mapsto t' \) any step of a transformation chain of \( s \) with respect to \( ST \) until \( \text{C-form}_{ST}(s) \). The proof is made by induction on the length \( l \) of the transformation chain of \( s \) with respect to \( ST \).

Let \( l = 1 \).

- if \( f \in C \), by Definition 13, there is nothing to prove since \( ST_f \) is not used for normalizing the whole term \( s \).
- if \( f = g \in D \), then by definition of the strategy \( S \), the first step of each proof tree of \( ST \) is Abstract if \( f \) is not a constant, or Narrow if \( f \) is a constant. Indeed, as the narrowing step is \( s \)-covering, \( a \) narrows. As \( l = 1 \), this should also be the last step of the branch in the proof tree, which is impossible, since \( \) by definition of \( S \), the last step is Stop.

Let \( l = n \geq 1 \).

Let \( s' = g(s'_1, \ldots, s'_n) \) be the term obtained from \( s \) with a transformation chain of length \( n \) with respect to \( ST \). Let the property be true for \( l = n \), i.e. for every step \( t \mapsto t' \) of the transformation chain of \( s \) with respect to \( ST \) until \( s' \), determined relatively to a step \( u \mapsto u' \) of the strategy tree \( ST_f \), then \( u \) is a ground instance of \( u' \).

Let \( s \mapsto s' \) be the \( n+1 \)-th step of the given transformation chain of \( s \) with respect to \( ST \), determined relatively to a step \( u \mapsto u' \) of the strategy tree \( ST_f \). We prove that \( s \) is a ground instance of \( u' \).

Let \( s \mapsto s' \) be the \( n \)-th step of the given transformation chain of \( s \) with respect to \( ST \), determined relatively to a step \( u \mapsto u' \) of the strategy tree \( ST_f \). By induction hypothesis, \( s' \) is a ground instance of \( u' \), so there exists a ground substitution \( \theta \) such that \( \theta u = s' \).

- if \( u \mapsto u' \) is a step Abstract, then \( u' \) is of the form \( u_0[X_{i_1}]_{i_1 \in \{i_1, \ldots, i_p\}} \) where \( i_1, \ldots, i_p \) are the abstraction positions of \( u_0 \) and the \( X_{i_1} \) are new abstraction variables.

By Definition 13, \( s \mapsto s' \) where the \( s_j \) are \( s|_{\mathcal{C}} \) if we have \( C-RED(u_0) \), and \( \text{C-form}_{ST}(s_{i_j}) \) otherwise:

- In the first case, \( s_{i_j}|_{\mathcal{C}} \) is a ground instance of \( X_{i_j} \); let us note that \( s|_{\mathcal{C}} \) exists since by induction hypothesis \( s_{i_j}|_{\mathcal{C}} \) is a ground instance of \( u'|_{i_j} \) and we have \( C-RED(u'|_{i_j}) \).
- In the second case, by Theorem 3 on each \( s|_{\mathcal{C}} \) strict subterm of \( s \), we have \( \text{C-form}_{ST}(s|_{\mathcal{C}}) \) in \( C \)-form, hence a ground instance of \( X_{i_j} \).

Therefore \( s \) is a ground instance of \( u' \).

- if \( u \mapsto u' \) is a step Narrow, by Definition 13, we have \( s_0 = u_0 \mapsto_{p,l \mapsto \sigma} s' = \mu \sigma \cdot u' \), for some ground substitution \( \mu \), where the used branch of the step Narrow (always existing since the set of narrowing substitutions of Narrow is \( u \)-covering) is \( u_0 \mapsto_{p,l \mapsto \sigma} u' \), with \( \theta = \mu \sigma \). So \( s \) is a ground instance of \( u' \).

- if \( u \mapsto u' \) is a step Stop, then \( u' \) does not exist, since the state in \( ST_f \) after this step is a state of the form \( (\theta, A, C) \). Then there is nothing to prove.

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

Proof. Let \( s = f(s_1, \ldots, s_m) \in T(\mathcal{F}) \) be a term to be reduced with respect to \( ST \). We prove if a transformation chain of \( s \) with respect to \( ST \) gives a term \( t \), \( t \) is also a reduced form of \( s \) for \( \mathcal{R} \). We also show that the last term of any transformation chain of \( s \) with respect to \( ST \) is a \( C \)-term. This term eventually exists, since the process always stops, for \( ST \) is a set of successful trees, whose branches only have a finite number of states.

Let us show that \( t \) is a reduced form of \( s \) for \( \mathcal{R} \), by induction on the length \( l \) of the transformation chain of \( s \) with respect to \( ST \). For that, we use an induction on \( T(\mathcal{F}) \) and the property: \( \text{C-form}_{ST}(s) \) is a \( C \)-reduced form of \( s \) for \( \mathcal{R} \), with an ordering \( \succ \) satisfying all constraints generated by \( S \) in proving that \( \mathcal{R} \) is \( C \)-reducing. Such an ordering exists since, by hypothesis, \( \mathcal{R} \) has been proved \( C \)-reducing using Theorem 2.

Let \( l = 1 \).

- if \( f \in C \), by Definition 13, \( \text{C-form}_{ST}(f(s_1, \ldots, s_m)) = f(\text{C-form}_{ST}(s_1), \ldots, \text{C-form}_{ST}(s_n)) \). With the second induction, we have \( s = f(s_1, \ldots, s_m) \succ s_1, \ldots, s_m \), and then, by induction hypothesis, \( \text{C-form}_{ST}(s_1), \ldots, \text{C-form}_{ST}(s_n) \) are respectively \( C \)-reduced forms of \( s_1, \ldots, s_m \) for \( \mathcal{R} \). Then \( f(\text{C-form}_{ST}(s_1), \ldots, \text{C-form}_{ST}(s_n)) \) is a reduced form of \( s \) for \( \mathcal{R} \), and as \( f \) is a constructor symbol, is a \( C \)-term.

- if \( f = g \in D \), then by definition of the strategy \( S \), the first step of each proof tree of \( ST \) is Abstract, or Narrow if \( f \) is a constant. As \( l = 1 \), this should also be the last step of the branch in the proof tree, which is impossible since, by definition of \( S \), the last step of a branch is Stop.

Let \( l = n \geq 1 \).

Let \( t \) be the term obtained with a transformation chain of length \( n \) from \( s \) with respect to \( ST \). Let the property be true for \( l = n \), i.e. \( t \) is a reduced form of \( s \) with \( \mathcal{R} \). We prove it for \( l = n + 1 \).

- If the current step in the proof tree of \( f \) is Abstract, by Definition 13, \( t \mapsto t' = \theta(t'_1)_{i_1 \in \{i_1, \ldots, i_p\}} \), for some \( u \) such that \( t \) is a ground instance of \( u \), where \( i_1, \ldots, i_p \) are the abstraction positions and the \( t_j \) are \( t|_{\mathcal{C}} \) if we have \( C-RED(u_j) \), and \( \text{C-form}_{ST}(t_j) \) otherwise.

For the \( j \) such that \( C-RED(u_j) \), the \( t_j \) are \( C \)-reduced forms of the \( t_j \), by definition, and then reduced forms of the \( t_j \).

For the \( j \) such that we do not have \( C-RED(u_j) \), we have \( u_{ref} \succ u_j \), where \( u_{ref} \) is the initial term of the proof tree, in which \( u \) appears. So \( t = \theta u_{ref} \succ t_j \), for some ground substitution \( \theta \), since by Lemma 1, \( t_j \) is a ground instance of \( u_j \). Then the induction hypothesis of the second induction applies to \( t_j \), so \( \text{C-form}_{ST}(t_j) \) is a \( C \)-reduced form of \( t_j \), and then a reduced form of \( t_j \).

Then the term \( t' = \theta(t'_1)_{i_1 \in \{i_1, \ldots, i_p\}} \) is a reduced form of \( t \) for \( \mathcal{R} \).

Since, by induction hypothesis (with the first induction), \( t \) is a reduced form of \( s \) for \( \mathcal{R} \), we conclude that \( t' \) is a reduced form of \( s \) for \( \mathcal{R} \).

- If the current step in the proof tree of \( f \) is Narrow, by Definition 13, \( t \mapsto t' = \theta \cdot t' \), for some ground substitution \( \mu \), where the used branch of the step Narrow (always existing since Narrow is such
that the set of narrowing substitutions is \( u \)-covering), is
\[ u \rightarrow^{p,l \rightarrow \tau,\sigma} u', \]
with \( \theta = \mu \sigma \).

Then \( \mu t' \) is a reduced form of \( t \). Thus, reasoning as in the previous case, as \( t \) is a reduced form of \( s \) with \( R \), we conclude that \( t' \) is a reduced form of \( s \) with \( R \).

- If the current step in the proof tree of \( f \) is \textbf{Stop}, then by Definition 13, \( t \rightarrow t' \) where \( t' = t'_{l_c} \) if we have \( CRED(u) \), and \( t' = Cform_{ST}(t) \) otherwise.

We prove in a similar manner than in the \textbf{Abstract} case for the \( t|_j \) that \( t' \) is a \( C \)-reduced form of \( t \). Therefore \( t' \) is a \( C \)-term.

This last step is the only step where the process with respect to \( ST \) stops, in the case where \( l = n > 1 \). As shown above, this step stops with a \( C \)-form, which ends the proof.