Proving Weak Properties of Rewriting

Isabelle Gnaedig, Hélène Kirchner

Proving Weak Properties of Rewriting

Isabelle GNAEDIG[a,*], Hélène KIRCHNER[b]

[a] *INRIA & LORIA (UMR 7503 CNRS-INPL-INRIA-Nancy 2-UHP), Campus Scientifique, BP 239, F-54506 Vandoeuvre-lès-Nancy Cedex*
[b] *INRIA, Centre de Recherche INRIA Bordeaux - Sud-Ouest, Bâtiment A29, 351, Cours de la Libération F-33405 Talence*

Abstract

In rule-based programming, properties of programs, such as termination, are in general considered in their strong acceptance i.e., on every computation branch. But in practice, they may hold in their weak acceptance only i.e., on at least one computation branch. Moreover, weak properties are often enough to ensure that programs give the expected result. There are very few results to handle weak properties of rewriting. We address here two of them: termination and reducibility to a constructor form, in a unified framework allowing us to prove them inductively. Proof trees are developed, which simulate rewriting trees by narrowing and abstracting subterms. Our technique is constructive in the sense that proof trees can be used to infer an evaluation strategy for any given input: the right computation branch is developed without using a costly breadth-first strategy nor backtracking.

Keywords: abstraction, completeness, induction, rewriting, narrowing, weak termination

1. Introducing the Problem

Rewriting is now widely used for specifying, programming and prototyping. For rule-based programs, written in languages like ASF+SDF [1], Maude [2], Cafe-OBJ [3], ELAN [4], or TOM [5], evaluation consists in exploring rewriting derivations of an input term. In this context, it is very important to be able to prove properties on the rewriting relation, either to ensure that computations always give a result, or to prove that the provided result is as expected. In this paper, we consider two key properties: termination, ensuring that computation branches are finite, and sufficient completeness, expressing that every input can reduce to a completely computed form.

For the strong form of these properties -every computation branch is finite, every computation leads to a completely computed form- we now have many efficient proof techniques. Unfortunately, in the every day life of the programmer,

*Corresponding author : Tel. : + 33 3 54 95 84 21 - Fax : + 33 3 54 95 84 01 Email: Isabelle.Gnaedig@loria.fr

these strong properties are not always verified. In addition, the indeterminism of programming languages is often reduced by the implementation choices. Thus, in many cases, it is sufficient to consider the above properties in their weak form i.e., for termination to consider that at least one of the computation branches is finite, and for completeness that at least one of the computation branches leads to a completely computed form. Such weak properties have not been much investigated until now, perhaps because most of the current proof techniques rely on syntactic or semantic criteria of rewrite rules, and thus cannot capture the selection of computation branches.

Our Previous Work

Since several years, we have been developing an inductive approach to prove properties of rewriting, based on an explicit induction on the property to be proved. The main principle is to simulate the rewriting trees from any term by alternatively abstracting and narrowing patterns, constructed according to the induction hypothesis on the terms encountered. Abstracting a subterm in a term then consists in replacing it by a variable representing a canonical form for the property to be proved. Narrowing represents all possibilities of rewriting instances of the abstracted terms. Because it observes the rewriting trees, this approach allows us to handle proofs of rewriting properties in a finer way than when only considering the rewrite rules defining the rewriting relation. This is the reason why it is well-suited to handle strategies that restrict the set of relevant rewriting derivations. Similarly, weak properties that only hold for a subset of derivations can also be observed and proved more easily. From the proof of weak termination or weak completeness of a given rewrite system, this approach even allows us extracting a rewrite strategy computing a canonical form for any given ground term: a normal form in the case of termination, a constructor form in the case of reducibility to a constructor term.

We first have given procedures to prove termination of rewriting under the innermost strategy [6], local strategies on operators [7], and the outermost strategy [8]. In [9, 10], we have proposed a general mechanism factorizing the three previous procedures, and showed how it can be instantiated to provide a new procedure for each of the three strategies, more simple than the original ones.

We then have adapted our inductive mechanism to the proof of weak termination [11] and of reducibility to a constructor form [12]. Going one step further in our generalizing work, in [13] we have extended our proof framework to properties expressed by propositions involving a reduction relation and specific elements characterized by a decidable property. In this paper, we propose to instantiate this framework for handling the weak properties of termination and reducibility to a constructor form of a rewriting relation. We thus present in a unified procedure the two initial results of [11] and [12], develop their proofs and give further examples.

Weak Termination

Weak termination is an interesting property for languages like ELAN, whose strategies can express that the result of the program evaluation on a given input

is *one of its possible* finite evaluations, or *the first* one. Weak termination then warrants a result for such evaluation strategies.

As said above, analyzing weak termination with our approach also allows choosing a terminating evaluation process. Indeed, if the program is strongly terminating, a depth-first evaluation can be used while a breadth-first algorithm, often much more costly, is necessary in general if the program is only weakly terminating. In the second case, if there is a way to find terminating branches, the breadth-first technique can be avoided, which yields a considerable gain for program executions. This is what we propose.

For termination, we focus on the innermost rewriting strategy, consisting of always rewriting at the lowest possible positions. In fact, our technique is naturally adapted to the innermost case when abstraction is normalizing. This is the case when we handle the termination property since abstracting a given subterm in a term comes down to normalize it. Moreover, the innermost strategy is often used as a built-in mechanism in the evaluation of rule-based languages and functional languages.

Like the previously cited works dealing with termination under specific strategies, the approach presented here gives a way to prove weak termination of standard rewriting, which consists of rewriting without any strategy.

But to our knowledge, it is the only approach able to handle rewrite systems which are not strongly but only weakly innermost terminating. This is the case for the following rewrite system:

$$\begin{array}{ll} f(g(x), s(0)) \to f(g(x), g(x)) & (1) \\ f(g(x), s(y)) \to f(h(x, y), s(0)) & (2) \\ g(s(x)) \to s(g(x)) & (3) \\ g(0) \to 0 & (4) \\ h(x, y) \to g(x). & (5) \end{array}$$

Obviously, $\mathcal{R}$ is not terminating, nor even, because of Rule (2), innermost terminating. For instance, the following innermost infinite sequence is possible in $\mathcal{R}$: $\underline{f(g(f(0,0)), s(0))} \to^{(2)} f(\underline{h(f(0,0),0)}, s(0)) \to^{(5)} \underline{f(g(f(0,0)), s(0))} \ldots$ However, $\mathcal{R}$ is weakly innermost terminating ; in particular, the cycle above can be avoided by using Rule (1) instead of Rule (2).

The weak termination property has been studied from several perspectives. For instance, B. Gramlich proved that weak termination can imply strong termination [14]. He also established conditions on rewrite systems for the property to be preserved by the union operation [15]. J. Goubault-Larrecq proposed a proof of weak termination of typed Lambda-Sigma calculi in [16]. Directly using the termination notion on terms has also been proposed in [17], for inductively proving well-foundedness of binary relations, among which path orderings. The approach differs from ours in that it works on general relations, that can then be used on term rewriting systems, whereas we directly handle property proofs of a given rewrite system. To some extent, for weak termination, our method has similarities with [18], where an automaton is built for normalization according to a needed redex strategy in the case of orthogonal rewrite systems.

Sufficient Completeness and Related Properties

Sufficient completeness also plays an important role in algebraic specifications, as well as in rewriting-based programs. In both contexts, terms are built on operators, among which we can distinguish constructors. Constructors are basic symbols, allowing us to describe expected results of computations. The other symbols, called defined symbols, represent functions defined on these values.

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 a constructor term or constructor form. It allows inductive proofs, in particular by consistency methods [19]. 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 at least one completely computed form for every input.

Proving sufficient completeness is undecidable in general. It has already been widely studied, for example in [20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30], 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 is terminating, confluent, and the normal form of a constructor term is again a constructor term [25]. Under these conditions, techniques developed for proving ground reducibility hold for sufficient completeness as well [31, 32, 25, 33, 34, 35].

As said before, we address here the problem of sufficient completeness from the programming point of view, and we go beyond the previous usual restrictions. We consider the case where a program or a rewrite system can be neither confluent, nor terminating, and we study its evaluations. We do not assume any other usual restriction such 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 input gives a completely evaluated result; in other words, whether for every ground term, there is 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.

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 to imply sufficient completeness, thus weakening the condition of the well-known theorem of [25]. We also note that, as our approach requires a covering property stronger than ground reducibility, this property directly implies sufficient completeness

if we suppose weak termination. Thus, with respect to sufficient completeness, $\mathcal{C}$-reducibility and the proof technique we propose are interesting:

- obviously when the rewrite system is not even weakly terminating,
- when it is terminating, or just weakly terminating, but there is no technique to prove it. The weak termination proof is difficult to handle in general, and to our knowledge, the proof procedure given in [11] is the only one today for first-order rewriting. However it fails for instance on the following small example, 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:

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

  With our technique, we show that C-reducibility can be proved even in this case, where weak termination cannot be proved.

- as an alternative to the weak termination proof when the constructor forms are irreducible, for instance 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 this case, C-reducibility implies weak termination.

Note finally that, unlike for termination, for C-reducibility, our technique is not developed with the innermost strategy. Indeed, for termination, alternating abstraction and narrowing provides a simulation of innermost rewriting. For C-reducibility however, abstraction of a subterm represents one of its constructor forms, which may still be reducible, and then applying abstraction and narrowing simulates standard rewriting.

Summary of the Paper

In Section 2, the background is presented. In Section 3, links between termination, sufficient completeness, ground reducibility and $\mathcal{C}$-reducibility are investigated. Our general inductive proof mechanism is presented in Section 4

and technical concepts are developed in Section 5. Section 6 presents the proof procedure, common to weak termination and $\mathcal{C}$-reducibility. In Sections 7 and 8, the proof procedure is applied on examples for weak innermost termination and $\mathcal{C}$-reducibility respectively. Section 9 details the constructive technique allowing us to extract of the proof a normal form in the case of termination, and a $\mathcal{C}$-form in the case of $\mathcal{C}$-reducibility.

2. The Background

We assume the reader familiar with the basic definitions and notations of algebras and term rewriting given for instance in [36, 37, 38, 39].

Abstract Reduction Systems

An *abstract reduction system* $(\mathcal{M}, \rightarrow)$ is given by a set $\mathcal{M}$ and a reduction relation $\rightarrow \subseteq \mathcal{M} \times \mathcal{M}$. A *derivation* is a chain of elements $a_1 \rightarrow a_2 \rightarrow \dots a_n$; a_1 is called the *source* of the derivation. The element a is *reducible* iff there is b such that $a \rightarrow b$ and *irreducible* otherwise.

Terms, Substitutions, Instantiations

$\mathcal{T}(\mathcal{F}, \mathcal{X})$ is the set of terms built from a finite set $\mathcal{F}$ of function symbols f with arity $n \in \mathbb{N}$ (which is denoted $ar(f) = n$), and a set $\mathcal{X}$ of variables denoted $x, y \dots$. $\mathcal{V}ar(t)$ is the set of variables of the term t. $\mathcal{T}(\mathcal{F})$ is the set of ground terms (without variables). The set $\mathcal{F}$ of symbols is split into a set of constructors $\mathcal{C}$ and a set of defined symbols $\mathcal{D}$.

The terms of $\mathcal{T}(\mathcal{C})$ are called *constructor terms*, or more briefly *$\mathcal{C}$-terms* (or *$\mathcal{C}$-forms*). 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. Let p and p' be two positions. The position p' is a (strict) *suffix* of p if $p' = p\lambda$, where λ is a (non-empty) sequence of integers. The notation $t|_p$ stands for the subterm of t at position p. If p is a position in t, then $t[t']_p$ denotes the term obtained from t by replacing the subterm at position p by the term t'.

A *substitution* is an assignment from $\mathcal{X}$ to $\mathcal{T}(\mathcal{F}, \mathcal{X})$, written $\sigma = (x \mapsto t) \dots (y \mapsto u)$. It uniquely extends to an endomorphism of $\mathcal{T}(\mathcal{F}, \mathcal{X})$. 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 defined by $Ran(\sigma) = \bigcup_{x \in Dom(\sigma)} Var(\sigma x)$. Id denotes the identity substitution. The composition of substitutions σ_1 followed by σ_2 is denoted $\sigma_2 \circ \sigma_1$ or simply $\sigma_2\sigma_1$. An *instantiation* is an assignment θ from $\mathcal{X}$ to $\mathcal{T}(\mathcal{F})$, extending to an application from $\mathcal{T}(\mathcal{F}, \mathcal{X})$ to $\mathcal{T}(\mathcal{F})$. The set of instantiations is denoted by Θ. A term $\theta t \in \mathcal{T}(\mathcal{F})$, for $\theta \in \Theta$ and $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, is called instantiation of t or ground instance of t.

Given two substitutions σ_1, σ_2 and a set of variables $\mathcal{X}$, we write $\sigma_1 = \sigma_2[\mathcal{X}]$ if $\forall x \in Dom(\sigma_1) \cap Dom(\sigma_2) \cap \mathcal{X}, \sigma_1 x = \sigma_2 x$.

Given a variable set $\mathcal{X}_1$, we write $\sigma_{\mathcal{X}_1}$ for the *restriction* of σ to the variables of $\mathcal{X}_1$ i.e., the substitution such that $Dom(\sigma_{\mathcal{X}_1}) = Dom(\sigma) \cap \mathcal{X}_1$ and $\forall x \in Dom(\sigma_{\mathcal{X}_1}) : \sigma_{\mathcal{X}_1} x = \sigma x$.

Orderings

An ordering $\succ$ on $\mathcal{T}(\mathcal{F},\mathcal{X})$ is *noetherian* iff there is no infinite decreasing chain for this ordering. It is *monotone* iff for any pair of terms t, t' of $\mathcal{T}(\mathcal{F},\mathcal{X})$, for any context $f(\ldots\ \ldots)$, $t \succ t'$ implies $f(\ldots t \ldots) \succ f(\ldots t' \ldots)$. It has the *subterm property* iff for any t of $\mathcal{T}(\mathcal{F},\mathcal{X})$, $f(\ldots t \ldots) \succ t$. It is *stable by substitution* iff for every substitution σ, $t \succ t'$ implies $\sigma t \succ \sigma t'$. For $\mathcal{F}$ and $\mathcal{X}$ finite, if $\succ$ is monotone and has the subterm property, then it is noetherian [40]. If, in addition, $\succ$ is stable by substitution, then it is called a *simplification ordering*. A *precedence* is an ordering on $\mathcal{F}$, denoted $>_{\mathcal{F}}$.

Rewriting

A set $\mathcal{R}$ of *rewrite rules* or *rewrite system* is a set of pairs of terms of $\mathcal{T}(\mathcal{F},\mathcal{X})$, denoted $l \to r$, such that $l \notin \mathcal{X}$ and $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$. In this paper, we only consider finite sets of rewrite rules.

The rewriting relation induced by $\mathcal{R}$ is denoted by $\to^{\mathcal{R}}$ ($\to$ if there is no ambiguity on $\mathcal{R}$), and defined by $s \to t$ iff there is a substitution σ and a position p in s such that $s|_p = \sigma l$ for some rule $l \to r$ of $\mathcal{R}$, and $t = s[\sigma r]_p$. This is written $s \to^{\mathcal{R}}_{p,l\to r,\sigma} t$ where either p, $l \to r$, σ or $\mathcal{R}$ may be omitted; $s|_p$ is called a *redex*. The reflexive transitive closure of the rewriting relation induced by $\mathcal{R}$ is denoted by $\overset{*}{\to}^{\mathcal{R}}$.

The notion of *constructor* is defined in different ways, depending on the property to be proved. For $\mathcal{C}$-reducibility, the set $\mathcal{C}$ of constructors is a given subset of $\mathcal{F}$. For weak termination, a constructor is a symbol of $\mathcal{F}$ that is not a top symbol of a left-hand side of a rule.

The *innermost* rewriting strategy consists of always reducing at the lowest possible positions. The *innermost* rewriting relation is denoted $\to^{Inn}$.

Given a term t, we call (innermost) *normal form* of t, and we denote it $t{\downarrow}$, any irreducible term u, if it exists, such that $t \overset{*}{\to}^{(Inn)} u$. If a term t rewrites to a $\mathcal{C}$-form, we write this $\mathcal{C}$-form $t{\downarrow_c}$, and say that it is a $\mathcal{C}$-reduced form for $\mathcal{R}$. Note that given t, its (innermost) normal form or its $\mathcal{C}$-form may be not unique.

Narrowing

Let $\mathcal{R}$ be a rewrite system on $\mathcal{T}(\mathcal{F},\mathcal{X})$. A term t is *narrowed* into t', at the non-variable position p, using the rewrite rule $l \to r$ of $\mathcal{R}$ and the substitution σ, when σ is a most general unifier of $t|_p$ and l, and $t' = \sigma(t[r]_p)$. This is denoted $t \rightsquigarrow^{\mathcal{R}}_{p,l\to r,\sigma} t'$ where either p, $l \to r$, σ or $\mathcal{R}$ may be omitted. It is always assumed that there is no variable in common between the rule and the term i.e., that $\mathcal{V}ar(l) \cap \mathcal{V}ar(t) = \emptyset$.

3. C-reducibility and Related Properties

Before comparing them, let us first formally define the properties presented in the introduction.

Definition 3.1. Let $\mathcal{R} = \{l_i \to r_i, i \in [1..m]\}$ be a rewrite system 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 denoted $\mathcal{E}$.

Definition 3.2. 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 $\mathcal{D}$) iff 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 3.3. Let $\mathcal{R}$ be a rewrite system on $\mathcal{T}(\mathcal{F}, \mathcal{X})$. A term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is *ground reducible* iff every ground instance of t is reducible with $\mathcal{R}$.

Given a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, a derivation of source t is called *full derivation* either if it is infinite or if its last term is irreducible.

Definition 3.4. Let $\mathcal{R}$ be a rewrite system on $\mathcal{T}(\mathcal{F}, \mathcal{X})$, where $\mathcal{F} = \mathcal{C} \cup \mathcal{D}$. A(n) (innermost) *$\mathcal{C}$-reducing derivation* of $t \in \mathcal{T}(\mathcal{F})$ is a(n) (innermost) derivation of source t containing a $\mathcal{C}$-term. A term $t \in \mathcal{T}(\mathcal{F})$ is (innermost) *$\mathcal{C}$-reducible* iff there is at least a(n) (innermost) $\mathcal{C}$-reducing derivation of source t. It is (innermost) *strongly $\mathcal{C}$-reducible* iff all full derivations of source t are (innermost) $\mathcal{C}$-reducing. The rewrite system $\mathcal{R}$ is said to be (innermost) (strongly) *$\mathcal{C}$-reducing* iff every term t of $\mathcal{T}(\mathcal{F})$ is (innermost) (strongly) $\mathcal{C}$-reducible.

A few remarks on these properties are useful. If $\mathcal{R}$ does not have rules whose left-hand sides are $\mathcal{C}$-terms, and more generally rules whose left-hand sides are in $\mathcal{T}(\mathcal{C}, \mathcal{X})$, the reached $\mathcal{C}$-forms are normal forms.

If there are such rules in $\mathcal{R}$, the $\mathcal{C}$-forms may be reducible. Then, either $\mathcal{R}$ is constructor-preserving (if a left-hand side of a rule of $\mathcal{R}$ is a $\mathcal{C}$-term, the corresponding right-hand side is a $\mathcal{C}$-term as well) 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 term stops on the first encountered constructor form, ensured to exist on at least one rewriting branch.

The diagram in Fig. 1 summarizes the links between the different properties. "Ground reducibility" stands for ground reducibility of all patterns $f(x_1, \ldots, x_m), f \in \mathcal{D}$. Arrows represent implications between properties; they are labeled by the necessary conditions on the rewrite system required by the implications. All links are quite obvious, except those between ground reducibility and sufficient completeness, given by Theorem 7 in [25].

From the diagram, we immediately set the following theorem, which weakens the terminating condition of the "if" part of Theorem 7 in [25].

Theorem 3.5. *Let $\mathcal{R}$ be an innermost terminating rewrite system 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.*

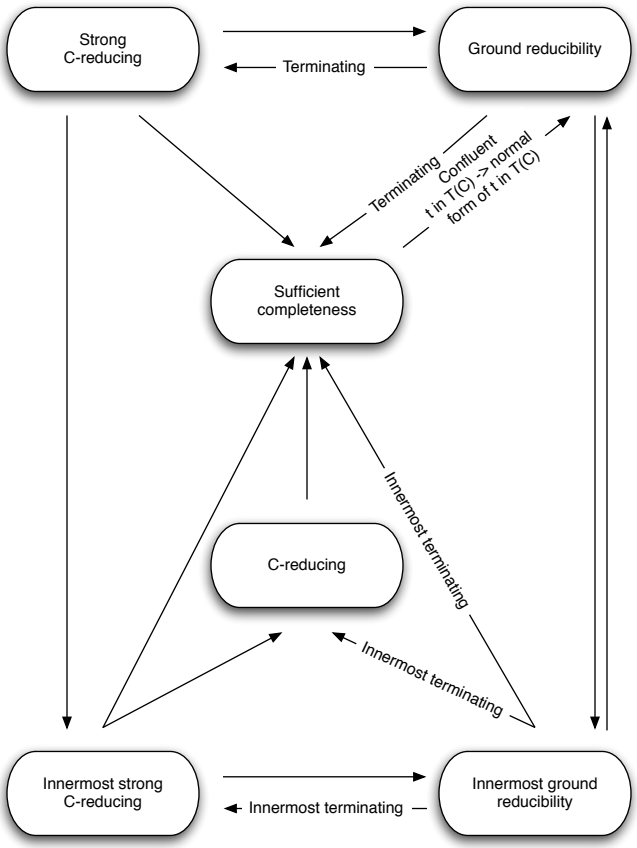

Figure 1: $\mathcal{C}$-reducing property and linked properties

This result may be verified in analyzing the proof of Theorem 7 in [25], 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, as illustrates the following terminating but not confluent example, where a, b, c are constants and c is the only constructor form:

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

The term b is equal, in the equational theory, to the constructor term c, but is not $\mathcal{C}$-reducible.

4. The Inductive Proof Process

From now on, we assume that $\mathcal{T}(\mathcal{F})$ is non empty and that there is a noetherian ordering $\succ$ defined on terms of $\mathcal{T}(\mathcal{F})$. For proving the proposition P for any element t of $\mathcal{T}(\mathcal{F})$, we proceed by induction on $\mathcal{T}(\mathcal{F})$ with the ordering $\succ$ as noetherian induction relation, assuming that for any t' such that $t \succ t'$, P holds for t'.

The concepts and different steps of the inductive proof process are illustrated in what follows on two examples already presented in the introduction.

Example 4.1. For the rewrite system:

$$\begin{array}{l} f(g(x), s(0)) \to f(g(x), g(x)) \\ f(g(x), s(y)) \to f(h(x, y), s(0)) \\ g(s(x)) \to s(g(x)) \\ g(0) \to 0 \\ h(x, y) \to g(x) \end{array}$$

the proposition P addressed here is : any ground term has an innermost normal form.

Example 4.2. For the following rewrite system, where 0 and 1 are constructors, and the set of $\mathcal{C}$-terms is $\{0, 1\}$:

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

the proposition P is : any ground term is $\mathcal{C}$-reducible.

4.1. P-Canonical Forms

The two properties addressed in this paper can be expressed by a proposition P involving the reduction relation $\to$ and specific elements of $\mathcal{T}(\mathcal{F})$ characterized by a decidable property: for termination proofs, this is the property of irreducibility w.r.t the reduction relation; for completeness proofs, this is the syntactic property to be built only with constructors. We distinguish these particular elements of $\mathcal{T}(\mathcal{F})$ by calling them *P-canonical elements*. When P in general is a strong proposition, it is stated on any given element t of $\mathcal{T}(\mathcal{F})$ as: on every derivation of source t, there is a P-canonical element. When P is a weak proposition, like the two properties considered in this paper, it is stated as: there is at least one derivation of source t having a P-canonical element. This leads to the definition of P-canonical form of an element.

Definition 4.3. [13] Let $\mathcal{R}$ be a rewrite system, P a proposition to be proved on $\mathcal{T}(\mathcal{F})$ and $T \subseteq \mathcal{T}(\mathcal{F})$ a decidable set of P-canonical elements. A *P-canonical form* $t{\downarrow_{\mathcal{P}}}$ of a term t is an element of T belonging to a derivation of source t. For P being weak termination, $t{\downarrow_{\mathcal{P}}} = t{\downarrow}$; for P being $\mathcal{C}$-reducibility, $t{\downarrow_{\mathcal{P}}} = t{\downarrow_{\mathcal{C}}}$.

4.2. Covering Patterns and Simulation

Our goal is to inductively prove the property P on $\mathcal{T}(\mathcal{F})$. For that, we simulate $(\mathcal{T}(\mathcal{F}), \rightarrow)$ with another abstract system $(\mathcal{T}(\mathcal{F}, \mathcal{X}), \not\rightarrow)$, by establishing a correspondence between the elements of $\mathcal{T}(\mathcal{F})$ and $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and between the reduction relations $\rightarrow$ and $\not\rightarrow$.

We first define a set of patterns of the form $f(x_1, \ldots, x_n)$ with $f \in \mathcal{F}$. We relate patterns to elements of $\mathcal{T}(\mathcal{F})$ by considering all possible instantiations $\theta f(x_1, \ldots, x_n)$. More generally, for a term u with variables, we denote by $\langle u \rangle$ the set $\{\theta u \mid \theta \in \Theta\}$ where Θ is the set of all instantiations of $\mathcal{T}(\mathcal{F}, \mathcal{X})$ into $\mathcal{T}(\mathcal{F})$ such that $\mathcal{V}ar(u) \subseteq Dom(\theta)$.

This definition extends to a set of terms $U = \{u_1, \ldots, u_k\}$ in the following way: $\langle U \rangle = \{\langle u_1 \rangle, \ldots, \langle u_k \rangle\}$.

Example 4.4. In Example 4.1, since $\mathcal{F}$ is $\{f, g, h, s, 0\}$, the set of patterns is $\{f(x_1, x_2), g(x_1), h(x_1, x_2), s(x_1), 0\}$.

Then the correspondence between the reduction relations $\rightarrow$ and $\not\rightarrow$ is expressed with a notion of simulation defined below. According to the property P to be proved, from a given term, only relevant reduction steps, called P-relevant reduction steps, have to be considered. For the two weak properties studied here, the P-relevant reduction steps from a given term are any innermost reduction step for weak innermost termination, and any reduction step for $\mathcal{C}$-reducibility.

Definition 4.5. [13] Let $(\mathcal{T}(\mathcal{F}), \rightarrow)$ and $(\mathcal{T}(\mathcal{F}, \mathcal{X}), \not\rightarrow)$ be two abstract reduction systems. $(\mathcal{T}(\mathcal{F}, \mathcal{X}), \not\rightarrow)$ is a *P-simulation* of $(\mathcal{T}(\mathcal{F}), \rightarrow)$ iff there is a relation $L \subseteq \mathcal{T}(\mathcal{F}, \mathcal{X}) \times \mathcal{T}(\mathcal{F})$, such that for every P-relevant reduction step $a_1 \rightarrow a_2$, with $a_1, a_2 \in \mathcal{T}(\mathcal{F})$, there is a corresponding reduction step $b_1 \not\rightarrow b_2$, with $b_1, b_2 \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, and $b_1 L a_1, b_2 L a_2$.

4.3. Lifting Rewriting Trees into Proof Trees

To ensure non-emptiness of $\mathcal{T}(\mathcal{F})$, we suppose that there is at least one P-canonical constant in $\mathcal{T}(\mathcal{F})$. We then observe the derivation tree of $\rightarrow$ starting from an element $t \in \mathcal{T}(\mathcal{F})$ which is any instance of a term $f(x_1, \ldots, x_m)$ for some function symbol $f \in \mathcal{F}$, and variables $x_1, \ldots, x_m$.

This derivation tree is simulated, with a lifting mechanism, by a proof tree developed from $f(x_1, \ldots, x_m)$ on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ by alternatively using two main operations, namely narrowing and abstraction, adapted to the property to be proved and to the considered reduction relation. Narrowing simulates the reduction possibilities of elements of $\mathcal{T}(\mathcal{F})$, according to the instances of the narrowed terms. The abstraction process simulates sequences of reductions steps in the derivations, which are valid under the induction hypothesis. More precisely,

it consists of replacing subterms by special variables, denoting any of their P-canonical forms. It is performed on subterms whose instances can be assumed to satisfy the proposition P by induction hypothesis.

The schematization of derivation trees is achieved through constraints. Each node of the developed proof trees is composed of a current term of $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and a constraint progressively built along the successive abstraction and narrowing steps, 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 actually characterize sets of elements of $\mathcal{T}(\mathcal{F})$. The latter may also be used for controlling the narrowing process, well known to easily diverge. A node schematizes the set of elements of $\mathcal{T}(\mathcal{F})$ given by the instantiations of the current term, which are solutions of the abstraction constraints.

4.4. The Overall Mechanism

Let us now consider a proof tree whose root is the pattern $f(x_1, \ldots, x_m)$. We see how the reduction relation on instances of $f(x_1, \ldots, x_m)$ can be schematized, with abstraction and narrowing applied on a current term t of the proof tree:

- first, some subterms t_j of the current term t of the proof tree are selected: if $\theta f(x_1, \ldots, x_m) \succ \theta t_j$ for the induction ordering $\succ$ and for every θ that is a solution of the constraint associated to t, we may suppose, by induction hypothesis, that the θt_j satisfy the proposition P. The t_j are then replaced in t by *abstraction variables* X_j representing respectively any of their P-canonical forms $t_j{\downarrow_{\mathcal{P}}}$. Reasoning by induction allows us to suppose the existence of the $t_j{\downarrow_{\mathcal{P}}}$ *without explicitly computing them*;

- second, narrowing the resulting term $u = t[X_j]_{j \in \{i_1, \ldots, i_p\}}$ (where $i_1, \ldots, i_p$ are the positions of the abstracted subterms t_j in t) into terms v, according to the possible instances of the X_j. In general, there are several possible narrowing steps from u. Among them, we consider a subset of narrowing steps simulating the relevant reductions of θu, where θ is a solution of the constraint associated to u.

Then the problem of proving P on the instantiations of t is reduced to the problem of proving P on the instantiations of v. If $\theta f(x_1, \ldots, x_m) \succ \theta v$ for every instantiation θ that is a solution of the constraint associated to v, by induction hypothesis, θv is supposed to satisfy P. Otherwise, the process is iterated on v, until we get a term t' such that either $\theta f(x_1, \ldots, x_m) \succ \theta t'$, or $\theta t'$ satisfies P.

The proof procedure given in this paper is described by deduction rules applied with a strategy later described in Section 6. Applying this procedure to the initial term $f(x_1, \ldots, x_m)$ builds a proof tree. Branching is produced by the different possible narrowing steps. The proposition P is established when the procedure terminates because the deduction rules do not apply anymore and all terminal nodes of all proof trees represent terms satisfying P.

5. Abstraction, Narrowing, and the Involved Constraints

Let us now formalize the concepts required for our technique.

5.1. Ordering Constraints

The induction ordering is constrained along the proof by imposing inequalities of the form $t > u$ between terms that must be comparable, each time the induction hypothesis is used in the abstraction mechanism. They are called *ordering constraints*.

Definition 5.1 (ordering constraint). *An ordering constraint* is a pair of terms of $\mathcal{T}(\mathcal{F}, \mathcal{X})$ denoted by $(t > t')$. It is *satisfiable* iff there is an ordering $\succ$, such that for every instantiation θ whose domain contains $\mathcal{V}ar(t) \cup \mathcal{V}ar(t')$, we have $\theta t \succ \theta t'$. Then we say that $\succ$ satisfies $(t > t')$. A conjunction C of ordering constraints is satisfiable iff there is an ordering satisfying all conjuncts. The empty conjunction, always satisfied, is denoted by $\top$.

An ordering constraint may be unsatisfiable, for example, when it contains antagonistic unequalities like, for example, $a > b$ and $b > a$. It may also be unsatisfiable, like $a > f(a)$, by an ordering enjoying, for instance, the subterm property, which is naturally required when one wants to work with a noetherian ordering.

As we are working with a lifting mechanism on the proof trees with terms of $\mathcal{T}(\mathcal{F}, \mathcal{X})$, we use an ordering $\succ_{\mathcal{X}}$ on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ such that $t \succ_{\mathcal{X}} u$ implies on $\mathcal{T}(\mathcal{F})$ that $\theta t \succ \theta u$, for every θ that is a solution of the constraint associated to u. Every ordering $\succ_{\mathcal{X}}$ on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ satisfying the above constraints and which is stable by instantiation fulfills the previous requirements on $\mathcal{T}(\mathcal{F})$. For convenience sake, the ordering $\succ_{\mathcal{X}}$ is also written $\succ$.

The satisfiability of a constraint conjunction C is undecidable, but a sufficient condition is to find an ordering $\succ$ on $\mathcal{T}(\mathcal{F}, \mathcal{X})$, stable by instantiation and such that $t \succ t'$ for any constraint $t > t'$ of C.

We often try to solve the constraints of C by finding simplification orderings. This is a well-known problem in rewriting. The easiest way to proceed is to test simple existing orderings like the subterm ordering, the Recursive Path Ordering (RPO), or the Lexicographic Path Ordering (LPO). This is often sufficient for the constraints considered here. Otherwise, automatic constraint solvers can provide adequate polynomial orderings. See [10] for experiments.

5.2. Abstraction

To abstract a term t at positions $j \in \{i_1, \ldots, i_p\}$, we assume that the $t|_j$ are such that every instantiation $\theta t|_j$ verifies the proposition P. It then reduces to a P-canonical form $\theta t|_j{\downarrow_{\mathcal{P}}}$, and we replace the $t|_j$ by abstraction variables X_j representing respectively any of their possible P-canonical forms. Let us define these special variables more formally.

Definition 5.2 (abstraction variable). Let $\mathcal{X}_A$ be a set of variables disjoint from $\mathcal{X}$. Symbols of $\mathcal{X}_A$ are called *abstraction variables*. Instantiations are extended to $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{X}_A)$ in the following way: for every instantiation θ, $\forall x \in Dom(\theta)$, if $x \in \mathcal{X}_A$, then θx is a P-canonical form.

Definition 5.3 (term abstraction). The term $t[t|_j]_{j\in\{i_1,\ldots,i_p\}}$ is *abstracted* into the term u (called *abstraction* of t) at positions $\{i_1, \ldots, i_p\}$ iff $u = t[X_j]_{j\in\{i_1,\ldots,i_p\}}$, where the $X_j, j \in \{i_1, \ldots, i_p\}$ are fresh distinct abstraction variables.

In fact, the proposition P is proved by reasoning on terms with abstraction variables i.e., on terms of $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{X}_A)$. Ordering constraints are extended to pairs of terms of $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{X}_A)$. When subterms $t|_j$ are abstracted by X_j, we put, in a set A, constraints on abstraction variables called *abstraction constraints*, to express that their instantiations can only be P-canonical forms of the corresponding instantiations of $t|_j$. Initially, they are of the form $t{\downarrow_{\mathcal{P}}} = X$ where $t \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{X}_A)$, and $X \in \mathcal{X}_A$, but we will see later how they are combined with the substitutions used for the narrowing process.

Example 5.4. In Example 4.1, a first abstraction applied on the pattern $f(x_1, x_2)$ leads to $f(X_1, X_2)$ with the abstraction constraint $A = (x_1{\downarrow} = X_1 \wedge x_2{\downarrow} = X_2)$. This step schematizes derivations that compute a normal form for each subterm of any ground instance of $f(x_1, x_2)$. This is captured by the abstraction constraint A.
To perform this step, we assume that the property P holds on the instances of x_1 and x_2, which is captured by the ordering constraint $C = (f(x_1, x_2) > x_1, x_2)$.

5.3. Narrowing

After abstracting the current term t into $t[X_j]_{j\in\{i_1,\ldots,i_p\}}$, we check whether the possible instantiations of $t[X_j]_{j\in\{i_1,\ldots,i_p\}}$ are reducible, according to the possible values assigned to the X_j. This is achieved by narrowing $t[X_j]_{j\in\{i_1,\ldots,i_p\}}$.

To simulate the reduction relation on $\mathcal{T}(\mathcal{F})$, a specific narrowing relation $\rightsquigarrow$ is chosen in such a way that $(\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{X}_A), \rightsquigarrow)$ is a P-simulation of $(\mathcal{T}(\mathcal{F}), \rightarrow)$.

5.3.1. The Case of Weak Termination of Innermost Rewriting

We need a narrowing relation simulating innermost rewriting on ground terms. We use a refinement of the usual definition of innermost narrowing, we have proposed in [6, 10] for proving strong termination of innermost rewriting. The underlying requirements are the following.

First, in the innermost case, to ensure P-simulation, an innermost narrowing redex in t must correspond to an innermost rewriting redex in a ground instance of t. This is the case only if, in the rewriting chain of the ground instance of t, there is no rewriting redex anymore in the part of the term brought by the instantiation. This condition is fulfilled because the variables of t are in $\mathcal{X}_A$. Indeed, t is issued from a pattern $f(x_1, \ldots, x_m)$, first abstracted into

$f(X_1, \ldots, X_m)$. The variables introduced by the next abstraction steps are abstraction variables. This is also the case for the variables introduced by the narrowing process, as we will see later.

Then, among the ground instances of t, there may be innermost rewriting positions p for some instances, and p' for other instances, such that p' is a suffix of p. So, when narrowing at some position p, the set of corresponding ground instances of t is defined by excluding the ground instances that would be narrowable at some suffix position p' of p.

Let us give an example. With the rewrite system $\{f(g(h(x))) \to a, h(a) \to a\}$, the ground instances of $f(x)$ rewrite with the first rule at the top position if the ground instances of x are of the form $g(h(x'))$, but x' has to be different from a. Indeed, if $x' = a$, the ground instance of $f(g(h(x)))$ rewrites with the second rule at a suffix position of the top, and the top position is not an innermost position anymore.

The narrowing steps of a given term t are thus computed in the following way. We first look at every non-variable position p of t such that $t|_p$ unifies with the left-hand side of a rule using a substitution σ. The position p is a narrowing position of t, iff there is no suffix position p' in t such that $\sigma t|_{p'}$ unifies with a left-hand side of rule. Then we look for every suffix position p' of p in t such that $\sigma t|_{p'}$ narrows with some substitution σ' and some rule $l' \to r'$, and we set a constraint to exclude these substitutions. So the substitutions used to narrow a term have in general to satisfy a set of disequalities coming from the complement of previous substitutions.

We now formalize this mechanism, starting by establishing a correspondence between narrowing and rewriting. For that, we characterize the set $RED_l(t)$ of instantiations β such that $\beta(t)$ is reducible at some position p by a rule $l \to r$.

In [41], we prove that $RED_l(t)$ is equal to $B^{t,\sigma} = \{\beta \in \Theta \mid Dom(\beta) = \mathcal{V}ar(t), \exists \mu \in \Theta, \beta = \mu\sigma[\mathcal{V}ar(t)]\}$ where σ is the most general unifier of $t|_p$ and l.

In the following, we identify a substitution $\sigma = (x_1 \mapsto t_1) \ldots (x_n \mapsto t_n)$ on $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{X}_A)$ with the finite conjunction of solved equations $(x_1 = t_1) \wedge \ldots \wedge (x_n = t_n)$, where $x_i \notin \mathcal{V}ar(t_i)$ for $i = 1, \ldots, n$. So, we have $\sigma(x_i) = t_i$. On another hand, since we can choose most general unifiers such that $Dom(\sigma) \cap Ran(\sigma) = \emptyset$, we have $\sigma(t_i) = t_i$ for $i = 1, \ldots, n$. Thus σ is a solution of $(x_1 = t_1) \wedge \ldots \wedge (x_n = t_n)$. Any ground instance of σ is also a ground solution.

So the previous set $B^{t,\sigma}$ is represented using the set $\Phi^{t,\sigma}$ of ground solutions of the equational formula $\bigwedge_i (x_i = t_i)$, which is the set of instantiations $\{\zeta \in \Theta \mid Dom(\zeta) = \mathcal{V}ar(t) \cup Ran(\sigma), \forall i, \zeta(x_i) = \zeta(t_i), x_i \in \mathcal{V}ar(t), t_i \in \mathcal{T}(\mathcal{F}, \mathcal{X})\}$.

In order to consider instead the ground instances of t which are not reducible at position p by the rule $l \to r$, we consider the *complement* formula $\bigvee_i (x_i \neq t_i)$, shortly denoted by $\overline{\sigma}$, whose set of ground solutions is $\overline{\Phi^{t,\sigma}} = \{\zeta \in \Theta \mid Dom(\zeta) = \mathcal{V}ar(t) \cup Ran(\sigma), \exists i, \zeta(x_i) \neq \zeta(t_i), x_i \in \mathcal{V}ar(t), t_i \in \mathcal{T}(\mathcal{F}, \mathcal{X})\}$. For details, see [41].

Now, the set of ground instances of t which are *innermost* reducible at po-

sition p by the rule $l \to r$ are the previous ground instances expressed using σ, except those which are reducible -by any rule- at a suffix position p' of p, expressed by most general unifiers μ_j. The unifier σ is then constrained by the $\overline{\mu_j}$.

Definition 5.5 (constrained substitution). A *constrained substitution* σ is a formula $\sigma_0 \wedge c$, where c is a conjunction of complement formulas $\bigwedge_{j\in[1..k]} \overline{\sigma_j}$, and $\sigma_0, \sigma_j, j \in [1..k]$ are substitutions.

This leads to an adapted definition of narrowing.

Definition 5.6 (Innermost narrowing). [10] A term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{X}_A)$ *innermost narrows* into a term $t' \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{X}_A)$ at the non-variable position p of t, using the rule $l \to r \in \mathcal{R}$ with the constrained substitution $\sigma = \sigma_0 \wedge \bigwedge_{j\in[1..k]} \overline{\sigma_j}$, which is written $t \rightsquigarrow^{Inn}_{p,l\to r,\sigma} t'$ iff

$$\sigma_0(l) = \sigma_0(t|_p) \text{ and } t' = \sigma_0(t[r]_p)$$

where σ_0 is the most general unifier of $t|_p$ and l and for all suffix positions p' of p in t, $\sigma_j, j \in [1..k]$ is the most general unifier of $\sigma_0 t|_{p'}$ with a left-hand side l' of a rule of $\mathcal{R}$, for all possible rules.

Example 5.7. In Example 4.1, the term $f(X_1, X_2)$ is narrowed with the second rule to the term $f(h(X_3, X_4), s(0))$ with the following most general unifier $\sigma_0 = (X_1 = g(X_3) \wedge X_2 = s(X_4))$ and the disequation constraint $(X_3 \neq s(X_5) \wedge X_3 \neq 0)$. This narrowing step schematizes rewriting steps of the form $f(g(t_1), s(t_2)) \to f(h(t_1, t_2), s(0))$ for ground terms t_1 and t_2 in normal form, provided $t_1 \neq s(t_3)$ for any t_3 and $t_1 \neq 0$.

A few remarks can be made on the choice of variables and on the domain of 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 $\mathcal{V}ar(l) \cap \mathcal{V}ar(t) = \emptyset$. 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 previous definition, $Dom(\sigma) \subseteq \mathcal{V}ar(l) \cup \mathcal{V}ar(t)$ and, as assumed above, we can choose $Ran(\sigma) \cap (\mathcal{V}ar(l) \cup \mathcal{V}ar(t)) = \emptyset$, thus introducing in the range of σ only fresh variables.

Moreover, narrowing is only performed on terms t of $\mathcal{T}(\mathcal{F}, \mathcal{X}_A)$, since an abstracting step is first applied on the initial patterns, of the form $g(x_1, \ldots, x_m)$, replacing $x_1, \ldots, x_m \in \mathcal{X}$ by $X_1, \ldots, X_m \in \mathcal{X}_A$. Then from Definition 5.2 we infer that for the most general unifiers σ produced during the proof process, all variables of $Ran(\sigma)$ are abstraction variables.

Notice also that in our process, we are interested in the narrowing substitution applied to the current term u, 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 u.

The following lifting lemma generalizes the lemma given in [42] and states that $(\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{X}_A), \rightsquigarrow^{Inn})$ P-simulates $(\mathcal{T}(\mathcal{F}), \rightarrow^{Inn})$.

Lemma 5.8 (Innermost lifting Lemma). *[10] Let $\mathcal{R}$ be a rewrite system. Let $s \in \mathcal{T}(\mathcal{F}, \mathcal{X}_A)$, α a (normalized) instantiation and $\mathcal{Y} \subseteq \mathcal{X}_A$ a set of variables such that $\mathcal{V}ar(s) \cup Dom(\alpha) \subseteq \mathcal{Y}$. If $\alpha s \rightarrow^{Inn}_{p,l\rightarrow r} t'$, then there is a term $s' \in \mathcal{T}(\mathcal{F}, \mathcal{X}_A)$ and substitutions $\beta, \sigma = \sigma_0 \wedge \bigwedge_{j\in[1..k]} \overline{\sigma_j}$ such that:*

1. $s \rightsquigarrow^{Inn}_{p,l\rightarrow r,\sigma} s'$,
2. $\beta s' = t'$,
3. $\beta\sigma_0 = \alpha[\mathcal{Y}]$
4. β satisfies $\bigwedge_{j\in[1..k]} \overline{\sigma_j}$

where σ_0 is the most general unifier of $s|_p$ and l and for all suffix positions p' of p in s, $\sigma_j, j \in [1..k]$ is the most general unifier of $\sigma_0 s|_{p'}$ with a left-hand side l' of a rule of $\mathcal{R}$, for all possible rules.

The proof is given in [41].

Note that Middeldorp and Hamoen's Lemma requires the instantiations to be normalized. The above lemma also fulfills this condition: α is always normalized since $\mathcal{V}ar(s) \subset \mathcal{X}_A$.

5.3.2. The case of C-reducibility of Standard Rewriting

Here, we refine the classical definition of narrowing given in Section 2. However, we do not have any lifting lemma, because substitutions may be not normalized. Indeed, abstraction variables represent possibly reducible $\mathcal{C}$-forms. So, contrary to the weak termination proof case, alternating abstract and narrow steps does not model innermost rewriting on ground terms, but standard rewriting and this modelization is not complete, as shown on the following example.

Example 5.9. Let $\mathcal{F} = \{f, g, a, b, c\}$, $\mathcal{C} = \{c\}$, $\mathcal{R} = \{f(x) \rightarrow f(g(a)), g(a) \rightarrow c, c \rightarrow b\}$. Alternating the abstract and narrow mechanisms on the initial pattern $f(x)$ gives the chain $f(X), f(g(a)), f(X'), f(g(a)), f(X'')...$, which does not model the innermost rewriting chain $f(a) \rightarrow f(g(a)) \rightarrow f(c) \rightarrow f(b) \rightarrow f(g(a)) \rightarrow f(c) \rightarrow f(b)....$ Indeed, as the abstraction variable X' cannot be instantiated by the defined symbol b, the subchain $f(b) \rightarrow f(g(a))$ is not captured by $f(X'), f(g(a))$.

But as $\mathcal{C}$-reducibility is a weak property, it is enough for narrowing to simulate at least one rewriting step for any ground instance of the considered term u.

First, to be narrowed, u must not be in $\mathcal{T}(\mathcal{C}, \mathcal{X}_A)$. Indeed, if $u \in \mathcal{T}(\mathcal{C}, \mathcal{X}_A)$, the ground instances of u are $\mathcal{C}$-terms and the proof has to stop with success on u.

Second, if u has defined symbols, every ground instance of u has to be reducible, otherwise $\mathcal{R}$ is not $\mathcal{C}$-reducing. To verify this, we have to narrow u in

all possible ways and to observe whether the narrowing steps model a rewrite step for all possible ground instances of u. Hence the following definition.

Definition 5.10. A set of substitutions Σ is said to *cover a term* $u \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{X}_A)$, or to be *u-covering* iff for any ground instance θu of u, $\exists \sigma \in \Sigma$ such that $Dom(\sigma) \cap \mathcal{V}ar(u) \neq \emptyset$, and $\theta u = \mu\sigma u$ for some instantiation μ.

The previous remarks lead to an adapted definition of narrowing.

Definition 5.11 (Cov-narrowing). Let $\mathcal{R}$ be a rewrite system, t a term of $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{X}_A)$, and Σ the set of narrowing substitutions of t for $\mathcal{R}$. The term t *Cov-narrows* into a term $t' \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{X}_A)$ at the non-variable position p of t, using the rule $l \to r \in \mathcal{R}$ with the substitution $\sigma \in \Sigma$, which is written $t \rightsquigarrow^{Cov}_{p,l\to r,\sigma} t'$ iff

- Σ is t-covering,
- $t \notin \mathcal{T}(\mathcal{C}, \mathcal{X}_A)$,
- $t \rightsquigarrow_{p,l\to r,\sigma} t'$.

Example 5.12. In Example 4.2, the pattern $not(x_1)$ is abstracted into $not(X_1)$, with the abstraction constraint $(x_1{\downarrow_{\mathcal{C}}} = X_1)$. Then, Cov-narrowing is applied on the term $not(X_1)$ with two rules, giving 1 and 0 with respective substitutions $\sigma_1 = (X_1 = 0)$ and $\sigma_2 = (X_1 = 1)$. The set $\Sigma = \{\sigma_1, \sigma_2\}$ is u-covering for $u = not(X_1)$ since for every possible ground instance θX_1 of X_1 (0 or 1), there is σ in Σ such that $\theta(not(X_1)) = \mu\sigma(not(X_1))$ for some μ, which is here equal to identity.

If u is not narrowable with a covering set of narrowing substitutions, some ground instances of u are not reducible at that step, and the proof has to stop with failure. Note that in this case, we cannot conclude that $\mathcal{R}$ is not $\mathcal{C}$-reducing: 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 modeled by our abstract-narrow mechanism.

The following lemma ensures that $(\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{X}_A), \rightsquigarrow^{Cov})$ P-simulates $(\mathcal{T}(\mathcal{F}), \to)$. More precisely, it warrants that with an u-covering set of narrowing substitutions, a narrowing step simulates at least one rewriting step for every ground instance of u.

Lemma 5.13. *[12] Let $\mathcal{R}$ be a rewrite system, u a term of $\mathcal{T}(\mathcal{F}, \mathcal{X}_A)$, and Σ the set of narrowing substitutions of u for $\mathcal{R}$. If Σ is covering u, then every ground instance αu of u is such that $\alpha u \to^{\mathcal{R}}_{p,l\to r,\beta\sigma} t'$ for some instantiation β and some $\sigma \in \Sigma$, and we have $u \rightsquigarrow^{Cov}_{p,l\to r,\sigma} v$ for some v of $\mathcal{T}(\mathcal{F}, \mathcal{X}_A)$, $\beta\sigma = \alpha$ on any variable set $\mathcal{Y} \supseteq \mathcal{V}ar(u) \cup Dom(\alpha)$, and $t' = \beta v$.*

As a first narrowing step is applied on the patterns $g(X_1, \ldots, X_m), g \in \mathcal{F}$, 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 of the patterns $g(x_1, \ldots, x_m)$. The first one is an obvious consequence of Lemma 5.13.

Proposition 5.14. *Let $\mathcal{R}$ be a rewrite system, $g \in \mathcal{D}$, $X_1, \ldots, X_m \in \mathcal{X}_A$. 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 shown by 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 5.15. *Let $\mathcal{R}$ be a rewrite system on $\mathcal{T}(\mathcal{F}, \mathcal{X})$, $X_1, \ldots, X_m \in \mathcal{X}_A$, $x_1, \ldots, x_m \in \mathcal{X}$. 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 $f \in \mathcal{D}$.*

Proof. Let $f \in \mathcal{D}$, and $t = f(t_1, \ldots, t_m)$ a ground instance of $f(x_1, \ldots, x_m)$. The proof is by structural induction on t. Let us suppose that $f(X_1, \ldots, X_m)$ is ground reducible.

- Either t is a constant a, and is ground reducible by hypothesis,
- or t is not a constant, and:
  - either there is a subterm t' of t of the form $g(u_1, \ldots, u_p)$ with $g \in \mathcal{D}$. By induction hypothesis, t' is reducible, and then t is,
  - or every subterm t' of t is of the form $g(u_1, \ldots, u_p)$, with $g \in \mathcal{C}$, so t is a ground instance of $f(X_1, \ldots, X_m)$ and we conclude thanks to the hypothesis.

The converse implication is obvious. □

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 instantiations can only be constructor terms. Let P be the set of constructor patterns $\{c(Y_1, \ldots, Y_m) | c \in \mathcal{C}, Y_1, \ldots, Y_m \in \mathcal{X}_A,$ with $ar(c) = m\}$. For $u \in \mathcal{T}(\mathcal{F}, \mathcal{X}_A)$, let Σ_P^u be the set of all possible pattern substitutions of u i.e., the set $\{\sigma_P^u = (X_1 = t_1, \ldots, X_p = t_p) | \{X_1, \ldots, X_p\} = \mathcal{V}ar(u), t_1, \ldots, t_p \in P\}$.

Then Σ is u-covering if for every $\sigma_P^u \in \Sigma_P^u$, there is $\sigma \in \Sigma$ 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 is in Σ a generalization of σ_P^u. This sufficient condition can be checked automatically.

5.4. 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 step on the current term u with narrowing substitution σ represents a rewriting step for any ground instance of σu. So, when narrowing, σ, considered as the narrowing constraint attached to the narrowing step, is added to the abstraction constraint. Note that if σ is not *compatible* with the abstraction constraint i.e., σ does not satisfy the constraint, the narrowing step is meaningless: it does not correspond to any rewriting step of the considered ground instances. This leads to the introduction of *abstraction constraint formulas* (ACFs for short).

Definition 5.16 (abstraction constraint formula). An *abstraction constraint formula* (ACF in short) is a formula $\bigwedge_i(t_i{\downarrow_{\mathcal{P}}} = t'_i) \wedge \bigwedge_j(x_j = t_j) \wedge \bigwedge_k \bigvee_{l_k}(u_{l_k} \neq v_{l_k})$, where $t_i, t'_i, t_j, u_{l_k}, v_{l_k} \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{X}_A)$, $x_j \in \mathcal{X} \cup \mathcal{X}_A$.

Definition 5.17 (satisfiability of an ACF). An *abstraction constraint formula* $A = \bigwedge_i(t_i{\downarrow_{\mathcal{P}}} = t'_i) \wedge \bigwedge_j(x_j = t_j) \wedge \bigwedge_k \bigvee_{l_k}(u_{l_k} \neq v_{l_k})$, is *satisfiable* iff there is at least one instantiation θ such that $\bigwedge_i(\theta t_i{\downarrow_{\mathcal{P}}} = \theta t'_i) \wedge \bigwedge_j(\theta x_j = \theta t_j) \wedge \bigwedge_k \bigvee_{l_k}(\theta u_{l_k} \neq \theta v_{l_k})$. The instantiation θ is then said to satisfy the ACF A and is called a solution of A.

Integrating a constrained substitution $\sigma = \sigma_0 \wedge \bigwedge_i \bigvee_{j_i}(x_{j_i} \neq t_{j_i})$ to an ACF A is done by adding the formula defining σ to A, thus giving the formula $A \wedge \sigma$. For a better readability on examples, we can propagate σ into A (by applying σ_0 to A), thus getting instantiated abstraction constraints of the form $t_i{\downarrow_{\mathcal{P}}} = t'_i$ from initial abstraction constraints of the form $t_i{\downarrow_{\mathcal{P}}} = X_i$. Note that in the case of $\mathcal{C}$-reducibility, there is no disequality part in the formula.

Example 5.18. In Example 4.2, when Cov-narrowing is applied on the term $not(X_1)$ with two rules, giving 1 and 0 with respective substitutions $\sigma_1 = (X_1 = 0)$ and $\sigma_2 = (X_1 = 1)$, the set of abstraction constraints $A = (x_1{\downarrow_{\mathcal{C}}} = X_1)$ becomes $A = (x_1{\downarrow_{\mathcal{C}}} = 0)$ in the first case and $A = (x_1{\downarrow_{\mathcal{C}}} = 1)$ in the second case, after propagating σ into A.

Example 5.19. In Example 4.2, the abstraction constraint $(x_1{\downarrow_{\mathcal{C}}} = X_1)$ is satisfiable by any instantiation θ such that $\theta x_1 = \theta X_1 = 0$. $A = (x_1{\downarrow_{\mathcal{C}}} = 0)$ and $A = (x_1{\downarrow_{\mathcal{C}}} = 1)$ are satisfiable by any instantiation θ such that $\theta x_1 = 0$ and $\theta x_1 = 1$ respectively.
Now, let us consider for instance an abstraction constraint $h(X_1, X_2){\downarrow_{\mathcal{C}}} = g(X_3) \wedge g(X_3){\downarrow_{\mathcal{C}}} = s(X_4)$. Let us assume the existence of an instantiation θ satisfying the formula: $\theta h(X_1, X_2){\downarrow_{\mathcal{C}}} = \theta g(X_3)$ and $\theta g(X_3){\downarrow_{\mathcal{C}}} = \theta s(X_4)$. The first equality requires $\theta g(X_3)$ to be in normal form, while the second equality implies that $\theta g(X_3)$ has to be rewritten. Therefore such an instantiation θ cannot exist, and hence the constraint is not satisfiable.

An ACF A is attached to each term u in the proof trees; its solutions characterize the interesting instantiations of u i.e., the θu such that θ is a solution of A. When A has no solution, the current node of the proof tree represents no element of $\mathcal{T}(\mathcal{F})$.

For weak termination, such nodes are then useless for the proof. Detecting and suppressing them when applying a narrowing step allows controlling the narrowing mechanism. So we have the choice between generating only the useful nodes of the proof tree, by testing the satisfiability of A at each step, or stopping the proof on a branch on an useless node, by testing the unsatisfiability of A. These are both facets of the same question, but in practice, they are handled in different ways. The satisfiability of A, although undecidable in general, can be proved by exhibiting an instantiation satisfying the constraints of A. Solutions based on constructor terms often hold: they can be generated in an automatic way. More generally, sufficient conditions can be given, which rely on a characterization of P-canonical forms.

Checking the unsatisfiability of A is also undecidable in general, but for weak termination, simple sufficient conditions can be used, very often applicable in practice. They rely on reducibility, unifiability, narrowing and constructor tests, like in the case of strong termination [10].

For $\mathcal{C}$-reducibility, we cannot deal with the unsatisfiability of A. Indeed, let Σ be a covering set of narrowing substitutions for a given term t. The t-covering narrowing step really warrants the reducibility of every αt if every narrowing branch corresponds to an effective rewriting step. This is the case if $A \wedge \sigma$ is satisfiable for every σ of Σ. As testing the unsatisfiability of A relies on sufficient conditions, a negative answer to this test does not imply the satisfiability of A. We thus have to directly test the satisfiability in this case.

Now, as the only relevant instances for the terms considered in the proof are given by the solutions of A, we can refine the satisfiability notion of C. Rather than being satisfied for every ground instance, the ordering constraints of C can be satisfied only by those instances which are solution of A. Hence, the following definition can be used instead of Definition 5.1.

Definition 5.20 (constraint problem). Let A be an abstraction constraint formula and C a conjunction of ordering constraints. The constraint problem C/A is satisfied by an ordering $\succ$ iff for every instantiation θ satisfying A, $\theta t \succ \theta t'$ for every conjunct $t > t'$ of C. C/A is satisfiable iff there is an ordering $\succ$ as above.

Note that C/A may be satisfiable even if A is not.

This restricted definition is very useful, because information in A can help to find an ordering satisfying C.

Example 5.21. In Example 7.2, we have to prove the satisfiability of the ordering constraint $f(x_1) > p(X_2)$. Using $A = (x_1{\downarrow} = s(X_2))$, we prove that any simplification ordering $\succ$ with the precedence $f \succ_{\mathcal{F}} p$ holds. Indeed, assuming that any ground term is greater or equal to any of its normal forms, we have

$\theta x_1 \succeq \theta x_1{\downarrow}$. As any instantiation satisfying A is such that $\theta x_1{\downarrow} \succeq \theta s(X_2)$ and $\theta s(X_2) \succ \theta X_2$, we get $\theta x_1 \succ \theta X_2$.

Definition 5.20 is also used in Example Appendix C.1.

5.5. Relaxing the Induction Hypothesis

It is important to point out the flexibility of the proof method that allows the combination with auxiliary proofs of P using different techniques. When the induction hypothesis cannot be applied on a term u i.e., when it is not possible to decide whether the ordering constraints are satisfiable, it may be possible to prove P for every instantiation of u (which is denoted by $P(\langle u \rangle)$) in another way.

For weak innermost termination, the notion of usable rules, introduced for innermost strong termination in [43], can be very useful to prove P. For details on their use in our context, see [10].

Moreover, $P(\langle u \rangle)$ is true when, in particular, every instantiation of u is a P-canonical form. For weak innermost termination, this is the case when u is not narrowable, and all variables of u are abstraction variables i.e., variables of $\mathcal{X}_A$. Indeed, according to Definition 5.2 and Lemma 5.8, every instantiation of u is a P-canonical form. This includes the cases where u itself is an abstraction variable, and where u is a non narrowable ground term. P is also true on a narrowable u whose variables are all in $\mathcal{X}_A$, and whose narrowing substitutions are not compatible with A. As said in Section 5.4, these narrowing possibilities do not represent any reduction step for the instantiations of u, which are then P-canonical forms. For $\mathcal{C}$-reducibility, $P(\langle u \rangle)$ is true for terms of $\mathcal{T}(\mathcal{C}, \mathcal{X}_A)$. Following Definition 5.2, every ground instance of such terms is already a $\mathcal{C}$-term.

6. The Proof Procedure

6.1. Inference Rules

We are now ready to describe the different steps of the proof mechanism presented in Section 4.

The proof steps generate proof trees in transforming 3-tuples (U, A, C), where $U = \{t\}$ or $\emptyset$, t is the current term on which the property P has to be proved, A is a conjunction of abstraction constraints and 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, for some initial pattern t_{ref}. They allow us to suppose, by induction, the existence of P-canonical forms for the ground instances of $t|_{i_1}, \ldots, t|_{i_p}$. Then, $t|_{i_1}, \ldots, t|_{i_p}$ are abstracted into abstraction variables $X_{i_1}, \ldots, X_{i_p}$. The abstraction constraint $t|_{i_1}{\downarrow_{\mathcal{P}}} = X_{i_1}, \ldots, t|_{i_p}{\downarrow_{\mathcal{P}}} = X_{i_p}$ is added to the ACF A. We call this rule **Abstract**.

  The abstraction positions are chosen so that the abstraction mechanism captures the greatest possible number of rewriting steps: then we abstract

all of the 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 whose ground instances are in P-canonical form.

- The second rule narrows the resulting term u, if its ground instances are not P-canonical forms, 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 5.6 for weak termination and to Definition 5.11 for C-reducibility. 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.

  In the case of $\mathcal{C}$-reducibility, u is narrowed if and only if its ground instances are not in $\mathcal{C}$-form and $\{\sigma_1, \ldots, \sigma_n\}$ is u-covering. Definition 5.11 ensures both conditions. Thanks to the satisfiability test of $A \wedge \sigma_i$, the corresponding narrowing branch is not empty and the narrowing step actually simulates at least one rewriting step for each possible ground instance of u. If u is in $\mathcal{C}$-form or $\{\sigma_1, \ldots, \sigma_n\}$ is not u-covering, the Cov-narrowing relation does not apply. In the second case, the proof process fails, as explained in Section 5.3.2.

- We finally have a **Stop** rule halting the proof process on the current branch of the proof tree, when P can be stated on the ground instances of the current term u. This happens when the whole current term u can be abstracted i.e., when the induction hypothesis is applied on it, or when we have $P(\langle u \rangle)$.

As said before, ordering constraints have to be satisfied by a noetherian ordering. Any simplification ordering holds.

For $\mathcal{C}$-reducibility, we can make further assumptions on the ordering to enable constraints to be satisfied. In particular, for a RPO or LPO $\succ$ 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 one symbol of $\mathcal{D}$ and $u \in \mathcal{T}(\mathcal{C})$.

Exploiting equalities in A using Definition 5.20 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 instantiation θ. As we do not abstract terms of $\mathcal{T}(\mathcal{C}, \mathcal{X}_A)$, if $t_i \neq x_i \in \mathcal{X}$, θt_i contains at least one symbol of $\mathcal{D}$, and then $\theta t_i \succ \theta X_i$ for a RPO or an LPO defined as above. If the root of the initial pattern t_{ref} is also a symbol of $\mathcal{D}$, we also have $\theta t_{ref} \succ \theta X$ for every $X \in \mathcal{X}_A$.

Starting from initial nodes $(T = \{f(x_1, \ldots, x_m)\}, A = \top, C = \top)$, with $f \in \mathcal{F}$ (if f is a constant, then $t_{ref} = f$), the proof process consists in applying the inference rules described in Table 1. The application conditions of these

Table 1: Inference rules applied on the pattern $p = f(x_1, \ldots, x_m)$

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

where t is abstracted into u at positions $i_1, \ldots, i_p \neq \epsilon$
if $COND{-}ABSTRACT$

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

if $t \rightsquigarrow_\sigma^{Inn/Cov} u$ and $COND{-}NARROW$

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

if $COND{-}STOP$

$$G_C(t) = \begin{cases} \top & \text{if } P(\langle t \rangle) \\ p > t & \text{otherwise.} \end{cases} \qquad H_C(t) = \begin{cases} \top & \text{if } P(\langle t \rangle) \text{ or} \\ & A \text{ unsatisfiable} \\ p > t & \text{otherwise} \end{cases}$$

$$\rightsquigarrow_\sigma^{Inn/Cov} = \begin{cases} \rightsquigarrow_\sigma^{Inn} & \text{if } P \text{ is weak innermost termination} \\ \rightsquigarrow_\sigma^{Cov} & \text{if } P \text{ is } \mathcal{C}-\text{reducibility} \end{cases}$$

rules depend on whether the satisfiability or the unsatisfiability of A is checked. These conditions are specified in Tables 2 and 3 respectively.

As said above, the ground terms whose termination is studied are defined by the solutions of A. When the satisfiability of A is checked at each inference step, the nodes of the proof tree exactly model the ground terms generated during the rewriting derivations.

When the satisfiability of A is not checked - in the only case where the property P to be proved is weak termination - nodes are generated in the proof tree, that may represent empty sets of ground terms. So the generated proof trees may have branches that do not represent any derivation on the ground terms. The unsatisfiability test of A is only used to stop the development of meaningless branches as soon as possible, with the sufficient conditions mentioned in Section 5.4.

The previous inference rules, applied to every pattern $t_{ref} = f(x_1, \ldots, x_m)$, where $x_1, \ldots, x_m \in \mathcal{X}$ and $f \in \mathcal{F}$, are combined with the following strategy S:

$$S = repeat^*(try(\textbf{Abstract}),\ try(\textbf{Narrow}),\ try(\textbf{Stop})).$$

"$repeat^*(T_1, \ldots, T_n)$" repeats the control strategies of the sequence ($T_1, \ldots, T_n$) until none of the T_i applies anymore. "$try(T)$" expresses that the rule T is tried and applied, or skipped when it cannot be applied.

According to strategy S, testing the satisfiability of A in conditions of Table 2 can be optimized on the basis of the following remarks. In the first application

Table 2: Conditions for inference rules dealing with the satisfiability of A

$COND-ABSTRACT:\quad (A \wedge t\|_{i_1}\downarrow_{\mathcal{P}} = X_{i_1} \ldots \wedge t\|_{i_p}\downarrow_{\mathcal{P}} = X_{i_p})$ and $(C \wedge G_C(t\|_{i_1}) \ldots \wedge G_C(t\|_{i_p}))$ are satisfiable
$COND-NARROW:\quad A \wedge \sigma$ is satisfiable
$COND-STOP:\quad (C \wedge H_C(t))$ is satisfiable

Table 3: Conditions for inference rules dealing with the unsatisfiability of A (for weak termination only)

$COND-ABSTRACT:\quad C \wedge G_C(t\|_{i_1}) \ldots \wedge G_C(\|t_{i_p})$ is satisfiable
$COND-NARROW:\quad true$
$COND-STOP:\quad (C \wedge H_C(t))$ is satisfiable

of **Abstract** for each initial node, $(A \wedge t|_{i_1}\downarrow_{\mathcal{P}} = X_{i_1} \ldots \wedge t|_{i_p}\downarrow_{\mathcal{P}} = X_{i_p}) = (\top \wedge x_1\downarrow_{\mathcal{P}} = X_1 \ldots \wedge x_m\downarrow_{\mathcal{P}} = X_m)$ is always satisfiable, since the signature admits at least one constructor constant. Moreover, the following possible current application of **Abstract** comes after an application of **Narrow**, for which it has been checked that $A\wedge\sigma$ is satisfiable. So $(A\wedge\sigma\wedge t|_{i_1}\downarrow_{\mathcal{P}} = X_{i_1} \ldots\wedge t|_{i_p}\downarrow_{\mathcal{P}} = X_{i_p})$ is satisfiable as well since $X_{i_1}, \ldots, X_{i_p}$ are fresh variables, not used in $A\wedge\sigma$. So it is useless to verify the satisfiability of $(A \wedge t|_{i_1}\downarrow_{\mathcal{P}} = X_{i_1} \ldots \wedge t|_{i_p}\downarrow_{\mathcal{P}} = X_{i_p})$ in $COND-ABSTRACT$.

This leads to the conditions expressed in Table 4, simplifying those of Table 2.

6.2. The General Theorem

The procedure may diverge, with infinite alternate applications of **Abstract** and **Narrow**. It may also stop on a branch of the proof tree, on a node of the form $(\{t\} \neq \emptyset, A, C)$, when no rule applies anymore. In both cases, nothing can be said on the property P to be proved. It can obviously also stop on a branch thanks to **Stop**, generating a final node of the form $(\emptyset, A, C)$. Such a branch is said to be successful.

Thus, the inductive proof of P is successful if for each proof tree, there is at least one successful branch in the proof tree, corresponding to each possible ground term. Let us develop this point.

Table 4: Optimized conditions for inference rules dealing with the satisfiability of A

$COND-ABSTRACT:\quad (C \wedge G_C(t\|_{i_1}) \ldots \wedge G_C(t\|_{i_p}))$ is satisfiable
$COND-NARROW:\quad A \wedge \sigma$ is satisfiable
$COND-STOP:\quad (C \wedge H_C(t))$ is satisfiable

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 denoted $\sigma_i > \sigma_j$, (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 which are more general than other ones then represent a set of ground instances that contains the other ones. So, for proving P for all of the ground instances at a branching point, it is sufficient to prove P only for the "most general states".

Note that the ignored states may schematize other rewriting steps than those we consider (at different positions, with different rewrite rules). So for the considered instances, if a "most general state" does not give rise to a sufficient set of successful branches, we lose the possibility to test whether the other branches are successful. In practice, this case rarely occurs and the gain is greater in avoiding to consider redundant subsets of instances.

Example 6.1. Let us illustrate our purpose for P being weak innermost termination, with the small example $\{f(a) \to b, f(g(x)) \to c, f(g(a)) \to f(g(a))\}$, where a, b, c are constants. Applying the inference rules to $f(x)$, we get:

$f(x)$	$A = \top$	$C = \top$		
Abstract				
$f(X)$	$A = (x\downarrow = X)$	$C = (f(x) > x)$		
Narrow				
b	$\sigma_1 = (X = a)$	$A = (x\downarrow = a)$	$C = (f(x) > x)$	(1)
c	$\sigma_2 = (X = g(X'))$	$A = (x\downarrow = g(X'))$	$C = (f(x) > x)$	(2)
$f(g(a))$	$\sigma_3 = (X = g(a))$	$A = (x\downarrow = g(a))$	$C = (f(x) > x)$	(3)

Narrow here produces one branch with the substitution $\sigma_1 = (X = a)$, one with the substitution $\sigma_2 = (X = g(X'))$ and one with the substitution $\sigma_3 = (X = g(a))$. The first narrowing branch with σ_1 models rewriting of the ground instances of $f(X)$ satisfying the substitution $\sigma = (X = a)$ i.e., the term $f(a)$. The second branch with σ_2 represents all ground instances of $f(X)$ satisfying the substitution $\sigma = (X = g(X'))$ i.e., all possible ground instances of $f(g(X'))$. The third one with σ_3 represents all ground instances of $f(X)$ satisfying the substitution $\sigma = (X = g(a))$ i.e., the term of $f(g(a))$. As the second branch represents ground instances which are not represented by the other branches, we have to develop it for the termination proof. On the contrary, the third one is useless, and may be not developed.

Therefore, for proving weak termination of all ground instances of $f(x)$, it will be enough to prove weak innermost termination from the state $(\{b\}, A = (x\downarrow = a), C = (f(x) > x))$, and from the state $(\{c\}, A = (x\downarrow = g(X')), C = (f(x) > x))$. We then have:

Stop(*twice*)		
$\emptyset$	$A = (x\downarrow = a)$	$C = (f(x) > x)$
$\emptyset$	$A = (x\downarrow = g(X'))$	$C = (f(x) > x)$

which ends the weak termination proof. Note that the third branch, useless in the proof, would have introduced an infinite succession of **Abstract** and **Narrow** from $(f(g(a)), A = (x\downarrow = g(a)), C = (f(x) > x))$. In this case, discarding this branch not only saves up from useless computations but also avoids divergence of the procedure.

This example shows how at each branching point, one can prune some branches. Let us formalize that now.

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 $(Dom(\sigma) \setminus \mathcal{V}ar(l)) \cup (Dom(\sigma') \setminus \mathcal{V}ar(l'))$, 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.

The recursive definition of a *weakly successful* proof tree is given as follows: either it is reduced to the state $(\emptyset, A, C)$, or at each branching node, it satisfies the following two conditions:

- for each class of equivalent substitutions, there is at least one weakly successful subtree corresponding to a substitution in this class,
- all subtrees corresponding to incomparable substitutions are weakly successful.

So **Narrow** 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.

For equivalent substitutions however, to keep all chances to get a successfull tree, we develop all corresponding subtrees in parallel. As soon as one of them is weakly successful, the other one is cut.

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

Let us assume that the rule **Narrow** is applied with $\rightsquigarrow^{Inn}$ to prove weak innermost termination and with $\rightsquigarrow^{Cov}$ to prove $\mathcal{C}$-reducibility.

Theorem 6.2. *Let $\mathcal{R}$ be a rewrite system on a set $\mathcal{F}$ of symbols, having at least one P-canonical constant. P is true on $\mathcal{T}(\mathcal{F})$ iff there is a noetherian ordering $\succ$ such that for each symbol $f \in \mathcal{F}$, we have $W\!-\!SUCCESS(f, \succ)$.*

An important point is that the ordering $\succ$ has to be the same for all $f \in \mathcal{F}$.

PROOF. The proof is made by induction on properties of weak termination and $\mathcal{C}$-reducibility with the induction ordering $\succ$, using an emptiness lemma, an abstraction lemma, a narrowing lemma and a stopping lemma (see the appendix).

It essentially follows the same line as the one of [10], establishing strong termination under strategies. Let us just point out where are the differences:

- in the definition of canonical forms: instead of considering normal forms according to the strategy, for proving strong termination, we consider here normal forms for weak termination and $\mathcal{C}$-forms for $\mathcal{C}$-reducibility;
- in observing the descendants of the current term t in a proof tree: as strong termination requires us to consider all narrowed forms of t, weak properties just require us to observe one narrowed form for each ground instance of t;
- in the simulation mechanism of the rewriting relation by narrowing: for strong termination, the correctness of simulation is ensured by the lifting lemma. Here, for weak termination, the same lemma is used, whereas for $\mathcal{C}$-reducibility, coveredness of the narrowing step and Lemma 5.13 are required. □

Note that the application of the above theorem can be optimized by developing the proof trees of the only defined symbols. Indeed, applying S to any constructor pattern always gives a successful tree, after applying **Abstract**, **Narrow** and **Stop** at most once (see the complete proof of the theorem).

The subtree cut process is formally described with a complete set of inference rules given in [44].

7. Examples for Weak Innermost Termination

7.1. The Introducing Example

Example 7.1. Let us consider again the rules of of Example 4.1.

$$\begin{array}{ll} f(g(x), s(0)) \to f(g(x), g(x)) & (1) \\ f(g(x), s(y)) \to f(h(x, y), s(0)) & (2) \\ g(s(x)) \to s(g(x)) & (3) \\ g(0) \to 0 & (4) \\ h(x, y) \to g(x). & (5) \end{array}$$

Let us prove weak innermost termination of $\mathcal{R}$ on $\mathcal{T}(\mathcal{F})$, with $\mathcal{F} = \{ar(f) = 2, ar(h) = 2, ar(g) = 1, ar(s) = 1, ar(0) = 0\}$.

Since the defined symbols of $\mathcal{R}$ are f, g, and h, we have to apply the inference rules to $f(x_1, x_2)$, $g(x_1)$ and $h(x_1, x_2)$. We use the rules with conditions of Table 3. The proof trees, given in Fig. 2 show how they are applied. When **Narrow** applies, we specify the narrowing substitution, and in parentheses, the number of the rewrite rule used to narrow.

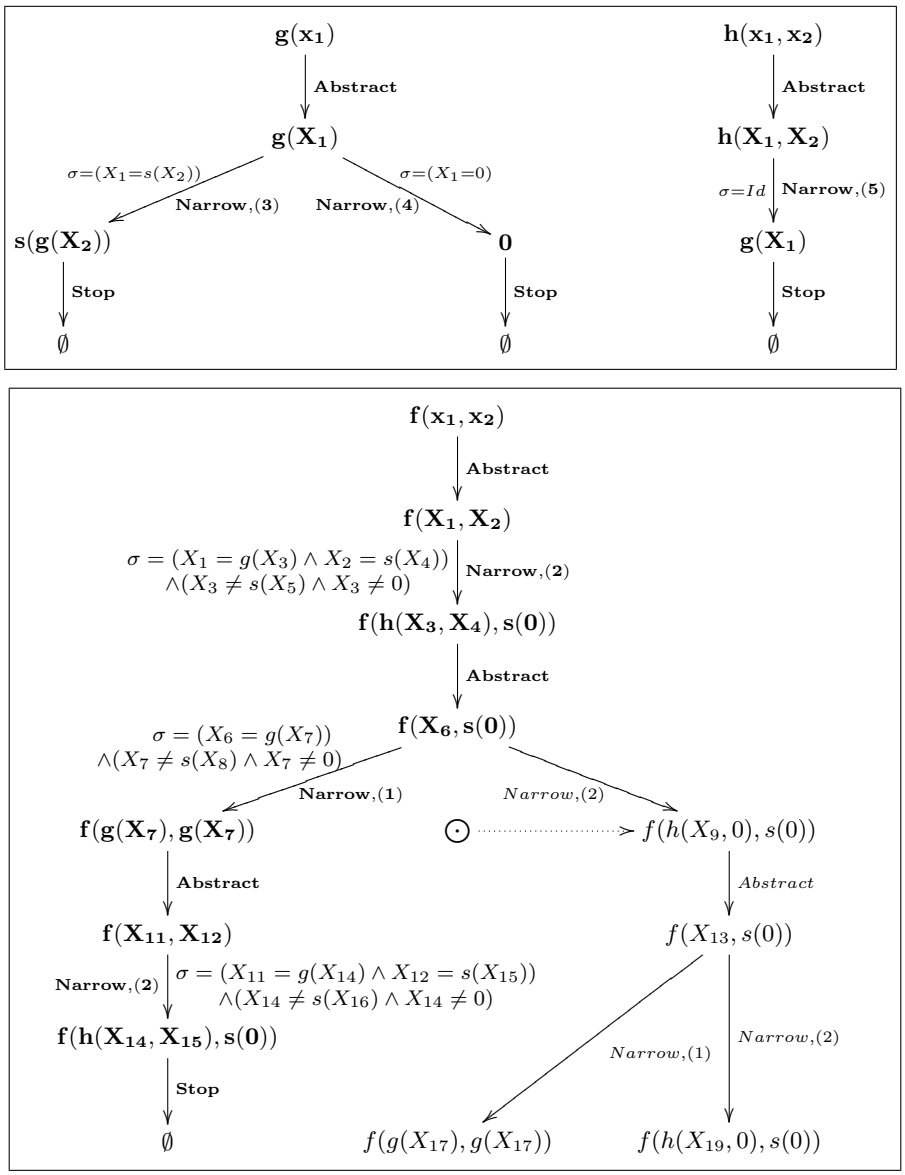

Figure 2: Proof trees for symbols g, h and f.

According to the optimization principle described above, when narrowing produces several steps simulating reduction steps for the same set of ground instances, we develop all branches in parallel to increase the chances to get success i.e., to get branches ending with an application of **Stop**. As soon as one is successful, the other ones are cut.

On the example, the subtree marked by $\odot$ in the proof tree of f is cut as soon as the subtree generated on the left from $\mathbf{f(X_6, s(0))}$ with the same substitution (up to a renaming) $\sigma = (X_6 = g(X_7)) \wedge (X_7 \neq s(X_8) \wedge X_7 \neq 0)$ is successful. The final proof trees are **bold**.

Let us now detail the proof with a linear representation: we give the states of the proof trees together with the position they have in these trees. We write in **bold** the states on which the next inference rule applies.

Applying the inference rules to $(f(x_1, x_2), \top, \top)$, we get:

ϵ $\quad f(x_1, x_2) \quad A = \top \quad C = \top$

Abstract

1 $\quad f(X_1, X_2)$

$A = (x_1{\downarrow} = X_1 \wedge x_2{\downarrow} = X_2) \quad C = (f(x_1, x_2) > x_1, x_2)$

Narrow

1.1 $\quad f(h(X_3, X_4), s(0))$

$\sigma = (X_1 = g(X_3) \wedge X_2 = s(X_4) \wedge X_3 \neq s(X_5) \wedge X_3 \neq 0)$

$A = (x_1{\downarrow} = g(X_3) \wedge x_2{\downarrow} = s(X_4) \wedge X_3 \neq s(X_5) \wedge X_3 \neq 0)$

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

Abstract

1.1.1 $\quad f(X_6, s(0))$

$A = (x_1{\downarrow} = g(X_3) \wedge x_2{\downarrow} = s(X_4) \wedge h(X_3, X_4){\downarrow} = X_6$

$\wedge X_3 \neq s(X_5) \wedge X_3 \neq 0)$

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

Note that the term of the state 1 could also be narrowed into $f(g(X_3'), g(X_3'))$, with rule (1) and the narrowing substitution $\sigma' = (X_1 = g(X_3') \wedge X_2 = s(0)) \wedge (X_3' \neq s(X_5') \wedge X_3' \neq 0)$. But this substitution is strictly less general than the narrowing substitution σ used above, and then, according to strategy S given in Section 6.1, this narrowing possibility is not considered for proving weak termination. More formally, we have $\Sigma = \{\sigma, \sigma'\}$ and, since $\sigma' > \sigma$, $\Sigma_0 = \{\sigma\}$. We then have to develop only the subtrees corresponding to Σ_0.

The second **Abstract** applies on the subterm $h(X_3, X_4)$ because we have $P(\langle h(X_3, X_4)\rangle)$. This is shown thanks to the notion of usable rules. Indeed, the set of usable rules of a term t is a subset of $\mathcal{R}$ including the rules that are likely to be used in all possible rewriting derivations (for the standard rewriting relation) from any ground instance of t. When the set of usable rules terminates, every ground instance of t terminates. For $h(X_3, X_4)$, this set is $\{h(x, y) \to g(x), g(s(x)) \to s(g(x)), g(0) \to 0\}$. It is orientable with any path ordering with the precedence $h \succ_{\mathcal{F}} g \succ_{\mathcal{F}} s$, and thus is terminating. So every ground instance of $h(X_3, X_4)$ (innermost) terminates. Let us go on the proof process.

Narrow

1.1.1.1 $f(g(X_7), g(X_7))$
$\sigma = (X_6 = g(X_7) \wedge X_7 \neq s(X_8) \wedge X_7 \neq 0)$
$A = (x_1{\downarrow} = g(X_3) \wedge x_2{\downarrow} = s(X_4) \wedge h(X_3, X_4){\downarrow} = g(X_7)$
$\wedge X_3 \neq s(X_5) \wedge X_3 \neq 0 \wedge X_7 \neq s(X_8) \wedge X_7 \neq 0)$
$C = (f(x_1, x_2) > x_1, x_2)$

1.1.1.2 $f(h(X_9, 0), s(0))$
$\sigma = (X_6 = g(X_9) \wedge X_9 \neq s(X_{10}) \wedge X_9 \neq 0)$
$A = (x_1{\downarrow} = g(X_3) \wedge x_2{\downarrow} = s(X_4) \wedge h(X_3, X_4){\downarrow} = g(X_9)$
$\wedge X_3 \neq s(X_5) \wedge X_3 \neq 0 \wedge X_9 \neq s(X_{10}) \wedge X_9 \neq 0)$
$C = (f(x_1, x_2) > x_1, x_2)$

Abstract (*on the two states*)

1.1.1.1.1 $f(X_{11}, X_{12})$
$A = (x_1{\downarrow} = g(X_3) \wedge x_2{\downarrow} = s(X_4) \wedge h(X_3, X_4){\downarrow} = g(X_7)$
$\wedge g(X_7){\downarrow} = X_{11} \wedge g(X_7){\downarrow} = X_{12} \wedge X_3 \neq s(X_5) \wedge X_3 \neq 0$
$\wedge X_7 \neq s(X_8) \wedge X_7 \neq 0)$
$C = (f(x_1, x_2) > x_1, x_2)$

1.1.1.2.1 $f(X_{13}, s(0))$
$A = (x_1{\downarrow} = g(X_3) \wedge x_2{\downarrow} = s(X_4) \wedge h(X_3, X_4){\downarrow} = g(X_9)$
$\wedge h(X_9, 0){\downarrow} = X_{13} \wedge X_3 \neq s(X_5) \wedge X_3 \neq 0 \wedge X_9 \neq s(X_{10})$
$\wedge X_9 \neq 0)$
$C = (f(x_1, x_2) > x_1, x_2)$

Narrow (*on the two states*)

1.1.1.1.1.1 $f(h(X_{14}, X_{15}), s(0))$
$\sigma = (X_{11} = g(X_{14}) \wedge X_{12} = s(X_{15}) \wedge X_{14} \neq s(X_{16}) \wedge X_{14} \neq 0)$
$A = (x_1{\downarrow} = g(X_3) \wedge x_2{\downarrow} = s(X_4) \wedge \underline{h(X_3, X_4){\downarrow} = g(X_7)}$
$\underline{\wedge g(X_7){\downarrow} = g(X_{14}) \wedge g(X_7){\downarrow} = s(X_{15}) \wedge X_3 \neq s(X_5)}$
$\underline{\wedge X_3 \neq 0 \wedge X_7 \neq s(X_8) \wedge X_7 \neq 0 \wedge X_{14}} \neq s(X_{16}) \wedge X_{14} \neq 0)$
$C = (f(x_1, x_2) > x_1, x_2)$

1.1.1.2.1.1 $f(g(X_{17}), g(X_{17}))$
$\sigma = (X_{13} = g(X_{17}) \wedge X_{17} \neq s(X_{18}) \wedge X_{17} \neq 0)$
$A = (x_1{\downarrow} = g(X_3) \wedge x_2{\downarrow} = s(X_4) \wedge h(X_3, X_4){\downarrow} = g(X_9)$
$\wedge h(X_9, 0){\downarrow} = g(X_{17}) \wedge X_3 \neq s(X_5) \wedge X_3 \neq 0 \wedge X_9 \neq s(X_{10})$
$\wedge X_9 \neq 0 \wedge X_{17} \neq s(X_{18}) \wedge X_{17} \neq 0)$
$C = (f(x_1, x_2) > x_1, x_2)$

1.1.1.2.1.2 $f(h(X_{19}, 0), s(0))$
$\sigma = (X_{13} = g(X_{19}) \wedge X_{19} \neq s(X_{20}) \wedge X_{19} \neq 0)$
$A = (x_1{\downarrow} = g(X_3) \wedge x_2{\downarrow} = s(X_4) \wedge h(X_3, X_4){\downarrow} = g(X_9)$
$\wedge h(X_9, 0){\downarrow} = g(X_{19}) \wedge X_3 \neq s(X_5) \wedge X_3 \neq 0 \wedge X_9 \neq s(X_{10})$
$\wedge X_9 \neq 0 \wedge X_{19} \neq s(X_{20}) \wedge X_{19} \neq 0)$
$C = (f(x_1, x_2) > x_1, x_2)$

Stop

1.1.1.1.1.1.1 $\emptyset$
$A = (x_1{\downarrow} = g(X_3) \wedge x_2{\downarrow} = s(X_4) \wedge h(X_3, X_4){\downarrow} = g(X_7)$
$\wedge g(X_7){\downarrow} = g(X_{14}) \wedge g(X_7){\downarrow} = s(X_{15}) \wedge X_3 \neq s(X_5)$
$\wedge X_3 \neq 0 \wedge X_7 \neq s(X_8) \wedge X_7 \neq 0 \wedge X_{14} \neq s(X_{16}) \wedge X_{14} \neq 0)$
$C = (f(x_1, x_2) > x_1, x_2)$

Stop applies on the state 1.1.1.1.1.1 because the abstraction constrained formula of this state is not satisfiable. For readability, we have underlined the conjuncts of the formula that make it unsatisfiable, as explained in Example 5.19. The sufficient conditions mentioned in Section 5.4 detect such cases of unsatisfiability.

The branch starting from the state 1.1.1.1, generated from the state 1.1.1 with the narrowing substitution $\sigma = (X_6 = g(X_7)) \wedge (X_7 \neq s(X_8) \wedge X_7 \neq 0)$ is now successful. Hence the branch starting from the state 1.1.1.2, generated from the state 1.1.1 with the narrowing substitution $\sigma = (X_6 = g(X_9)) \wedge (X_9 \neq s(X_{10}) \wedge X_9 \neq 0)$ can be cut, since both substitutions are equivalent.

Applying the inference rules to $(g(x_1), \top, \top)$, we get:

ϵ	$g(x_1)$	$A = \top$	$C = \top$
Abstract			
1	$g(X_1)$	$A = (x_1{\downarrow} = X_1)$	$C = (g(x_1) > x_1)$
Narrow			
1.1	$s(g(X_2))$	$\sigma = (X_1 = s(X_2))$	
		$A = (x_1{\downarrow} = s(X_2))$	$C = (g(x_1) > x_1)$
1.2	0	$\sigma = (X_1 = 0)$	
		$A = (x_1{\downarrow} = 0)$	$C = (g(x_1) > x_1)$
Stop			
1.1.1	$\emptyset$	$A = (x_1{\downarrow} = s(X_2))$	$C = (g(x_1) > x_1)$
1.2.1	$\emptyset$	$A = (x_1{\downarrow} = 0)$	$C = (g(x_1) > x_1)$

Stop applies to the state 1.1 because, thanks to the usable rules, we prove $P(\langle s(g(X_2))\rangle)$. Indeed, the usable rules of $s(g(X_2))$ consist of the system $\{g(s(x)) \to s(g(x)), g(0) \to 0\}$, orientable with any path ordering with the precedence $g \succ_{\mathcal{F}} s$. Since the term 0 is in normal form, **Stop** also applies to the state 1.2.

Applying the inference rules to $(h(x_1, x_2), \top, \top)$, we get :

ϵ	$h(x_1, x_2)$	$A = \top$	$C = \top$
Abstract			
1	$h(X_1, X_2)$	$A = (x_1{\downarrow} = X_1 \wedge x_2{\downarrow} = X_2)$	
		$C = (f(x_1, x_2) > x_1, x_2)$	

Narrow			
1.1	$g(X_1)$	$\sigma = Id$	
		$A = (x_1\downarrow = X_1 \wedge x_2\downarrow = X_2)$	
		$C = (f(x_1, x_2) > x_1, x_2)$	
Stop			
1.1.1	$\emptyset$	$A = (x_1\downarrow = X_1 \wedge x_2\downarrow = X_2)$	
		$C = (f(x_1, x_2) > x_1, x_2)$	

Stop applies to the state 1.1 because, thanks to the usable rules, we prove $P(\langle g(X_1)\rangle)$. Indeed, the usable rules of $g(X_1)$ are the same as the usable rules of $s(g(X_2))$, studied above.

Note that any simplification ordering holds for satisfying all ordering constraints.

7.2. Another example

Example 7.2. Let us consider the following rewrite system, built on $\mathcal{F} = \{ar(f) = 1, ar(p) = 1, ar(s) = 1, ar(0) = 0\}$:

$$\begin{array}{ll} f(x) \to p(s(x)) & (1) \\ f(x) \to p(s(s(x))) & (2) \\ p(s(s(x))) \to p(x) & (3) \\ p(0) \to 0 & (4) \\ p(s(0)) \to f(0) & (5) \end{array}$$

We prove that every ground term t of $\mathcal{T}(\mathcal{F})$ can be innermost normalized with $\mathcal{R}$.

Since the defined symbols of $\mathcal{R}$ are f and p, we have to apply the inference rules to $f(x_1)$ and to $p(x_1)$.

For readability, we do not write the development of branches which are going to be cut. However, we highlight the first state of such branches when it is generated, then write it in *italics* until we highlight it again when the branches initiated by this state are cut.

Let us apply the inference rules on $f(x_1)$ with conditions of Table 3.

ϵ	$f(x_1)$	$A = \top$	$C = \top$
Abstract			
1	$f(X_1)$	$A = (x_1\downarrow = X_1)$	$C = (f(x_1) > x_1)$
Narrow			
1.1	$p(s(X_1))$	$\sigma = Id$	
		$A = (x_1\downarrow = X_1)$	$C = (f(x_1) > x_1)$
1.2	$p(s(s(X_1)))$	$\sigma = Id$	
		$A = (x_1\downarrow = X_1)$	$C = (f(x_1) > x_1)$

In the following, we show that the branch starting from the state 1.1 is successful, which allows us to cut the branch starting from 1.2.

Narrow			
1.1.1	$p(X_2)$	$\sigma = (X_1 = s(X_2))$	
		$A = (x_1{\downarrow} = s(X_2))$	$C = (f(x_1) > x_1)$
1.1.2	$f(0)$	$\sigma = (X_1 = 0)$	
		$A = (x_1{\downarrow} = 0)$	$C = (f(x_1) > x_1)$
1.2	$p(s(s(X_1)))$	$A = (x_1{\downarrow} = X_1)$	$C = (f(x_1) > x_1)$
Stop			
1.1.1.1	$\emptyset$	$A = (x_1{\downarrow} = s(X_2))$	$C = (f(x_1) > x_1, p(X_2))$
1.1.2	$f(0)$	$A = (x_1{\downarrow} = 0)$	$C = (f(x_1) > x_1)$
1.2	$p(s(s(X_1)))$	$A = (x_1{\downarrow} = X_1)$	$C = (f(x_1) > x_1)$
Narrow			
1.1.1.1	$\emptyset$	$A = (x_1{\downarrow} = s(X_2))$	$C = (f(x_1) > x_1, p(X_2))$
1.1.2.1	$p(s(0))$	$A = (x_1{\downarrow} = 0)$	$C = (f(x_1) > x_1)$
		$\sigma = Id$	
1.1.2.2	$p(s(s(0)))$	$A = (x_1{\downarrow} = 0)$	$C = (f(x_1) > x_1)$
		$\sigma = Id$	
1.2	$p(s(s(X_1)))$	$A = (x_1{\downarrow} = X_1)$	$C = (f(x_1) > x_1)$

Stop applies on $p(X_2)$ with any simplification ordering $\succ$ with the precedence $f \succ_{\mathcal{F}} p$. Indeed, using Definition 5.20, we take advantage of $A = (x_1{\downarrow} = s(X_2))$. Assuming that any ground term is greater or equal to any of its normal forms, we get that any instantiation satisfying A is such that $\theta x_1 \succeq \theta s(X_2) \succ \theta X_2$, and hence $\theta f(x_1) \succ \theta p(X_2)$.

The above assumption according to which any ground term is greater than any of its normal forms can be made here, taking advantage of the fact that the studied system is well-covered and innermost terminating, which implies that any ground term in normal form is composed of constructor symbols only [22]. Then, as explained in Section 6.1 for $\mathcal{C}$-reducibility, by choosing for the induction ordering a precedence-based ordering with defined symbols greater than constructor symbols, we get the desired property.

The complete proof tree of f is given in Fig. 3, where $Subtree_f$, given in Fig. 4, is the subtree starting from $p(s(s(X_1)))$, as deep as the subtree starting from $p(s(X_1))$ on the left.

The subtree $\bigodot 1$ is cut as soon as the second subtree generated on the right from $\mathbf{f(0)}$ with the same substitution Id is successful. Then the subtree $\bigodot 2$ ($Subtree_f$) can be cut, since the subtree on the left generated from $\mathbf{f(X_1)}$ with the same substitution Id becomes successful.

The proof tree of p is given in Fig. 5. The subtree $\bigodot 3$ is cut when the second subtree generated on the right from $\mathbf{f(0)}$ with the same substitution Id is successful.

Since the proof trees are both successful, $\mathcal{R}$ is proved weakly innermost terminating on the ground term algebra.

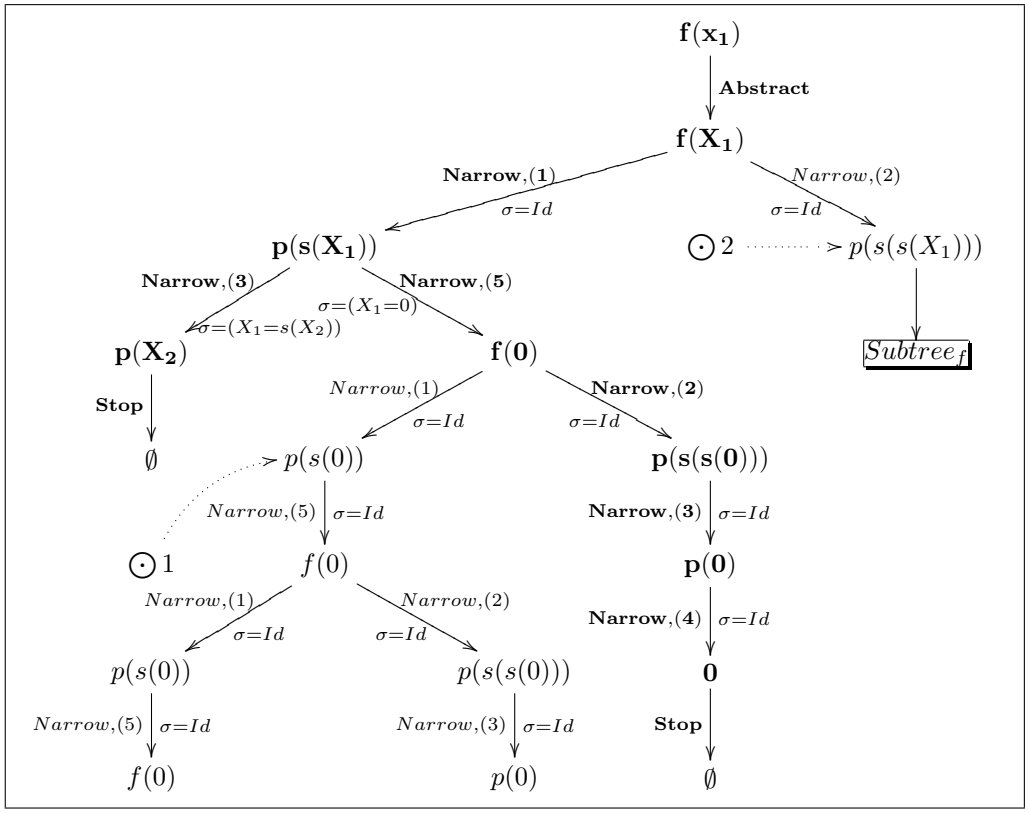

Figure 3: Proof tree for symbol f

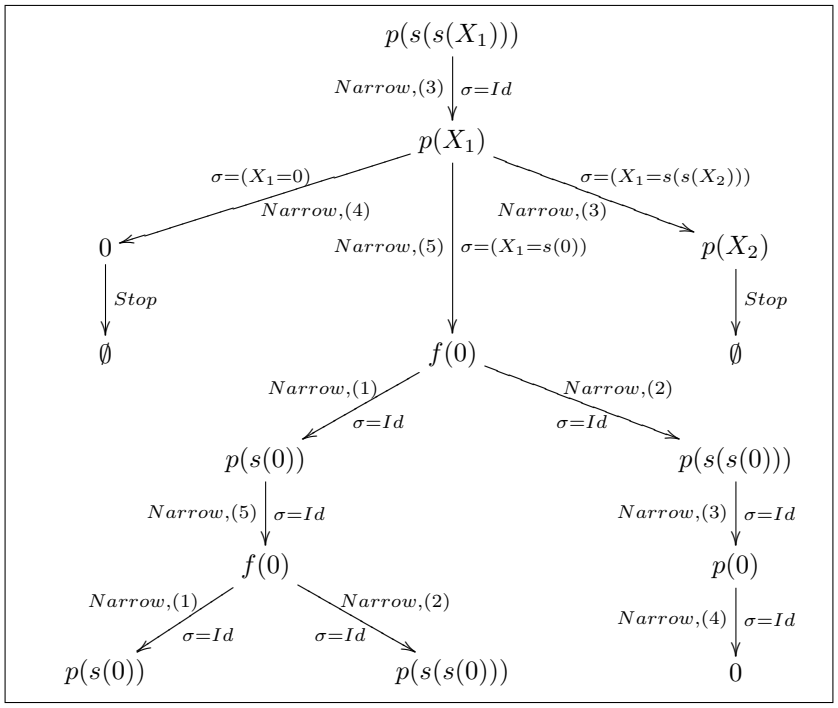

Figure 4: $Subtree_f$

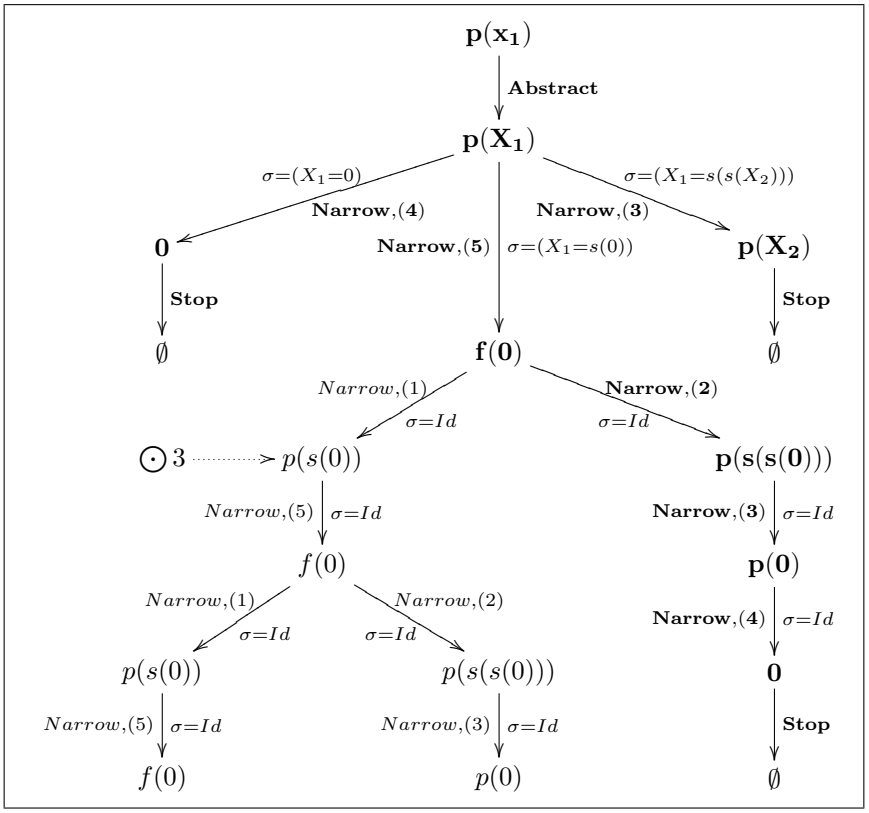

Figure 5: Proof tree for symbol p

Additional examples can be found in [44].

8. Example for $\mathcal{C}$-reducibility

Example 8.1. We prove that the rewrite system

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

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

Applying the rules to $not(x)$ with conditions of Table 4, we get:

$not(x_1)$	$A = \top$	$C = \top$	
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 allowed since $and(X_2, X_3) \notin \mathcal{T}(\mathcal{C}, \mathcal{X}_A)$.

Moreover $\Sigma = \{\sigma_1, \sigma_2\}$ is u-covering for $u = not(X_1)$, as argued in Example 5.12.

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

As explained in Example 5.19, the abstraction constraints $(x_1{\downarrow_{\mathcal{C}}} = X_1)$, $(x_1{\downarrow_{\mathcal{C}}} = 0)$ and $(x_1{\downarrow_{\mathcal{C}}} = 1)$ are satisfiable.

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:

$or(x_1, x_2)$	$A = \top$	$C = \top$
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	$\sigma_1 = (X_1 = 0)$	
	$A = (x_1{\downarrow_c} = 0 \wedge x_2{\downarrow_c} = X_2)$	$C = (or(x_1, x_2) > x_1, x_2)$
1	$\sigma_2 = (X_1 = 1)$	
	$A = (x_1{\downarrow_c} = 1 \wedge x_2{\downarrow_c} = X_2)$	$C = (or(x_1, x_2) > x_1, x_2)$

Stop(*twice*)

$\emptyset$	$A = (x_1{\downarrow_c} = 0 \wedge x_2{\downarrow_c} = X_2)$	$C = (or(x_1, x_2) > x_1, x_2)$
$\emptyset$	$A = (x_1{\downarrow_c} = 1 \wedge x_2{\downarrow_c} = X_2)$	$C = (or(x_1, x_2) > x_1, x_2)$

$\Sigma = \{\sigma_1, \sigma_2\}$ is $or(X_1, X_2)$-covering. **Stop** applies since X_2 and 1 are terms of $\mathcal{T}(\mathcal{C}, \mathcal{X}_A)$.
$A = (x_1{\downarrow_c} = X_1 \wedge x_2{\downarrow_c} = X_2)$ is satisfiable by any instantiation θ such that $\theta x_1 = \theta X_1 = \theta x_2 = \theta X_2 = 0$. $A = (x_1{\downarrow_c} = 0 \wedge x_2{\downarrow_c} = X_2)$ and $A = (x_1{\downarrow_c} = 1 \wedge x_2{\downarrow_c} = X_2)$ are satisfiable by any instantiation θ such that $\theta x_1 = \theta x_2 = \theta X_2 = 0$ and $\theta x_1 = \theta x_2 = \theta X_2 = 1$ respectively.

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

$and(x_1, x_2)$	$A = \top \quad C = \top$

Abstract

$and(X_1, X_2)$	$A = (x_1{\downarrow_c} = X_1 \wedge x_2{\downarrow_c} = X_2)$
	$C = (and(x_1, x_2) > x_1, x_2)$

Narrow

0	$\sigma_1 = (X_1 = 0)$
	$A = (x_1{\downarrow_c} = 0 \wedge x_2{\downarrow_c} = X_2)$
	$C = (and(X_1, X_2) > x_1, x_2)$
X_2	$\sigma_2 = (X_1 = 1)$
	$A = (x_1{\downarrow_c} = 1 \wedge x_2{\downarrow_c} = X_2)$
	$C = (and(X_1, X_2) > x_1, x_2)$
$not(not(and(1, X_2)))$	$\sigma_3 = (X_1 = 1)$
	$A = (x_1{\downarrow_c} = 1 \wedge x_2{\downarrow_c} = X_2)$
	$C = (and(X_1, X_2) > x_1, x_2)$

Stop(*three times*)

$\emptyset$	$A = (x_1{\downarrow_c} = 0 \wedge x_2{\downarrow_c} = X_2)$
	$C = (and(X_1, X_2) > x_1, x_2))$
$\emptyset$	$A = (x_1{\downarrow_c} = 1 \wedge x_2{\downarrow_c} = X_2)$
	$C = (and(X_1, X_2) > x_1, x_2))$
$\emptyset$	$A = (x_1{\downarrow_c} = 1 \wedge x_2{\downarrow_c} = X_2)$
	$C = (and(X_1, X_2) > x_1, x_2)$

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

Stop applies on the first two branches because 0 and X_2 are terms of $\mathcal{T}(\mathcal{C}, \mathcal{X}_A)$, and on the third one because it is issued from the same narrowing substitution as the second one, on which **Stop** applies.

The ACFs A are the same as in the previous proof tree.

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 rewrite system.

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

$$\begin{aligned} 1 &\to true \\ true &\to 1 \\ 0 &\to false \\ false &\to 0 \end{aligned}$$

where $true$ and $false$ are also constructor symbols, we lose the weak termination property. But $\mathcal{C}$-reducibility is proved in the same way as previously, with the same proof trees.

Two other examples, among which the example of introduction, are handled in the appendix. Additional examples can be found in [45].

9. Finding *P*-canonical Forms for Weak Properties

As P is a weak proposition, computing a P-canonical form with the reduction relation $\to$ in general requires to develop the reduction trees with a breadth-first strategy to capture one branch leading to a good element. But such a strategy is often very costly, and it is much better to have hints about the good derivations to compute them directly with a depth-first mechanism.

Our proof process, as it simulates the reduction mechanism, gives complete information on the interesting reduction branches. It allows extracting the exact application of reduction steps that yields an interesting form i.e., a normal form or a constructor form. The breadth-first strategy is used once, for generating the proof trees. Then, to reduce an element, it is enough to follow the reduction scheme simulated by abstraction and narrowing in the proof trees. So a P-canonical form for any element of $\mathcal{T}(\mathcal{F})$ is computed with a reduction strategy ST that is built according to the proof trees establishing the proposition P.

Definition 9.1. Let $\mathcal{R}$ be a rewrite system and P a property proved on $\mathcal{R}$ using Theorem 6.2. The *strategy tree* ST_f associated to $f \in \mathcal{D}$ is the final proof tree obtained from the initial node $(\{f(x_1, \ldots, x_m)\}, \top, \top)$.

The computation of a P-canonical form of any element of $\mathcal{T}(\mathcal{F})$ follows the strategy trees.

Definition 9.2. Let $\mathcal{R}$ be a rewrite system and P a property proved on $\mathcal{R}$ using Theorem 6.2. Let $ST = \{ST_f \mid f \in \mathcal{D}\}$ be the set of strategy trees of $\mathcal{R}$, and $t = f(t_1, \ldots, t_n)$ be an element of $\mathcal{T}(\mathcal{F})$. *Computing a P-canonical form* $can_{ST}(t)$ for $\mathcal{R}$ with respect to ST is done recursively in the following way:

- if $f \in \mathcal{C}$, then $can_{ST}(f(t_1, \ldots, t_n)) = f(can_{ST}(t_1), \ldots, can_{ST}(t_n))$,
- else let $u = f(u_1, \ldots, u_n)$ be the root term of ST_f. We have $t = \theta u$ for some instantiation θ. Then $can_{ST}(t) = can_{TREE}(t, u)$,
  where $can_{TREE}(t, u)$ is:
  - if the step applied on u is **Abstract**, at positions $i_1, \ldots, i_p$, to give a term u', then $can_{TREE}(t', u')$,
    where $t' = t[t'_1]_{i_1} \ldots [t'_p]_{i_p}$, and $t'_j = \begin{cases} t|_{i_j}\downarrow_{\mathcal{P}} & \text{if } P(\langle u|_{i_j}\rangle) \\ can_{ST}(t|_{i_j}) & \text{otherwise} \end{cases}$
  - if the step is **Narrow** with $u \rightsquigarrow^{Inn/Cov}_{p,l\to r,\sigma} u'$, and if there is μ such that $\theta = \mu\sigma$ on $\mathcal{V}ar(u) \cup \mathcal{D}om(\theta)$,
    then $can_{TREE}(t', u')$ where $t' = \mu u'$,
    else t
  - if the step is **Stop**, then $\begin{cases} t\downarrow_{\mathcal{P}} & \text{if } P(\langle u\rangle) \\ can_{ST}(t) & \text{otherwise.} \end{cases}$

where the relation $\rightsquigarrow^{Inn/Cov}$ is $\rightsquigarrow^{Inn}$ when the property P is weak termination and $\rightsquigarrow^{Cov}$ when P is $\mathcal{C}$-reducibility.

The previous definition assumes that if, at some **Abstract** or **Stop** step, the proposition P has been proved on a particular element w during the proof, one is able to build a strategy to compute a P-canonical form of $\theta w\downarrow_{\mathcal{P}}$ of θw for any θ. In the case of weak termination for instance, a simple sufficient condition is that w is proved strongly terminating, which can be established in most cases with the usable rules of w. Under this assumption, the following theorem holds.

Theorem 9.3. *Let $\mathcal{R}$ be a rewrite system proved to have the property P using Theorem 6.2 with the set $ST = \{ST_f \mid f \in \mathcal{D}\}$ of strategy trees. Then for every element $s \in \mathcal{T}(\mathcal{F})$, $can_{ST}(s)$ is a P-canonical form of s.*

The principle of the proof of Theorem 9.3 is the following. The recursive computation of $can_{ST}(s)$ for a term $s = f(s_1, \ldots, s_m) \in \mathcal{T}(\mathcal{F})$ can be represented as a transformation chain of s with respect to ST to compute $can_{ST}(s)$: $s \mapsto s' \ldots t \mapsto t' \ldots s'' \mapsto can_{ST}(s)$, where each step $t \mapsto t'$ corresponds to a step $u \hookrightarrow u'$ in the strategy tree ST_f of the symbol f. At each such step, there are instantiations θ and θ' such that $t = \theta u$ and $t' = \theta' u'$. This corresponds to a recursive computation $can_{TREE}(t, u) = can_{TREE}(t', u')$. Since we consider the case where the proof trees are finite, this transformation chain has a finite number of steps. But each step may recursively involve calls to can_{ST} on subterms of t. Thanks to an induction with the ordering $\succ$ used in Theorem 6.2 for proving P on $\mathcal{R}$, these recursive calls compute the P-canonical forms of these

subterms. Moreover each term t in the transformation chain is an (innermost) reduced form of s, and the last term of the transformation chain is a P-canonical element. Thus it is a P-canonical form of s. For the complete proof, see the appendix.

Example 9.4. We now show on Example 4.1 for weak innermost termination how Definition 9.2 is used to build a strategy for computing a weak normal form for $f(g(f(0,0)), s(0))$.

Let $s \in \mathcal{T}(\mathcal{F})$. We note $t \mapsto t'$ any step of a transformation chain of s with respect to ST, computing $can_{ST}(s)$ according to Definition 9.2.

> **(Step 1 in $\mathbf{ST_f}$: Abstract)** The first step is **Abstract** at positions 1 and 2 by application of the induction hypothesis, and then we get $f(g(f(0,0)), s(0)) \mapsto f(can_{ST}(g(f(0,\ 0))), can_{ST}(s(0)))$. Since s is a constructor, we have $can_{ST}\ (s(0)) = s(can_{ST}(0))$. Since 0 is a constructor constant, we have $can_{ST}(0) = 0$, and finally $can_{ST}(s(0)) = s(0)$. We now have to compute $can_{ST}(g(f(0,\ 0)))$, by following the steps of ST_g.
>
> > **(Step 1 in $\mathbf{ST_g}$: Abstract)** The first step is **Abstract** at position 1 by application of the induction hypothesis, and then we get $g(f(0,0)) \mapsto g(can_{ST}(f(0,0)))$. To compute $can_{ST}(f(0,0))$, we have to follow the steps of ST_f.
> >
> > > **(Step 1 in $\mathbf{ST_f}$: Abstract)** The first step is **Abstract** at positions 1 and 2 by application of the induction hypothesis, and then we get $f(0,0) \mapsto f(can_{ST}(0),\ can_{ST}(0))$. As $can_{ST}(0)$ has been computed to be 0, we have $f(0,0) \mapsto f(0,0)$.
> > >
> > > **(Step 2 in $\mathbf{ST_f}$: Narrow)** The second step is **Narrow** at the top position, with rule (2). The narrowing substitution σ is such that our current term $f(0,0)$ is not a ground instance of $\sigma f(X_1, X_2)$. Therefore $f(0,0) \mapsto f(0,0)$, and finally $can_{ST}(f(0,\ 0)) = f(0,0)$. We then come back to normalization of $g(f(0,0))$.
> >
> > **(Step 2 in $\mathbf{ST_g}$: Narrow)** Our current term is $g(f(0,0))$, and the second step of ST_g is **Narrow** at the top position, with rules (3) and (4). None of the narrowing substitutions σ is such that our current term $g(f(0,0))$ is a ground instance of $\sigma g(X_1)$. Therefore $g(f(0,0)) \mapsto g(f(0,0))$, and finally $can_{ST}(g(f(0,0))) = g(f(0,0))$. We then come back to normalization of our main term.
>
> **(Step 2 in $\mathbf{ST_f}$: Narrow)** Our current term is $f(g(f(0,0)), s(0))$, and the current step in $\mathbf{ST_f}$ is **Narrow** at the top position with rule (2). The narrowing substitution σ is such that our current term is a ground instance of $\sigma f(X_1, X_2)$. So $f(g(f(0,0)), s(0)) \to^{\epsilon,(2)} f(h(f(0,0),0), s(0))$.
>
> **(Step 3 in $\mathbf{ST_f}$: Abstract)** The current step in the proof tree is **Abstract** at position 1 with $P(\langle h(X_3, X_4)\rangle)$, thanks to the usable rules of $h(X_3, X_4)$ which give a strong terminating system. Then we have

$h(f(0,0),0) \mapsto h(f(0,0),0)\downarrow$, and it suffices to rewrite $h(f(0,0),0)$ as long as a normal form is reached, which is guaranteed by the termination of the usable rules. Here we have $h(f(0,0),0) \rightarrow^{\epsilon,(5)} g(f(0,0))$. Finally we get $f(h(f(0,0),0),s(0)) \mapsto f(g(f(0,0)),s(0))$.

(Step 4 in $\mathbf{ST_f}$: Narrow) The current step in the tree is **Narrow** at the top position with rule (1). The narrowing substitution σ is such that our current term is a ground instance of $\sigma f(X_6, s(0))$. So $f(g(f(0,0)),s(0)) \rightarrow^{\epsilon,(1)} f(g(f(0,0)),g(f(0,0)))$.

(Step 5 in $\mathbf{ST_f}$: Abstract) The current step in the tree is **Abstract** at positions 1 and 2 with $P(\langle g(f(0,0))\rangle)$, and then $f(g(f(0,0)),g(f(0,0))) \mapsto f(g(f(0,0))\downarrow, g(f(0,0))\downarrow)$. Since $g(f(0,0))$ is in normal form, we get $f(g(f(0,0)), g(f(0,0))) \mapsto f(g(f(0,0)),g(f(0,0)))$.

(Step 6 in $\mathbf{ST_f}$: Narrow) The current step of ST_f is **Narrow** at the top position, with rule (2). The narrowing substitution σ is such that our current term is a not a ground instance of $\sigma f(X_{11}, X_{12})$. Therefore the normalizing process stops on $f(g(f(0,0)),g(f(0,0)))$, which hence is a normal form of $f(g(f(0,0)),s(0))$.

Example 9.5. Let us also show how the term $f(s(s(s(0))))$ is normalized with the rewrite system of Example 7.2.

(Step 1 in $\mathbf{ST_f}$: Abstract) The first step is **Abstract** at position 1 by application of the induction hypothesis, and then we get $f(s(s(s(0)))) \mapsto f(can_{ST}(s(s(s(0)))))$. Since s is a constructor, we have :
$can_{ST}(s(s(s(0)))) = s(can_{ST}(s(s(0)))) = s(s(can_{ST}(s(0)))) = s(s(s(can_{ST}(0))))$.
Since 0 is a constructor constant, we get $can_{ST}(0) = 0$, and finally $can_{ST}(s(s(s(0)))) = s(s(s(0)))$. We are now on $f(X_1)$ in ST_f, with the current term $f(s(s(s(0))))$ in the derivation.

(Step 2 in $\mathbf{ST_f}$: Narrow) The second step is **Narrow** at the top position, with rule (1). The narrowing substitution σ is such that our current term is a ground instance of $\sigma f(X_1)$. So $f(s(s(s(0)))) \longrightarrow^{\epsilon,(1)} p(s(s(s(s(0)))))$.

(Step 3 in $\mathbf{ST_f}$: Narrow) The third step is **Narrow** at the top position, with rules (3) and (5). For (5), there is no narrowing substitution σ such that our current term $p(s(s(s(s(0)))))$ is a ground instance of $\sigma p(s(X_1))$. For (3), however, this is the case. So we rewrite our current term in the derivation with (3). We get : $p(s(s(s(s(0))))) \longrightarrow^{\epsilon,(3)} p(s(s(0)))$.

(Step 4 in $\mathbf{ST_f}$: Stop) The current step in the tree is **Stop** thanks to the induction hypothesis, and then we get $can_{ST}(p(s(s(0))))$. We now have to follow ST_p to evaluate $can_{ST}(p(s(s(0))))$.

(Step 1 in $\mathbf{ST_p}$: Abstract) Since the first step of ST_p is **Abstract** at position 1 by application of the induction hypothesis, we get $p(s(s(0))) \mapsto p(can_{ST}(s(s(0))))$. Reasoning as previously, we have $can_{ST}(s(s(0))) = s(s(0))$.

(Step 2 in $\mathbf{ST_p}$: Narrow) The second step is **Narrow** at the top position with rules $(3), (4), (5)$. The only rule such that the narrowing substitution σ is such that our current term $p(s(s(0)))$ is a ground instance of $\sigma p(X_i)$ is the rule (3), and then we get : $p(s(s(0))) \longrightarrow^{\epsilon,(3)} p(0)$.

(Step 3 in $\mathbf{ST_p}$: Stop) The current step in ST_p is **Stop** thanks to the induction hypothesis, and then we get $can_{ST}(p(0))$. Once again, we have to follow ST_p to evaluate $can_{ST}(p(0))$.

(Step 1 in $\mathbf{ST_p}$: Abstract) The first step is **Abstract** at position 1, and then we get $p(0) \mapsto p(can_{ST}(0))$. Since 0 is a constructor constant, we have $can_{ST}(0) = 0$.

(Step 2 in $\mathbf{ST_p}$: Narrow) The second step is **Narrow** at the top position with rules $(3), (4), (5)$. The only possible narrowing substitution is the one of the rule (4), and then we get : $p(0) \longrightarrow^{\epsilon,(4)} 0$.

(Step 3 in $\mathbf{ST_p}$: Stop) The current step is **Stop** on a ground term in normal form, which ends the normalizing process on 0, which is then a normal form of $f(s(s(s(0))))$.

10. Conclusion

This paper presents a method to prove weak properties of term rewriting systems, in particular for innermost termination and reducibility to constructor forms, according to the general framework proposed in [13] for inductively proving properties of reduction relations.

Our approach is based on an explicit induction on the property to be proved. To simulate the rewriting derivations of any ground term, we generate proof trees issued from patterns $f(x_1, \ldots, x_m)$ with two mechanisms: abstraction, introducing variables that represent canonical forms for the considered property, and narrowing, schematizing rewriting on ground terms. The induction relation is a noetherian ordering defined by constraints set along the proof.

When all proof trees have a successful branch for all ground instances of the patterns, the weak property of the rewrite system is proved. Then from these successful branches, a strategy leading to a canonical form can be inferred for any ground term. We have shown how to extract the relevant information from the proof trees to guide the reduction process.

The important point to automate our proof principle is the satisfaction of the constraints at each step of the proof. 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 a usual ordering like the Recursive

Path Ordering. If not, they can be delegated to automatic ordering constraint solvers. As illustrated with examples of Section 8, the satisfiability of A can often be proved automatically by generating constructor-based instantiations. The unsatisfiability test can also be automated with the given sufficient conditions cited in Section 5.4. So our proof procedure can be completely automated. For experiments in the case of strong termination, see the results given in [10].

Weak termination is relevant since real programs do not always enjoy the strong termination property. It becomes of practical interest once terminating branches can be reached with a reasonable cost, which is the case with the technique proposed here. To our knowledge, our approach is the first one to prove weak termination of first-order rewrite systems and to deduce from the proof a finite derivation leading to a normal form for every term.

$\mathcal{C}$-reducibility of ground terms is interesting from several points of view. It allows us to warrant a completely computed form for every input of rewrite programs, and directly implies sufficient completeness of specifications. For non terminating programs, it also allows establishing a pseudo-termination property, since the computation can be stopped as soon as the $\mathcal{C}$-form is reached.

Unlike most of the existing methods ensuring completeness, this approach 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 rule with a constructor left-hand side, it even provides a proof of weak termination.

Proving weak termination or $\mathcal{C}$-reducibility of a program and deducing a strategy to reach a normal form or a constructor form can be achieved at *compile-time*. Then, evaluation at *run-time* is made very efficient, since it always leads to a result with a rewriting depth-first strategy, avoiding repeating the use of the costly breadth-first strategy.

The first three applications of our inductive approach to strong termination of innermost, outermost and local strategies have been implemented in a prototype named CARIBOO [6]. To test the expressive power of a rule-based language, and to give a reflexive aspect to our proof tool, we have developed this prototype in the ELAN language, particularly suitable for developing tree structures with a depth-first strategy. The breadth-first strategy, required here for constructing the proof trees for weak properties, would be more difficult to carry out in CARIBOO. Future work is to design a new architecture for CARIBOO, reflecting the generic aspect of our approach as developed in [13] and allowing us to easily develop proof trees with a breadth-first strategy. We also would like to generalize this technique to other weak properties like the reachability of particular forms, called decisions, in rule-based programs for security policies as defined in [46].

Finally, the proof process presented here, based on the inference rules Abstract, Narrow and Stop described in Table 1, has been designed to handle rewriting relations on first-order terms. But it is generic enough to cover several extensions. Thus, terms could be replaced by many- or order-sorted terms, or by equivalence classes of terms modulo a decidable equational theory. The rewriting relation $\rightarrow$ could be replaced accordingly by sorted rewriting, using

sorted matching, or by equational rewriting, using equational matching. Conditional rewriting and constraint rewriting could also be considered, depending on decidability of conditions and of constraint satisfiability. The difficulty would then be to solve more complex constraints in the proof process. As for the narrowing relation modeling rewriting, it could be replaced by the corresponding narrowing extension. Like for the rewriting relations presented in this work, a specific lifting lemma would have to be proved. For abstraction constraints, solving and satisfiability may be an additional issue when equational constraints solvers are needed, since known results are yet partial. For ordering constraints, an interesting point to strengthen is that the ordering relation used for induction only depends on the term structure it applies on, possibly up to an equivalence relation; it is independent from the reduction relation.

Acknowledgments

We are grateful to the referees for carefully reading the manuscript and for the pertinent and constructive remarks that helped us to improve the quality of the presentation.

Appendix A. Proof of Theorem 6.2

THEOREM 6.2. *Let $\mathcal{R}$ be a rewrite system on a set $\mathcal{F}$ of symbols, having at least one P-canonical constant. P is true on $\mathcal{T}(\mathcal{F})$ iff there is a noetherian ordering $\succ$ such that for each symbol $f \in \mathcal{F}$, we have $W{-}SUCCESS(f, \succ)$.*

PROOF. We require an emptiness lemma, an abstraction lemma, a narrowing lemma and a stopping lemma, given after this main proof.

Let us suppose that P is true for every ground term and show that the construction of the proof trees always terminates.

Let $f \in \mathcal{F}$. The initial pattern of its proof tree is $f(x_1, \ldots, x_m)$, on which **Abstract** applies to give $f(X_1, \ldots, X_m), X_1, \ldots, X_m \in \mathcal{X}_A$. Indeed, there is always a noetherian ordering $\succ$ on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ having the subterm property, so $f(x_1, \ldots, x_m) \succ x_1, \ldots, x_m$. Then,

- if P is weak termination,
  - either $f \in \mathcal{C}$, so **Narrow** does not apply and **Stop** applies because we have $P(\langle f(X_1, \ldots, X_m)\rangle)$,
  - or $f \in \mathcal{F} \setminus \mathcal{C}$ and **Narrow** applies on $f(X_1, \ldots, X_m)$, to give terms $v_i \in \mathcal{T}(\mathcal{F}, \mathcal{X}_A)$. As we have $P(\langle v_i\rangle)$, **Stop** applies on each v_i.
- if P is $\mathcal{C}$-reducibility,
  - either $f \in \mathcal{C}$, so $f(X_1, \ldots, X_m) \in \mathcal{T}(\mathcal{C}, \mathcal{X}_A)$ and **Narrow** does not apply but **Stop** applies as previously,
  - or $f \in \mathcal{F} \setminus \mathcal{C}$, so
    - if the narrowing substitutions of $f(X_1, \ldots, X_m)$ are $f(X_1, \ldots, X_m)$-covering, **Narrow** applies on $f(X_1, \ldots, X_m)$ as previously to give terms $v_i \in \mathcal{T}(\mathcal{F}, \mathcal{X}_A)$, on which **Stop** applies,
    - otherwise, **Narrow** does not apply and **Stop** applies because we have $P(\langle f(X_1, \ldots, X_m)\rangle)$.

In the particular case where f is a constant, **Abstract** does not apply. Then the proof works as previously, except if P is $\mathcal{C}$-reducibility and $f \in \mathcal{F} \setminus \mathcal{C}$. In this case, **Narrow** always applies until giving a constructor constant on one of the branches of the proof tree. Otherwise, P would not be true on f. Then **Stop** applies on this constructor constant and the other branches can be cut.

So every proof tree is finite, and $W{-}SUCCESS(f, \succ)$ for every $f \in \mathcal{F}$, with any noetherian ordering $\succ$.

For the converse part, we prove by induction on $\mathcal{T}(\mathcal{F})$ that any ground instance $\theta f(x_1, \ldots, x_m)$ of any term $f(x_1, \ldots, x_m) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ with $f \in \mathcal{F}$ satisfies P, if we have $W{-}SUCCESS(f, \succ)$ for every $f \in \mathcal{F}$.

The induction ordering is constrained along the proof. At the beginning, it has at least to be noetherian. Such an ordering always exists on $\mathcal{T}(\mathcal{F})$ (for instance the embedding relation). Let us denote it $\succ$.

If f is a defined symbol, to each node N of the proof tree of f, characterized by a current term t and the set of constraints A, we associate the set of elements $G = \{\alpha t \mid \alpha \text{ satisfies } A\}$, that is the set of elements of $\mathcal{T}(\mathcal{F})$ represented by N.

Inference rule **Abstract** (resp. **Narrow**) transforms $(\{t\}, A)$ to which is associated $G = \{\alpha t \mid \alpha \text{ satisfies } A\}$, into $(\{t'\}, A')$ to which is associated $G' = \{\beta t' \mid \beta \text{ satisfies } A'\}$ (resp. into $(\{t'_i\}, A'_i), i \in [1..l]$ to which are associated $G'_i = \{\beta_i t'_i \mid \beta_i \text{ satisfies } A'_i\}$).

If αt is reducible, by Abstraction (resp. narrowing) Lemma, for each αt in G, there is a $\beta t'$ (resp. a $\beta_i t'_i$) in G' (resp. in G'_i for some i) such that P is true for $\beta t'$ (resp. for $\beta_i t'_i$) implies P is true for αt.

If αt is irreducible, either the property P to be proved is weak termination, and then αt has the property P. Or the property P to be proved is $\mathcal{C}$-reducibility, and αt is a $\mathcal{C}$-term, so it has the property P. If αt would not be a $\mathcal{C}$-term, as all proof trees are successful, there would be a t-covering **Narrow** step applying on t, and αt would be reducible.

When the inference rule **Stop** applies on $(\{t\}, A, C)$:

- either A is satisfiable, in which case, by Stopping Lemma, P is true on every term of $G = \{\alpha t \mid \alpha \text{ satisfies } A\}$,
- or A is unsatisfiable. In this case, G is empty. The property P to be proved is weak termination. By Emptiness Lemma, all previous nodes on the branch correspond to empty sets G_i, until an ancestor node $N_p = (\{t_p\}, A_p, C_p)$, where A_p is satisfiable, and whose predecessors all have a satisfiable ACF (abstraction constraint formula) A.

  The only rule transforming the state N_p with a satisfiable ACF A_p into N_{p+1} with an unsatisfiable ACF A_{p+1} is **Narrow**. As A_{p+1} is unsatisfiable, the narrowing branch from N_p to N_{p+1} does not correspond to any rewriting step.

  - If there are other narrowing branches from N_p to some N^i_{p+1} with A^i_{p+1} satisfiable, the rule **Stop** eventually applies on them and can be recursively handled.
  - If all narrowing branches from N_p give states N^i_{p+1} with unsatisfiable A^i_{p+1}, every term αt of G_p is irreducible. Otherwise, by Narrowing Lemma, G^i_{p+1} would not be empty for some i. As the property P to be proved is weak termination, every αt trivially has the property P.

Therefore, P is true for the initial set $G_0 = \{\alpha f(x_1, \ldots, x_m) | \forall \alpha\}$.

If f is a constructor, we consider the pattern $f(x_1, \ldots, x_m)$. The proof then works like in the case of defined symbols, but with only two proof steps:

Abstract applies on $f(x_1, \ldots, x_m)$, to give $f(X_1, \ldots, X_m)$. **Narrow** does not apply:

- if P is weak termination, because $f(X_1, \ldots, X_m)$ is not narrowable,
- if P is $\mathcal{C}$-reducibility, because $f(X_1, \ldots, X_m) \in \mathcal{T}(\mathcal{C}, \mathcal{X}_A)$.

Then **Stop** applies:

- if P is weak termination, because $f(X_1, \ldots, X_m)$ is not narrowable and all its variables are in $\mathcal{X}_A$,
- if P is $\mathcal{C}$-reducibility, because $f(X_1, \ldots, X_m) \in \mathcal{T}(\mathcal{C}, \mathcal{X}_A)$.

In the particular case where f is a constant, **Abstract** does not apply on f, so **Stop** directly applies.

Therefore, the proposition P is true for $G_0 = \{\alpha f(x_1, \ldots, x_m) | \forall \alpha\}$ for every $f \in \mathcal{F}$. □

Lemma Appendix A.1 (Emptiness Lemma). *Let $(\{t\}, A, C)$ be a node of any proof tree, giving $(\{t'\}, A', C')$ by application of* **Abstract** *or* **Narrow**. *If A is unsatisfiable, then so is A'.*

Proof. If **Abstract** is applied, $A' = A \wedge t|_{p_1}\downarrow_{\mathcal{P}} = X_{p_1} \ldots \wedge t|_{p_k}\downarrow_{\mathcal{P}} = X_{p_k}$. If **Narrow** is applied, $A' = A \wedge \sigma$. So if A is unsatisfiable, A' is unsatisfiable as well. □

Lemma Appendix A.2 (Abstraction Lemma). *Let $(\{t\}, A, C)$ be a node of any proof tree, giving the node $(\{t' = t[X_j]_{j \in \{i_1, \ldots, i_p\}}\}, A', C')$ by application of* **Abstract**.

For any instantiation α satisfying A, if αt is reducible, there is β such that P is true for $\beta t'$ implies P is true for αt. Moreover, β satisfies A'.

Proof. We prove that $\alpha t \stackrel{*}{\rightarrow}^{(Inn)} \beta t'$, where $\beta = \alpha \cup \bigcup_{j \in \{i_1, \ldots, i_p\}} X_j = \alpha t|_j\downarrow_{\mathcal{P}}$.

First, the abstraction positions in t are chosen so that the $\alpha t|_j$ can be supposed to have the property P. Indeed, each term $t|_j$ is such that:

- either $P(\langle t|_j \rangle)$ is true, so $\alpha t|_j$ has the property P;
- or $t_{ref} > t|_j$ is satisfiable by $\succ$ and then, by induction hypothesis, $\alpha t|_j$ has the property P.

So for every $\alpha t|_j, j \in \{i_1, \ldots, i_p\}$, there is $\alpha t|_j\downarrow_{\mathcal{P}}$.

Whatever the positions $i_1, \ldots, i_p$ in the term t, we have $\alpha t \stackrel{*}{\rightarrow}^{(Inn)} \alpha t[\alpha t|_{i_1}\downarrow_{\mathcal{P}}]_{i_1} \ldots [\alpha t|_{i_p}\downarrow_{\mathcal{P}}]_{i_p} = \beta t'$, for every derivation that reduces the subtems $\alpha t|_j$ into $\alpha t|_j\downarrow_{\mathcal{P}}$, for $j \in \{i_1, \ldots, i_p\}$. As $\beta t'$ represents a reduced form of αt on at least one rewriting branch of αt, then P is true on $\beta t'$ implies P is true on αt.

Clearly in all cases, β satisfies $A' = A \wedge t|_{i_1}\downarrow_{\mathcal{P}} = X_{i_1} \ldots \wedge t|_{i_p}\downarrow_{\mathcal{P}} = X_{i_p}$ provided the X_i are neither in A, nor in $Dom(\alpha)$, which is true since the X_i are fresh variables. □

Lemma Appendix A.3 (Narrowing Lemma). *Let the narrowing relation* $\rightsquigarrow$ *used in* **Narrow** *be such that* $(\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{X}_A), \rightsquigarrow)$ *is a* P*-simulation of* $(\mathcal{T}(\mathcal{F}), \rightarrow)$. *Let* $(\{t\}, A, C)$ *be a node of any proof tree, giving the nodes* $(\{v_i\}, A'_i, C'_i), i \in [1..l]$, *by application of* **Narrow**.

For any instantiation α *satisfying* A, *if* αt *is reducible, then there is* $i \in [1..l]$ *and a substitution* β_i *such that* P *is true on* $\beta_i v_i$ *implies* P *is true on* αt. *Moreover,* β_i *satisfies* A'_i.

PROOF. We reason by case on the property P.

- If the property P is weak termination, if αt is reducible, there is a strictly positive number, say m, of innermost rewriting steps applying to αt, and the innermost lifting lemma can be applied.

  So for each of these m steps, say $\alpha t \rightarrow^{Inn}_{p,l \rightarrow r} t'$, there is a narrowing step modeling it: for a set of variables $\mathcal{Y} \subseteq \mathcal{X}_A$ such that $\mathcal{V}ar(t) \cup Dom(\alpha) \subseteq \mathcal{Y}$, there is a term v and substitutions β and $\sigma = \sigma_0 \wedge \bigwedge_{j \in [1..k]} \overline{\sigma_j}$ such that:

  1. $t \rightsquigarrow^{Inn}_{p,l \rightarrow r,\sigma} v$,
  2. $\beta v = t'$
  3. $\beta\sigma_0 = \alpha[\mathcal{Y}]$,
  4. β satisfies $\bigwedge_{j \in [1..k]} \overline{\sigma_j}$

  where σ_0 is the most general unifier of $t|_p$ and l and $\sigma, j \in [1..k]$ are all most general unifiers of $\sigma_0 t|_{p'}$ and a left-hand side l' of a rule of $\mathcal{R}$, for all position p' which are suffix positions of p in t.

  As **Narrow** is applied in all possible ways, all above narrowing steps are produced. More precisely, the nodes $(\{v_i\}, A'_i, C'_i), i \in [1..l]$ generated by **Narrow** are produced respectively by narrowing steps $t \rightsquigarrow^{Inn}_{p_i,l_i \rightarrow r_i,\sigma^i} v^i, i \in [1..l]$, among which the above narrowing step $t \rightsquigarrow^{Inn}_{p,l \rightarrow r,\sigma} v$.

  Then weak innermost termination of any βv, obtained by one of the m rewriting steps from αt, implies weak innermost termination of αt.

  If instead, **Narrow** is optimized as explained in Section 6.2, then the set of narrowing substitutions Σ is reduced to Σ_0, in such a way that if $\sigma \in \Sigma \setminus \Sigma_0$, the corresponding narrowing branch is not considered by **Narrow**.

  If $\sigma \in \Sigma \setminus \Sigma_0$, there is $\sigma' \in \Sigma_0$ such that $\sigma'_0 < \sigma_0$ or $\sigma'_0 = \sigma_0$, and then $\mu\sigma'_0 = \sigma_0$ for some substitution μ, possibly equal to Id.

  As σ is the narrowing substitution of the narrowing step $t \rightsquigarrow^{Inn}_{p,l \rightarrow r,\sigma} v$ modeling the rewrite step $\alpha t \rightarrow^{Inn}_{p,l \rightarrow r} t'$, then $\alpha = \beta\sigma_0$ for some substitution β. We have $\sigma_0 = \mu\sigma'_0$, so $\beta\sigma_0 = \beta\mu\sigma'_0 = \alpha$ and σ' is the narrowing substitution of the narrowing step $t \rightsquigarrow^{Inn}_{p',l' \rightarrow r',\sigma'} v'$. Then, the narrowing step

$t \rightsquigarrow^{Inn}_{p',l'\to r',\sigma'} v'$, actually produced by **Narrow**, also models a rewriting step from αt.

Then weak innermost termination of $\beta' v'$ (for $\beta' = \beta\mu$) implies weak innermost termination of αt.

Let us now prove that the substitution β connecting one of the m rewriting steps from αt to the narrowing step modeling it satisfies the constraint $A' = A \wedge \sigma_0 \wedge \bigwedge_{j\in[1..k]} \overline{\sigma_j}$.

By Lifting Lemma, we have $\alpha = \beta\sigma_0$ on $\mathcal{Y}$. As we can take $\mathcal{Y} \supseteq Var(A)$, we have $\alpha = \beta\sigma_0$ on $\mathcal{V}ar(A)$.

More precisely, on $Ran(\sigma_0)$, β is such that $\beta\sigma_0 = \alpha$ and on $\mathcal{V}ar(A) \setminus Ran(\sigma_0)$, $\beta = \alpha$. As $Ran(\sigma_0)$ only contains fresh variables, we have $\mathcal{V}ar(A) \cap Ran(\sigma_0) = \emptyset$, so $\mathcal{V}ar(A) \setminus Ran(\sigma_0) = \mathcal{V}ar(A)$. So $\beta = \alpha$ on $\mathcal{V}ar(A)$ and then, β satisfies A.

Moreover, as $\beta\sigma_0 = \alpha$ on $Dom(\sigma_0)$, β satisfies σ_0.

So β satisfies $A \wedge \sigma_0$. Finally, with the point 4. of the lifting lemma, we conclude that β satisfies $A' = A \wedge \sigma_0 \wedge \bigwedge_{j\in[1..k]} \overline{\sigma_j}$.

If **Narrow** is optimized, we prove in the same way that the above substitution β' satisfies $A' = A \wedge \sigma'_0 \wedge \bigwedge_{j\in[1..k]} \overline{\sigma'_j}$.

- If the property P is $\mathcal{C}$-reducibility, **Narrow** is applied on $(\{t = f(u_1, \ldots, u_m)\}, A, C)$ in all possible ways and the set Σ of narrowing substitutions is t-covering. If **Narrow** is optimized as explained in Section 6.2, then Σ is reduced to Σ_0, which is t-covering as well.

  For any α satisfying A, as Σ is t-covering, and by Lemma 5.13, there is a rewriting step $\alpha t \to_p t'$ such that $t \rightsquigarrow^{Cov}_{p,\sigma} v$ with $\sigma \in \Sigma$, for some term $v \in \mathcal{T}(\mathcal{F}, \mathcal{X}_A)$, and there is an instantiation β such that $\beta\sigma = \alpha$ on a set of variables $\mathcal{Y} \supseteq \mathcal{V}ar(t) \cup Dom(\alpha)$, and $t' = \beta v$.

  So $\mathcal{C}$-reducibility of βv implies $\mathcal{C}$-reducibility of αt.

  Let us now prove that β satisfies $A' = A \wedge \sigma$.

  As we can take $\mathcal{Y} \supseteq \mathcal{V}ar(A)$, we have $\alpha = \beta\sigma$ on $\mathcal{V}ar(A)$.

  More precisely, on $Ran(\sigma)$, β is such that $\beta\sigma = \alpha$ and on $\mathcal{V}ar(A) \setminus Ran(\sigma)$, $\beta = \alpha$. As $Ran(\sigma)$ only contains fresh variables, we have $\mathcal{V}ar(A) \cap Ran(\sigma) = \emptyset$, so $\mathcal{V}ar(A) \setminus Ran(\sigma) = \mathcal{V}ar(A)$. Thus $\beta = \alpha$ on $\mathcal{V}ar(A)$ and then, β satisfies A.

  Moreover, as $\beta\sigma = \alpha$ on $Dom(\sigma)$, β satisfies σ.

  So β satisfies $A \wedge \sigma$. □

Lemma Appendix A.4 (Stopping Lemma). *Let $(\{t\}, A, C)$ be a node of any proof tree with A satisfiable, and giving the node $(\emptyset, A', C')$ by application of an inference rule. Then for any instantiation α satisfying A, αt satisfies P.*

PROOF. The only rule giving the node $(\emptyset, A', C')$ is **Stop** and $A' = A$. When **Stop** is applied, then

- either $P(\langle t \rangle)$ and then P is true on αt for every instantiation α,
- or $(t_{ref} > t)$ is satisfiable. Then, for every instantiation α satisfying A, $\alpha t_{ref} \succ \alpha t$. By induction hypothesis, P is true on αt. □

Appendix B. Proof of Theorem 9.3

We proceed by induction on the property to be proved: for $s \in \mathcal{T}(\mathcal{F})$, $can_{ST}(s)$ is a P-canonical form of s. The induction ordering is the ordering $\succ$ used in Theorem 6.2 for proving P on $\mathcal{R}$. Such an ordering exists since, by hypothesis, $\mathcal{R}$ is a rewrite system proved to have the property P with Theorem 6.2.

Theorem 9.3. *Let $\mathcal{R}$ be a rewrite system proved to have the property P using Theorem 6.2 with the set $ST = \{ST_f \mid f \in \mathcal{F}\}$ of strategy trees. Then for every element $s \in \mathcal{T}(\mathcal{F})$, $can_{ST}(s)$ is a P-canonical form of s.*

PROOF. Let $s = f(s_1, \ldots, s_m) \in \mathcal{T}(\mathcal{F})$ be a term to be reduced with respect to ST. We prove that $can_{ST}(s)$ is a P-canonical form of s, by induction with the ordering $\succ$ used in Theorem 6.2 for proving P on $\mathcal{R}$.

- if $f \in \mathcal{C}$, according to Definition 9.2, $can_{ST}(f(s_1, \ldots, s_m)) = f(can_{ST}(s_1), \ldots, can_{ST}(s_n))$. As $f(s_1, \ldots, s_m) \succ s_1, \ldots, s_n$, then, by induction hypothesis, $can_{ST}(s_1), \ldots, can_{ST}(s_n)$ are P-canonical forms of $s_1, \ldots, s_n$ respectively. As $f \in \mathcal{C}$, $t = f(can_{ST}(s_1), \ldots, can_{ST}(s_n))$ is a P-canonical form for s:
  - if P is weak innermost termination, t is irreducible,
  - if P is $\mathcal{C}$-reducibility, t is a $\mathcal{C}$-form.
- if $f \in \mathcal{D}$, s is an instance of a pattern $f(x_1, \ldots, x_n)$, which is the root term of a strategy tree ST_f and the computation of $can_{ST}(s)$ follows a finite branch of this proof tree. Let $s \mapsto s' \ldots t \mapsto t' \mapsto s'' \mapsto can_{ST}(s)$ be the transformation chain where each step $t \mapsto t'$ corresponds to a step $u \hookrightarrow u'$ of the strategy tree ST_f, whose associated ACFs are A and A' respectively.

  Let us prove that at each step $t \mapsto t'$ of the transformation chain, such that $t = \theta u$ with θ satisfying the ACF A (which is the case for s since $A = \top$), and t is an (innermost) reduced form of s (which is the case for s),
  - there is an instantiation θ' such that $t' = \theta' u'$ and θ' satisfies A',
  - t' is an (innermost) reduced form of s.

  Let us look at the different cases.

– If the step applied on u in the strategy tree of f is **Abstract** at positions $i_1, \ldots, i_p$, then according to Definition 9.2, $t' = t[t'_1]_{i_1} \ldots [t'_p]_{i_p}$, and the t'_j are $t|_{i_j}\downarrow_{\mathcal{P}}$ if we do have $P(\langle u|_{i_j}\rangle)$, and $can_{ST}(t|_{i_j})$ otherwise.

  * For $j \in \{1, \ldots, p\}$ such that $P(\langle u|_{i_j}\rangle)$, every instantiation of $u|_{i_j}$ has the property P. Since t is an instantiation of u, $t|_{i_j}$ is an instantiation of $u|_{i_j}$, hence $t|_{i_j}$ has the property P, and has at least one P-canonical form $t'_j = t|_{i_j}\downarrow_{\mathcal{P}}$, which is an innermost reduced form of t_{i_j}.
  * For $j \in \{1, \ldots, p\}$ such that we do not have $P(\langle u|_{i_j}\rangle)$, we have $u_{ref} \succ u|_{i_j}$ for some u_{ref}, and then, by induction hypothesis, $can_{ST}(\theta u|_{i_j})$ is a P-canonical form of $\theta u|_{i_j}$ for every instantiation θ. As previously, $t|_{i_j}$ is an instantiation of $u|_{i_j}$, so $can_{ST}(t|_{i_j})$ is a P-canonical form of $t|_{i_j}$.

  Then, the term $t[t'_1]_{i_1} \ldots [t'_p]_{i_p}$ is an (innermost) reduced form of t for $\mathcal{R}$. Since, by hypothesis, t is an (innermost) reduced form of s for $\mathcal{R}$, we conclude that t' is an (innermost) reduced form of s for $\mathcal{R}$.

  In the Abstract step $u \hookrightarrow u'$, u' is of the form $u[X_1]_{i_1} \ldots [X_p]_{i_p}$ where $i_1, ..., i_p$ are the abstraction positions of u and the X_j, for $\in \{1, \ldots, p\}$, are new abstraction variables. As t is an instantiation of u, $t' = t[t'_1]_{i_1} \ldots [t'_p]_{i_p}$ and $t'_1, \ldots, t'_p$ are P-canonical forms, then t' is an instantiation $\theta' u'$ of u', where θ' is the instantation $(X_1 \mapsto t'_1 \ldots, X_p \mapsto t'_p)$. Thanks to the proof of Abstraction Lemma, θ' satisfies the abstraction constraint associated to u'.

– if the step applied on u is **Narrow**, according to Definition 9.2, two cases may occur:

  * either $t = \theta u \rightarrow^{(Inn)}_{p,l\to r,\mu} t'$, where the used branch of the step **Narrow** is such that $u \rightsquigarrow^{Inn/Cov}_{p,l\to r,\sigma} u'$, with $\theta = \mu\sigma[\mathcal{V}ar(u) \cup \mathcal{D}om(\theta)]$ and $t' = \mu u'$. Then t' is an (innermost) reduced form of t. Thus, reasoning as in the previous case, as t is an (innermost) reduced form of s for $\mathcal{R}$, we conclude that t' is an (innermost) reduced form of s for $\mathcal{R}$. Clearly, t' is an instantiation of u' and $\theta' = \mu$. Thanks to the proof of Narrowing Lemma, θ' satisfies A''.
  * Or the latter rewriting is not possible (this is the case where σ such that $t = \mu\sigma u$ for some μ does not exist), so t is already in P-canonical form. Indeed, if the property P is weak termination, if there is no narrowing step corresponding to θu, θu is in normal form. If P is C-reducibility, this case does not arise because the narrowing step is u-covering. Then the process stops on t, which is, by hypothesis, an innermost reduced form of s for $\mathcal{R}$.

– if the step applied on u is **Stop**, which is a terminal step in the strategy tree, it is a terminal step in the transformation chain of

source s. Let us prove that t gives a term t', which is a P-canonical form of t.

According to Definition 9.2, $t' = t{\downarrow_{\mathcal{P}}}$ if we do have $P(\langle u \rangle)$ and $t' = can_{ST}(t)$ otherwise.

In the first case, every instantiation of u has the property P. Since t is an an instantiation of u, then it has the property P, so $t' = t{\downarrow_{\mathcal{P}}}$ exists.

In the second case, we have $u_{ref} \succ u$ for some u_{ref}, so by the main induction hypothesis, $can_{ST}(\theta u)$ is a P-canonical form of θu for every instantiation θ. As previously, t is an instantiation of u, and then $can_{ST}(t)$ is a P-canonical form of t.

Note that we do not have here the case where **Stop** applies because A is unsatisfiable. As $t = \theta u$, θ is a solution of A, so A is satisfiable.

To conclude, the **Stop** step and the special case in **Narrow**, when there is no u corresponding to t, are the only steps where the reduction process with respect to ST stops. For the case of a **Stop** case, the term t' is a P-canonical element and a reduced form of s, so it is a P-canonical form of s. For the special case in **Narrow**, t is a reduced form of s and is already in P-canonical form, which ends the proof for $f \in \mathcal{D}$. □

Appendix C. Additional examples

Example Appendix C.1. Let us now consider the second example of the introduction.

$$
\begin{aligned}
&quot(0, s(y), s(z)) \to 0\\
&quot(s(x), s(y), z) \to quot(x, y, z)\\
&quot(x, 0, s(z)) \to s(quot(x, s(z), s(z)))\\
&quot(x, y, 0) \to error\\
&quot(error, y, z) \to error\\
&quot(x, error, z) \to error\\
&quot(x, y, error) \to error
\end{aligned}
$$

$\mathcal{F} = \{ar(quot) = 3, ar(s) = 1, ar(0) = 0, ar(error) = 0\}$ and $\mathcal{C} = \{s, 0, error\}$.

So, either we prove innermost termination of the rewrite system, (the dependency pair method [47] works), and we infer sufficient completeness, with Propositions 5.14, 5.15 and Theorem 3.5, 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 to $quot(x_1, x_2, x_3)$ with conditions of Table 4, we get:

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

Applying **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 $P = \{0, error, s(X) | X \in \mathcal{X}_A\}$, 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, for any θ satisfying A, we have $quot(\theta x_1, \theta x_2, \theta x_3) \succ quot(\theta X_1', \theta X_2', \theta X_3)$ for a LPO with any precedence and a left-to-right status for

quot. Note that unlike in Example 7.2, the coveredness of $\mathcal{R}$ does not imply that $\theta x_1 \succeq \theta X_1$, because $\mathcal{R}$ is not terminating and X_1 is a priori not a normal form of x_1. However, $\mathcal{R}$ does not contain any rule whose left-hand side is a $\mathcal{C}$-term. Then, if θx_1 is a $\mathcal{C}$-term, it is in normal form, and then $\theta x_1 = \theta x_1{\downarrow_c}$. Thanks to Definition 5.20, using A, we have $\theta x_1{\downarrow_c} = s(\theta X_1')$ and then $\theta x_1 \succ \theta X_1'$. If θx_1 contains a defined symbol, then $\theta x_1 \succ \theta X_1'$ (see Section 6.1). In a similar way, we obtain $\theta x_2 \succ \theta X_2'$.

Now, if θx_3 contains a defined symbol, as previously for x_1, we have $\theta x_3 \succ \theta X_3$. If θx_3 is a $\mathcal{C}$-term, we get $\theta x_3 = \theta x_3{\downarrow_c} = \theta X_3$. Then $\theta x_3 \succeq \theta X_3$.

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

Then we get:

Stop (**six times**)

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

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

Narrow

$s(0)$	$\sigma_1 = (X_1 = 0)$ $A = (x_1{\downarrow_c} = 0 \wedge x_2{\downarrow_c} = 0 \wedge x_3{\downarrow_c} = s(X'_3))$ $C = (quot(x_1, x_2, x_3) > x_1, x_2, x_3)$
$s(error)$	$\sigma_2 = (X_1 = error)$ $A = (x_1{\downarrow_c} = error \wedge x_2{\downarrow_c} = 0 \wedge x_3{\downarrow_c} = s(X'_3))$ $C = (quot(x_1, x_2, x_3) > x_1, x_2, x_3)$
$s(quot(X'_1, X'_3, s(X'_3))$	$\sigma_3 = (X_1 = s(X'_1))$ $A = (x_1{\downarrow_c} = s(X'_1) \wedge x_2{\downarrow_c} = 0 \wedge x_3{\downarrow_c} = s(X'_3))$ $C = (quot(x_1, x_2, x_3) > x_1, x_2, x_3)$

$\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$ 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, $quot(\theta x_1, \theta x_2, \theta x_3) \succ s(quot(\theta X'_1, \theta X'_3, s(\theta X'_3))$ for any θ satisfying A. Indeed, as $quot \in \mathcal{D}$ and $s \in \mathcal{C}$, we have $quot \succ_{\mathcal{F}} s$. We then have to verify that $quot(\theta x_1, \theta x_2, \theta x_3) \succ quot(\theta X'_1, \theta X'_3, s(\theta X'_3)$. In a similar way as previously, if θx_1 is a $\mathcal{C}$-term, $\theta x_1 = \theta x_1{\downarrow_c} = s(\theta X'_1) \succ \theta X'_1$. If θx_1 contains a defined symbol, $\theta x_1 \succ \theta x_1{\downarrow_c} = s(\theta X'_1) \succ \theta X'_1$. We also get $\theta x_3 \succeq s(\theta X'_3) \succ \theta X'_3$.

By definition of the LPO, as we have $\theta x_1 \succ \theta X'_1$, we have to verify that $quot(\theta x_1, \theta x_2, \theta x_3) \succ \theta X'_3, s(\theta X'_3)$. This is true since $\theta x_3 \succeq s(\theta X'_3) \succ \theta X'_3$.

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

The different ACFs A are easily showed satisfiable, replacing variables in an adequate way by $0, s(0)$ and $error$.

Example Appendix C.2. The following RS

$$
\begin{array}{l}
f(x) \to if(x, c, f(true)) \\
if(true, x, y) \to x \\
if(false, x, y) \to y \\
if(c, x, y) \to c
\end{array}
$$

with $\mathcal{F} = \{ar(if) = 3, ar(f) = 1, ar(c) = 0, ar(true) = 0, ar(false) = 0\}$

and $\mathcal{C} = \{c, true, false\}$ is neither terminating, nor even innermost terminating. Ve prove that it is $\mathcal{C}$-reducing.

Applying the rules with conditions of Table 4 to $f(x_1)$, we get:

$f(x_1)$	$A = \top, \quad C = \top$
Abstract	
$f(X_1)$	$A = (x_1{\downarrow_c} = X_1) \quad C = (f(x_1) > x_1)$
Narrow	
$if(X_1, c, f(true))$	$\sigma = Id$
	$A = (x_1{\downarrow_c} = X_1) \quad C = (f(x_1) > x_1)$

Narrow		
c	$\sigma_1 = (X_1 = true)$	
	$A = (x_1{\downarrow_c} = true)$	$C = (f(x_1) > x_1)$
$f(true)$	$\sigma_2 = (X_1 = false)$	
	$A = (x_1{\downarrow_c} = false)$	$C = (f(x_1) > x_1)$
c	$\sigma_3 = (X_1 = c)$	
	$A = (x_1{\downarrow_c} = c)$	$C = (f(x_1) > x_1)$
$if(X_1, c, if(true, c, f(true)))$	$\sigma_4 = Id$	
	$A = (x_1{\downarrow_c} = X_1)$	$C = (f(x_1) > x_1)$

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

As the first three 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 $if(X_1, c, if(true, c, f(true)))$) is applied again, which gives the following states.

c	$\sigma_1 = (X_1 = true)$	
	$A = (x_1{\downarrow_c} = true)$	$C = (f(x_1) > x_1)$
$if(true, c, f(true))$	$\sigma_2 = (X_1 = false)$	
	$A = (x_1{\downarrow_c} = false)$	$C = (f(x_1) > x_1)$
c	$\sigma_3 = (X_1 = c)$	
	$A = (x_1{\downarrow_c} = c)$	$C = (f(x_1) > x_1)$
$if(X_1, c, c)$	$\sigma_4 = Id$	
	$A = (x_1{\downarrow_c} = X_1)$	$C = (f(x_1) > x_1)$
$if(X_1, c, if(true, c,$ $if(true, c, f(true))))$	$\sigma_5 = Id$	
	$A = (x_1{\downarrow_c} = X_1)$	$C = (f(x_1) > x_1)$

Again, we can cut the first three branches. **Stop** applies on the fourth state since we have $f(x_1) \succ if(X_1, c, c)$ for a RPO with the precedence $f >_{\mathcal{F}} if, c$. Indeed, we have $f(x_1) \succ c$, and $f(x_1) \succ X_1$ following the remarks of Section 6.1.

Then, the last branch, whose narrowing substitution is the same as for the fourth one, can be cut.

Applying the rules to $if(x_1, x_2, x_3)$, we get:

$$if(x_1, x_2, x_3) \qquad A = \top, \quad C = \top$$

Abstract

$$if(X_1, X_2, X_3) \quad A = (x_1{\downarrow_c} = X_1, x_2{\downarrow_c} = X_2, x_3{\downarrow_c} = X_3)$$
$$C = (if(x_1, x_2, x_3) > x_1, x_2, x_3)$$

Narrow

X_2
$$\sigma_1 = (X_1 = true)$$
$$A = (x_1{\downarrow_c} = true \wedge x_2{\downarrow_c} = X_2 \wedge x_3{\downarrow_c} = X_3)$$
$$C = (if(x_1, x_2, x_3) > x_1, x_2, x_3)$$

X_3
$$\sigma_2 = (X_1 = false)$$
$$A = (x_1{\downarrow_c} = false \wedge x_2{\downarrow_c} = X_2 \wedge x_3{\downarrow_c} = X_3)$$
$$C = (if(x_1, x_2, x_3) > x_1, x_2, x_3)$$

c
$$\sigma_3 = (X_1 = c)$$
$$A = (x_1{\downarrow_c} = c \wedge x_2{\downarrow_c} = X_2 \wedge x_3{\downarrow_c} = X_3)$$
$$C = (if(x_1, x_2, x_3) > x_1, x_2, x_3)$$

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

$$\emptyset \quad A = (x_1{\downarrow_c} = true \wedge x_2{\downarrow_c} = X_2 \wedge x_3{\downarrow_c} = X_3)$$
$$C = (if(x_1, x_2, x_3) > x_1, x_2, x_3)$$

$$\emptyset \quad A = (x_1{\downarrow_c} = false \wedge x_2{\downarrow_c} = X_2 \wedge x_3{\downarrow_c} = X_3)$$
$$C = (if(x_1, x_2, x_3) > x_1, x_2, x_3)$$

$$\emptyset \quad A = (x_1{\downarrow_c} = c \wedge x_2{\downarrow_c} = X_2 \wedge x_3{\downarrow_c} = X_3)$$
$$C = (if(x_1, x_2, x_3) > x_1, x_2, x_3)$$

The different ACFs A are easily showed satisfiable, replacing variables in an adequate way by $c, true$ and $false$.

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

[2] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, J. F. Quesada, Maude: Specification and programming in rewriting logic, Theoretical Computer Science 285 (2002) 187–243.

[3] K. Futatsugi, A. Nakagawa, An overview of the Cafe specification environment – an algebraic approach for creating, verifying, and maintaining formal specifications over networks, in: Proceedings of the 1st IEEE International Conference on Formal Engineering Methods, Hiroshima, Japan, 1997, p. 170.

[4] P. Borovanský, C. Kirchner, H. Kirchner, P.-E. Moreau, C. Ringeissen, An Overview of ELAN, in: C. Kirchner, H. Kirchner (Eds.), Proceedings of the 2nd International Workshop on Rewriting Logic and its Applications, Vol. 15 of Electronic Notes in Theoretical Computer Science, Elsevier Science Publishers B. V. (North-Holland), Pont-à-Mousson, France, 1998, pp. 55–70.

[5] P.-E. Moreau, C. Ringeissen, M. Vittek, A Pattern Matching Compiler for Multiple Target Languages, in: G. Hedin (Ed.), Proceedings of the 12th Conference on Compiler Construction, Vol. 2622 of Lecture Notes in Computer Science, Springer, Warsaw, Poland, 2003, pp. 61–76.

[6] O. Fissore, I. Gnaedig, H. Kirchner, CARIBOO : An induction based proof tool for termination with strategies, in: Proceedings of the 4th International Conference on Principles and Practice of Declarative Programming, ACM Press, Pittsburgh, USA, 2002, pp. 62–73.

[7] O. Fissore, I. Gnaedig, H. Kirchner, Termination of rewriting with local strategies, in: M. P. Bonacina, B. Gramlich (Eds.), Selected papers of the 4th International Workshop on Strategies in Automated Deduction, Vol. 58 of Electronic Notes in Theoretical Computer Science, Elsevier Science Publishers B. V. (North-Holland), Siena, Italy, 2001, pp. 155–188.

[8] O. Fissore, I. Gnaedig, H. Kirchner, Outermost ground termination, in: Proceedings of the 4th International Workshop on Rewriting Logic and Its Applications, Vol. 71 of Electronic Notes in Theoretical Computer Science, Elsevier Science Publishers B. V. (North-Holland), Pisa, Italy, 2002, pp. 188–207.

[9] I. Gnaedig, H. Kirchner, Termination of rewriting strategies: a generic approach, in: H.-W. L. M. Hofmann (Ed.), Proceedings of the APPSEM II Workshop 2005, APPSEM II & Ludwig Maximilians Universität München, Chiemsee, Germany, 2005.

[10] I. Gnaedig, H. Kirchner, Termination of rewriting under strategies, ACM Transactions on Computational Logic 10 (2) (2009) 1–52.
URL http://hal.inria.fr/inria-00182432/en/

[11] O. Fissore, I. Gnaedig, H. Kirchner, A proof of weak termination providing the right way to terminate, in: Proceedings of the 1st International Colloquium on Theoretical Aspects of Computing, Vol. 3407 of Lecture Notes in Computer Science, Springer-Verlag, Guiyang, China, 2004, pp. 356–371.

[12] I. Gnaedig, H. Kirchner, Computing Constructor Forms with Non Terminating Rewrite Programs, in: Proceedings of the 8th ACM-SIGPLAN International Symposium on Principles and Practice of Declarative Programming, ACM Press, Venice, Italy, 2006, pp. 121–132.

[13] I. Gnaedig, H. Kirchner, Narrowing, abstraction and constraints for proving properties of reduction relations, in: H. Comon-Lundh, C. Kichner, H. Kirchner (Eds.), Rewriting, Computation and Proof. Essays Dedicated to Jean-Pierre Jouannaud on the Occasion of His 60th Birthday, Vol. 4600 of Lecture Notes in Computer Science, Springer-Verlag, 2007, pp. 44–67.

[14] B. Gramlich, Relating innermost, weak, uniform and modular termination of term rewriting systems, in: Proceedings of the 3rd International Conference on Logic Programming and Automated Reasoning, Vol. 624, Lecture Notes in Artificial Intelligence, St. Petersburg, Russia, 1992, pp. 285–296.

[15] B. Gramlich, On termination and confluence properties of disjoint and constructor-sharing conditional rewrite systems, Theoretical Computer Science 165 (1) (1996) 97–131.

[16] J. Goubault-Larrecq, A proof of weak termination of typed lambda-sigma-calculi, in: Proceedings of the Workshop Types for Proofs and Programs, Vol. 1512 of Lecture Notes in Computer Science, Springer-Verlag, Aussois, France, 1998, pp. 134–153.

[17] J. Goubault-Larrecq, Well-founded recursive relations, in: Proceedings of the 15th International Workshop in Computer Science Logic, Vol. 2142 of Lecture Notes in Computer Science, Springer-Verlag, Paris, France, 2001, pp. 484–498.

[18] G. Huet, J.-J. Lévy, Computations in orthogonal rewriting systems, I, in: J.-L. Lassez, G. Plotkin (Eds.), Computational Logic, The MIT press, 1991, Ch. 11, pp. 395–414.

[19] H. Comon, Inductionless induction, in: A. Robinson, A. Voronkov (Eds.), Handbook of Automated Reasoning, Vol. I, Elsevier Science, 2001, Ch. 14, pp. 913–962.

[20] G. Huet, J.-M. Hullot, Proofs by induction in equational theories with constructors, Journal of Computer and System Sciences 25 (2) (1982) 239–266,

preliminary version in Proceedings of the 21st Symposium on Foundations of Computer Science, IEEE, 1980.

[21] T. Nipkow, G. Weikum, A decidability result about sufficient completeness of axiomatically specified abstract data types, in: Proceedings of the 6th GI-Conference on Theoretical Computer Science, Vol. 145 of Lecture Notes in Computer Science, Springer-Verlag, Dortmund, Germany, 1982, pp. 257–267.

[22] E. Kounalis, Completeness in data type specifications, in: B. Buchberger (Ed.), Proceedings of the EUROCAL Conference, Vol. 204 of Lecture Notes in Computer Science, Springer-Verlag, Linz, Austria, 1985, pp. 348–362.

[23] D. Kapur, P. Narendran, H. Zhang, Proof by induction using test sets, in: J. Siekmann (Ed.), Proceedings of the 8th International Conference on Automated Deduction, Vol. 230 of Lecture Notes in Computer Science, Springer-Verlag, Oxford, UK, 1986, pp. 99–117.

[24] H. Comon, Sufficient completeness, term rewriting system and anti-unification, in: J. Siekmann (Ed.), Proceedings of the 8th International Conference on Automated Deduction, Vol. 230 of Lecture Notes in Computer Science, Springer-Verlag, Oxford, UK, 1986, pp. 128–140.

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

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

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

[28] A. Bouhoula, F. Jaquemard, Automatic verification of. sufficient completeness for. specifications of complex data structures, Research Report RR-LSV-05-17, INRIA (2005).

[29] A. Bouhoula, F. Jacquemard, Automatic Verification of Sufficient Completeness for Conditional Constrained Term Rewriting Systems, Research Report RR-5863, INRIA (2006).

[30] J. Hendrix & al., The Maude sufficient completeness checker.
URL `http://maude.cs.uiuc.edu/tools/scc/`

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

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

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

[34] A.-C. Caron, J.-L. Coquide, M. Dauchet, Encompassment properties and automata with constraints, in: C. Kirchner (Ed.), Proceedings of the 5th International Conference on Automated Deduction, Vol. 690 of Lecture Notes in Computer Science, Springer-Verlag, Montreal, Canada, 1993, pp. 328–342.

[35] H. Comon, F. Jacquemard, Ground reducibility is EXPTIME-complete, in: Proceedings of the 12th IEEE Symposium on Logic in Computer Science, IEEE Comp. Soc. Press, Warsaw, Poland, 1997, pp. 26–34.

[36] N. Dershowitz, J.-P. Jouannaud, Handbook of Theoretical Computer Science, Vol. B, Elsevier Science Publishers B. V. (North-Holland), 1990, Ch. 6: Rewrite Systems, pp. 244–320, also as: Research report 478, LRI.

[37] F. Baader, T. Nipkow, Term *Re*writing and *all That*, Cambridge University Press, New York, NY, USA, 1998.

[38] N. Dershowitz, D. A. Plaisted, Rewriting, in: A. Robinson, A. Voronkov (Eds.), Handbook of Automated Reasoning, Vol. I, Elsevier Science Publishers B. V. (North-Holland), 2001, Ch. 9, pp. 535–610.

[39] E. Barendsen, I. Bethke, J. Heering, R. Kennaway, P. Klint, V. van Oostrom, F. van Raamsdonk, F.-J. de Vries, H. Zantema, Term Rewriting Systems, Vol. 55 of Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 2003.

[40] J. B. Kruskal, Well-quasi ordering, the tree theorem and Vazsonyi's conjecture, Transactions of the American Mathematical Society 95 (1960) 210–225.

[41] I. Gnaedig, H. Kirchner, Rewriting on ground terms, HAL-INRIA Open Archive Number inria-00387058 (2009).
URL `http://hal.inria.fr/inria-00387058/en/`

[42] A. Middeldorp, E. Hamoen, Completeness results for basic narrowing, Applicable Algebra in Engineering, Communication and Computation 5 (3 & 4) (1994) 213–253.

[43] T. Arts, J. Giesl, Proving innermost normalisation automatically, in: Proceedings of the 8th Conference on Rewriting Techniques and Applications, Vol. 1232 of Lecture Notes in Computer Science, Springer-Verlag, Sitges, Spain, 1997, pp. 157–171.

[44] O. Fissore, I. Gnaedig, H. Kirchner, Proving weak termination also provides the right way to terminate - Extended version -, HAL-INRIA Open Archive Number inria-00099872 (2004).
URL `http://hal.inria.fr/inria-00099872/en/`

[45] I. Gnaedig, H. Kirchner, Computing constructor forms with non terminating rewrite programs - extended version -, HAL-INRIA Open Archive Number inria-00113146 (2006).
URL http://hal.inria.fr/inria-00113146/en/

[46] D. J. Dougherty, C. Kirchner, H. Kirchner, A. Santana de Oliveira, Modular access control via strategic rewriting, in: Proceedings of 12th European Symposium On Research In Computer Security, Dresden, 2007, pp. 578–593.

[47] J. Giesl, R. Thiemann, P. Schneider-Kamp, S. Falke, Improving dependency pairs, in: Proceedings of the 10th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Vol. 2850 of Lecture Notes in Artificial Intelligence, Springer-Verlag, Almaty, Kazakhstan, 2003, pp. 165–179.

Vitae

Isabelle Gnaedig obtained her Ph.D. in Computer Science at the University of Nancy, France and a researcher position at INRIA in 1986. In 1992, she joined the Protheo project focusing on constraints, automated deduction and software property proofs. She recently moved to the Carte project working on adversary computations. From 1987 to 1996, she was in charge of communication at INRIA Lorraine and at the Computer Science Research Center of Nancy.

Her domains of interest include automated deduction, models for programming languages and verification of dynamic systems. She has worked on completion and termination of rewriting with sorts and strategies and on new program proof techniques, in particular for completeness, weak and probabilistic termination. She recently began working on modeling and detection of computer viruses.

Hélène Kirchner has obtained her PhD in Computer Science in 1982 and her Habilitation (These d'Etat) in 1985. She entered CNRS in 1982 and became Research Director in 1995. After leading the Protheo project from 1997 to 2000, she took the Direction of the joint laboratory LORIA and of the INRIA Lorraine research center (2001-2007). Since 2007, she is Deputy Scientific Director at INRIA.

Her research is concerned with the design and development of safe software: formal specifications, logic and automated deduction, program verification, with a special emphasis on deduction and computation by rewriting and strategies. Since 2005, she applies these techniques to the design and verification of security policies, in particular access control policies, and to the modeling of bio-chemical processes with graph transformations.