Termination of Priority Rewriting

Isabelle Gnaedig

Termination of Priority Rewriting

Isabelle GNAEDIG
LORIA-INRIA
BP 239 F-54506 Vandffuvre-les-Nancy Cedex
France
Isabelle.Gnaedig@loria.fr

ABSTRACT

Introducing priorities on rules in rewriting increases their expressive power and helps to limit computations. Priority rewriting is used in rule-based programming as well as in functional programming. Termination of priority rewriting is then important to guarantee that programs give a result. We describe in this paper an inductive proof method for termination of priority rewriting, relying on an explicit induction on the termination property. It works by generating proof trees, modeling the rewriting relation by using abstraction and narrowing. As it specifically handles priorities on the rules, our technique allows proving termination of term rewrite systems that would diverge without priorities.

Categories and Subject Descriptors

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

General Terms

Algorithms, Languages, Verification

Keywords

Abstraction, Constraint, Narrowing, Priority, Termination

1. INTRODUCTION

In [3, 4], priority rewriting systems (PRS's in short) have been introduced. A PRS is a term rewrite system (TRS in short) with a partial ordering on rules, determining a priority between some of them. Considering priorities on the rewrite rules to be used can be very useful for an implementation purpose, to reduce the non-determinism of computations or to enable divergent systems to terminate, and for a semantical purpose, to increase the expressive power of rules. Priority rewriting is enabled in rule-based languages like ASF+SDF [23] or Maude [15]. It is also used as a computation model for functional programming [18], and is underlying in the functional strategy, used for example in Lazy ML [1], Clean [5], or Haskell [13]. Let us also cite recent works on specification and correctness of security policies using rewriting with priorities [7, 8].

But priority rewriting is delicate to handle. First, the priority rewriting relation is not always decidable, because a term rewrites with a given rule only if in the redex, there is no reduction leading to another redex, reducible with a rule of higher priority. A way to overcome the undecidability can be to force evaluation of the terms in reducing subterms to strong head normal form via some strategy [18], or to use the innermost strategy [16]. But in these cases, normalization can lead to non-termination.

Second, the semantics of a PRS is not always clearly defined. In [3], a semantics is proposed, relying on a notion of unique sound and complete set of closed instances of the rules of the PRS, and it is showed that bounded -the bounded property is weaker than termination- PRS's have a semantics. In [21], a fixed point based technique is proposed to compute the semantics of a PRS. It is also proved that for a bounded PRS with finitely many rules, the set of successors of any term is finite and computable. In [16], a logical semantics of PRS's, based on equational logic is given. A particular class of PRS's is proved sound and complete with respect to the initial algebra, provided every priority rewriting sequence from every ground term terminates.

Then the termination problem of the priority rewriting relation naturally arises, either to guarantee that it has a semantics, or to ensure that rewriting computations always give a result. Surprisingly, it seems not to have been much investigated until now. Let us cite [19], discussing a normalizing strategy of PRSs i.e., a strategy giving only finite derivations for terms having a normal form with usual rewriting, and [22], where it is proved that termination of innermost rewriting implies termination of generalized innermost rewriting with ordered rules. But to our knowledge,

the problem of finding a specific termination proof technique has only been specifically addressed in [16], where the use of reduction orderings is extended with an instantiation condition on rules linked with the priority order.

Our purpose here is to consider termination of priority rewriting from an operational point of view, with the concern of guaranteeing a result for every computation. So it seems interesting to focus on the innermost priority rewriting of [16], because it is decidable, easy to manipulate, and the innermost strategy is often used in programming contexts where priorities on rules are considered. The security policies given in [7] have indeed been executed in TOM [17] with an innermost evaluation mechanism.

Obviously, a PRS is terminating if the underlying TRS is. So usual rewriting termination proof techniques can be used for priority rewriting. Here, we propose to be finer in considering non (innermost) terminating TRS's, that become terminating using priorities on rules.

We use an inductive method, whose principle has already been applied for proving termination of rewriting (without priorities) under strategies [12]. The idea is to prove on the ground term algebra that every derivation starting from any term terminates, supposing that it is true for terms smaller than the starting terms for an induction ordering.

Priority rewriting is defined in Section 2. In Section 3, we present the inductive proof principle of our approach. Section 4 develops the basic concepts of the inductive proof mechanism relying on abstraction and narrowing, and the involved constraints. Section 5 presents the proof procedure and the related correctness and completeness theorem. Section 6 describes a variant of the previous procedure, dealing with the unsatisfiability of the involved constraints. We conclude in Section 7.

2. PRIORITY REWRITING

2.1 The background

We assume that the reader is familiar with the basic definitions and notations of term rewriting given for instance in [2, 9, 20]. $\mathcal{T}(\mathcal{F},\mathcal{X})$ is the set of terms built from a given finite set $\mathcal{F}$ of function symbols f having arity $n \in \mathbb{N}$, and a set $\mathcal{X}$ of variables denoted by $x, y \ldots$. $\mathcal{T}(\mathcal{F})$ is the set of ground terms (without variables). The terms reduced to a symbol 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 said to be a (strict) suffix position of p if $p' = p\lambda$, where λ is a (non-empty) sequence of integers. For a position p of a term t, we denote by $t|_p$ the subterm of t at position p, and by $t[s]_p$ the term obtained by replacing with s the subterm at position p in t.

A substitution is an assignment from $\mathcal{X}$ to $\mathcal{T}(\mathcal{F},\mathcal{X})$, written $\sigma = (x = t, \ldots, y = u)$. 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 by $Dom(\sigma)$ is the finite subset of $\mathcal{X}$ such that $\sigma x \neq x$. The range of σ, denoted by $Ran(\sigma)$, is defined by $Ran(\sigma) = \bigcup_{x \in Dom(\sigma)} Var(\sigma x)$. An instantiation or ground substitution is an assignment from $\mathcal{X}$ to $\mathcal{T}(\mathcal{F})$. Id denotes the identity substitution. The composition of substitutions σ_1 followed by σ_2 is denoted by $\sigma_2\sigma_1$. Given a subset $\mathcal{X}_1$ of $\mathcal{X}$, we write $\sigma_{\mathcal{X}_1}$ for the *restriction* of σ to the variables of $\mathcal{X}_1$ i.e., the substitution such that $Dom(\sigma_{\mathcal{X}_1}) \subseteq \mathcal{X}_1$ and $\forall x \in Dom(\sigma_{\mathcal{X}_1}) : \sigma_{\mathcal{X}_1} x = \sigma x$.

A set $\mathcal{R}$ of (term) rewrite rules or term rewrite system on $\mathcal{T}(\mathcal{F},\mathcal{X})$ is a set of pairs of terms of $\mathcal{T}(\mathcal{F},\mathcal{X})$, denoted by $l \to r$, such that $l \notin \mathcal{X}$ and $Var(r) \subseteq Var(l)$. Given a term rewrite system $\mathcal{R}$, a function symbol in $\mathcal{F}$ is called a *constructor* iff it does not occur in $\mathcal{R}$ at the top position of a left-hand side of rule, and is called a *defined function symbol* otherwise. The set of defined function symbols is denoted by $\mathcal{D}_R$ ($\mathcal{R}$ is omitted when there is no ambiguity). In this paper, we only consider finite sets of function symbols and 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 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 innermost rewriting relation consists of always rewriting at the lowest possible positions.

Let $\mathcal{R}$ be a term 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 by $t \leadsto^{\mathcal{R}}_{p,l\to r,\sigma} t'$ where 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 $Var(l) \cap Var(t) = \emptyset$.

An ordering $\succ$ on $\mathcal{T}(\mathcal{F},\mathcal{X})$ is said to be noetherian iff there is no infinitely 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 $\mathcal{T}(\mathcal{F},\mathcal{X})$, $f(\ldots t \ldots) \succ t$.

For $\mathcal{F}$ and $\mathcal{X}$ finite, if $\succ$ is monotone and has the subterm property, then it is noetherian [14]. If, in addition, $\succ$ is stable under substitution (for any substitution σ, any pair of terms $t, t' \in \mathcal{T}(\mathcal{F},\mathcal{X})$, $t \succ t'$ implies $\sigma t \succ \sigma t'$), then it is called a simplification ordering.

2.2 Priority rewriting

A priority term rewrite system (PRS in short) is a pair $(\mathcal{R}, \blacktriangleright)$ of an underlying rewrite system $\mathcal{R}$ (always considered as finite in this paper) and a partial ordering $\blacktriangleright$ on the rules of $\mathcal{R}$. A rule r_1 has a higher priority than a rule r_2 iff $r_1 \blacktriangleright r_2$, which is also written $\downarrow^{r_1}_{r_2}$.

DEFINITION 1. *[16] Let $\mathcal{R}$ be a PRS on $\mathcal{T}(\mathcal{F},\mathcal{X})$. A term s is IP-reducible and (IP-) rewrites to t at position p with the rule $l \to r$, and the substitution σ which is written $s \to^{IP}_{p,l\to r,\sigma} t$ iff:*

- $s \to_{p,l\to r,\sigma} t$
- *no proper subterm of the redex $s|_p$ is IP-reducible*
- *$s|_p$ is not IP-reducible by any rule in $\mathcal{R}$ of higher priority than $l \to r$.*

Example: With the rewrite system

$$\begin{array}{ll} f(g(x)) \to b & (1) \\ g(a) \to c & (2) \\ g(a) \to d & (3) \end{array}$$

$\downarrow$

on the term $f(g(a))$, Rule (1) should apply, but this would not be an innermost rewriting step. So Rule (2) applies, but Rule (3) does not, because (2) $\blacktriangleright$ (3).

A PRS $\mathcal{R}$ IP-terminates if and only if every IP-derivation of the rewriting relation induced by $\mathcal{R}$ is finite. If $t \stackrel{*}{\rightarrow}^{IP} t'$ with t' IP-irreducible, then t' is called a(n) (IP-) normal form of t and denoted by $t{\downarrow}$. Note that given t, $t{\downarrow}$ may be not unique.

3. INDUCTIVELY PROVING TERMINATION OF IP-REWRITING

We prove termination of IP-rewriting by induction on the ground terms. Working on ground terms is appropriate, since most of the time, the algebraic semantics of rule-based languages is initial. Moreover, in [16], to guarantee stability by substitution of the innermost rewriting relation, the rules without highest priority only can reduce ground terms. Finally, there are TRS's which are non-innermost terminating on $\mathcal{T}(\mathcal{F},\mathcal{X})$ and innermost terminating on $\mathcal{T}(\mathcal{F})$. A termination proof method working on $\mathcal{T}(\mathcal{F},\mathcal{X})$ could not handle them.

For proving that a priority term rewrite system on $\mathcal{T}(\mathcal{F})$ IP-terminates, we reason with a local notion of termination on terms: a term t of $\mathcal{T}(\mathcal{F})$ is said to be IP-terminating for a PRS $\mathcal{R}$ if every IP-rewriting chain (or derivation) starting from t is finite.

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

Rewriting derivations are simulated, using a lifting mechanism, by a proof tree developed from $g(x_1,\ldots,x_m)$ on $\mathcal{T}(\mathcal{F},\mathcal{X})$, for every $g \in \mathcal{D}$, by alternatively using narrowing and an abstraction mechanism. Narrowing schematizes the rewriting possibilities of terms, abstraction simulates the reduction of subterms in the derivations until these subterms become normal forms. It expresses the application of the induction hypothesis on these subterms, for which, as they are supposed to be IP-terminating, a normal form exists.

The nodes of the developed proof trees are composed of a current term of $\mathcal{T}(\mathcal{F},\mathcal{X})$, and a constraint composed of two kinds of formulas: ordering constraints, set to guarantee the validity of the inductive steps, and abstraction constraints combined to narrowing substitutions. Each node in a proof tree schematizes the set of ground instances of the current term, which are solutions of the abstraction constraint.

For a term t of $\mathcal{T}(\mathcal{F},\mathcal{X})$ occurring in a proof tree issued from an initial pattern $t_{ref} = g(x_1,\ldots,x_m)$,

- first, some subterms $\theta t|_j$ of θt are supposed to be IP-terminating for every θ solution of the constraint associated to t, by the induction hypothesis, if $\theta t_{ref} \succ \theta t|_j$ for the induction ordering $\succ$. So the $t|_j$ are replaced in t by *abstraction variables* X_j representing respectively any of their normal forms. Reasoning by induction allows us to only suppose the existence of the normal forms *without explicitly computing them.* If the ground instances of the resulting term are IP-terminating (either if the induction hypothesis can be applied to them, or if they can be proved IP-terminating by other means, which we will present later), then the ground instances of the initial term are IP-terminating. Otherwise,
- the resulting term $u = t[X_j]_{\{i_1,\ldots,i_p\}}$ (where $i_1,\ldots,i_p$ are the abstraction positions in t) is narrowed in all possible ways into terms v, with an appropriate narrowing relation corresponding to IP-rewriting u according to the possible instances of the X_j.

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

This technique was inspired from the one we proposed for proving the innermost termination of classical rewrite systems in [12]. We now give the concepts needed to formalize and automate it.

4. ABSTRACTION, NARROWING, CONSTRAINTS

The induction ordering is constrained along the proof by inequalities between terms that must be comparable, each time the induction hypothesis is used in the abstraction mechanism.

This ordering is not defined a priori, but just has to verify inequalities of the form $t > u_1,\ldots,u_m$, accumulated along the proof, and which are called *ordering constraints.* Thus, for establishing the inductive termination proof, it is sufficient to decide whether ordering constraints are satisfiable.

DEFINITION 2 (ORDERING CONSTRAINT). An ordering constraint *is a pair of terms of* $\mathcal{T}(\mathcal{F},\mathcal{X})$ *denoted by* $(t > t')$. *It is said to be* satisfiable *if 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* $\theta t \succ \theta t'$. *We say that* $\succ$ *satisfies* $(t > t')$.

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

Satisfiability of a constraint conjunction C of this form is undecidable. But a sufficient condition for an ordering $\succ_{\mathcal{P}}$ on $\mathcal{T}(\mathcal{F},\mathcal{X})$ to satisfy C is that $t \succ_{\mathcal{P}} t'$ for every constraint $t > t'$ of C, and $\succ_{\mathcal{P}}$ is stable under substitution.

Simplification orderings fulfill such a condition. So, in practice, it is sufficient to find a simplification ordering $\succ_{\mathcal{P}}$ such that $t \succ_{\mathcal{P}} t'$ for every constraint $t > t'$ of C.

The ordering $\succ_{\mathcal{P}}$, defined on $\mathcal{T}(\mathcal{F},\mathcal{X})$, can then be seen as an extension of the induction ordering $\succ$ on $\mathcal{T}(\mathcal{F})$. For convenience sake, $\succ_{\mathcal{P}}$ will also be written $\succ$.

Solving ordering constraints in finding simplification orderings is a well-known problem. The simplest way and an automatable way to proceed is to test simple existing orderings like the subterm ordering, the Recursive Path Ordering,

or the Lexicographic Path Ordering. This is often sufficient for the constraints considered here: thanks to the power of induction, they are often simpler than for termination methods directly using ordering for orienting rewrite rules.

If these simple orderings are not powerful enough, automatic constraint solvers like C*ime* [1] can provide adequate polynomial orderings.

4.1 Abstraction

Let us define the abstraction variables more formally.

DEFINITION 3. *Let $\mathcal{N}$ be a set of variables disjoint from $\mathcal{X}$. Symbols of $\mathcal{N}$ are called* abstraction variables. *Substitutions and instantiations are extended to $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$ in the following way: for any substitution σ (resp. instantiation θ) such that $Dom(\sigma)$ (resp. $Dom(\theta)$) contains a variable $X \in \mathcal{N}$, σX (resp. θX) is in IP-normal form.*

DEFINITION 4 (TERM ABSTRACTION). *The term $t[t|_j]_{j\in\{i_1,\ldots,i_p\}}$ is said to be* abstracted *into the term u (called* abstraction *of t) at positions $\{i_1, \ldots, i_p\}$ iff*

$$u = t[X_j]_{j\in\{i_1,\ldots,i_p\}},$$

where the $X_j, j \in \{i_1, \ldots, i_p\}$ are fresh distinct abstraction variables.

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

4.2 Narrowing

After abstracting the current term t into $t[X_j]_{j\in\{i_1,\ldots,i_p\}}$, we test whether the possible ground instances of $t[X_j]_{j\in\{i_1,\ldots,i_p\}}$ are reducible, according to the possible values of the instances of the X_j. This is achieved by innermost narrowing $t[X_j]_{j\in\{i_1,\ldots,i_p\}}$, with the priority rewrite system.

In a first time, to schematize innermost rewriting on ground terms, we need to refine the usual notion of narrowing. In fact, with the usual innermost narrowing relation, if a position p in a term t is a narrowing position, no suffix position of p can be a narrowing position as well. However, if we consider ground instances of t, we can have rewriting positions p for some instances, and p' for other instances, such that p' is a suffix position of p. So, when using the narrowing relation to schematize innermost rewriting of ground instances of t, the narrowing positions p to consider depend on a set of ground instances of t, which is defined by excluding the ground instances of t that would be narrowable at some suffix position of p. For instance, with the TRS $R = \{g(a) \to a, f(g(x)) \to b\}$, the innermost narrowing positions of the term $f(g(X))$ are 1 with the narrowing substitution $\sigma = (X = a)$, and ϵ with any σ such that $\sigma X \neq a$. This leads us to introduce constrained substitutions.

[1] Available at http://cime.lri.fr/

Let σ be a substitution on $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$. In the following, we identify $\sigma = (x_1 = t_1, \ldots, x_n = t_n)$ with the equality formula $\bigwedge_i (x_i = t_i)$, with $x_i \in \mathcal{X} \cup \mathcal{N}$, $t_i \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$. Similarly, we call *negation* $\overline{\sigma}$ of the substitution σ the formula $\bigvee_i (x_i \neq t_i)$. The negation of Id means that no substitution can be applied.

DEFINITION 5. *A substitution σ is said* to satisfy *a constraint $\bigwedge_j \bigvee_{i_j} (x_{i_j} \neq t_{i_j})$, iff for every ground instantiation θ, $\bigwedge_j \bigvee_{i_j} (\theta\sigma x_{i_j} \neq \theta\sigma t_{i_j})$. A* constrained substitution *σ is a formula $\sigma_0 \wedge \bigwedge_j \bigvee_{i_j} (x_{i_j} \neq t_{i_j})$, where σ_0 is a substitution, and $\bigwedge_j \bigvee_{i_j} (x_{i_j} \neq t_{i_j})$ the constraint to be satisfied by σ_0.*

DEFINITION 6 (INNERMOST NARROWING [12]). *A term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$* innermost narrows *into a term $t' \in$ 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*

$$t \leadsto^{Inn}_{p,l\to r,\sigma} t'$$

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

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

Now, we have to see how to simulate the IP-rewriting steps of a given term following the possible instances of its variables, by narrowing it with the rules, considering their priority. Unlike for simulating rewriting without priorities, where the narrowing process only depends on the term to be rewritten and of the rule considered, simulating IP-rewriting of ground instances of a term with a given rule requires to consider the narrowing steps with the rules having a higher priority. This also requires to use negations of substitutions. Let us consider the following example:

$$\left\downarrow \begin{array}{l} f(g(x), y) \to a \\ f(x, h(y)) \to b \\ f(x, y) \to c. \end{array}\right.$$

The term $f(x, y)$ innermost narrows into a with the first rule and the most general unifier $(x = g(x'))$, into b with the second rule, the most general unifier $(y = h(y'))$ and the constraint $x \neq g(x')$, and finally into c with the third rule, the most general unifier equal to Id and the constraint $x \neq g(x') \wedge y \neq h(y')$. So, applying the rules one after the other, with the current narrowing most general unifier we have to accumulate the negation of the most general unifiers of the previous constrained substitutions, ignoring their constraint part.

If the narrowing substitutions $\sigma_0 \wedge \bigwedge_{j\in[1..k]} \overline{\sigma_j}$ of the previous rules have a constraint part coming from the innermost mechanism of Definition 6, this constraint part is also ignored by the priority mechanism. Indeed, the constraint part is defined from σ_0, and has no meaning for the negation of σ_0. With the PRS:

$$\left\downarrow \begin{array}{l} f(g(h(x))) \to a \\ h(a) \to b \\ f(g(x)) \to c \end{array}\right.$$

the term $f(x)$ innermost narrows with the first rule and $\sigma_1 = (x = g(h(x')) \wedge x' \neq a)$, the second rule does not apply, and the third rule applies with $\sigma_3 = (x = g(x'') \wedge x \neq g(h(x')))$.

Also, if the constraint part of a substitution is due to the priority mechanism, the negation of this substitution by the innermost mechanism also only considers the most general unifier of the substitution. With the PRS:

$$\begin{array}{l} \downarrow \; f(g(h(x,y)),z) \to a \\ \downarrow \; f(x,y) \to b \\ \downarrow \; h(a,x) \to a \\ \downarrow \; h(x,b) \to b \end{array}$$

the term $f(x,y)$ innermost narrows into a with the first rule and the constrained substitution $(x = g(h(x',y')) \wedge x' \neq a \wedge y' \neq b)$, because $h(x',y')$ narrows with the third rule and the substitution $(x' = a)$, and with the fourth rule and the substitution $(y' = b \wedge x' \neq a)$.

The term $f(x,y)$ also innermost narrows into b with the second rule and the substitution $(Id \wedge x \neq g(h(x',y')))$.

DEFINITION 7 (INNER. PRIOR. NARROWING). *Let $\mathcal{R}$ be a priority term rewrite system. A term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$ IP-*narrows *into $t' \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$ at the non-variable position p of t, using the rule $l \to r \in \mathcal{R}$ with the constrained substitution $\sigma = \sigma_0 \wedge \bigwedge_{j \in [1..k]} \overline{\sigma_j} \bigwedge_{i \in [1..n]} \overline{\sigma_0^i}$, which is written*

$$t \leadsto^{IP}_{p,l \to r,\sigma} t'$$

iff $t' = \sigma_0(t[r]_p)$, where σ_0 is the most general unifier of $t|_p$ and l, $\sigma_j, j \in [1..k]$ are all most general unifiers of $\sigma_0 t|_{p'}$ and a left-hand side l' of a rule of $\mathcal{R}$, for all suffix position p' of p in t, and $\sigma_0^1, \ldots, \sigma_0^n$ are the most general unifiers of $t|_p$ with the left-hand sides of the rules having a greater priority than $l \to r$.

4.3 Accumulating 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 does not satisfy the abstraction 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.

DEFINITION 8. *An* abstraction constraint formula *(ACF in short) is a formula $\bigwedge_i (t_i\downarrow = t'_i) \wedge \bigwedge_j (x_j = u_j)$, where $x_j \in \mathcal{X} \cup \mathcal{N}$, $t_i, t'_i, u_j, \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$.*

DEFINITION 9. *An abstraction constraint formula $A = \bigwedge_i (t_i\downarrow = t'_i) \wedge \bigwedge_j (x_j = u_j)$ is* satisfiable *iff there is at least one instantiation θ such that $\bigwedge_i (\theta t_i\downarrow = \theta t'_i) \wedge \bigwedge_j (\theta x_j = \theta u_j)$. The instantiation θ is then said to satisfy the ACF A and is called solution of A.*

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

The satisfiability of A is in general undecidable, but it is often easy in practice to exhibit an instantiation satisfying it: most of the time, solutions built on constructor terms can be synthesized in an automatic way. Other automatable sufficient conditions, relying in particular on the characterization of normal forms, are also under study. The unsatisfiability of A is also undecidable in general, but here also, simple automatable sufficient conditions can be used [12], as to test whether A contains equalities $t\downarrow = u$, where u is reducible. In the following section, we present the procedure exactly simulating the rewriting trees i.e., dealing with the satisfiability of A. In the next section, we present the alternative approach dealing with the unsatisfiability.

5. THE IP-TERMINATION PROCEDURE

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

- The first rule abstracts the current term t at given positions $i_1, \ldots, i_p$ into $t[X_j]_{j \in \{i_1,\ldots,i_p\}}$.
  The constraint $\bigwedge_{j \in \{i_1,\ldots,i_p\}} t_{ref} > t|_j$ is set in C. The abstraction constraint $\bigwedge_{j \in \{i_1,\ldots,i_p\}} t|_j\downarrow = X_j$ 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, because $C \wedge \bigwedge_{j \in \{i_1,\ldots,i_p\}} t_{ref} > t|_j$ cannot be proved satisfiable, 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 non-narrowable subterms of $\mathcal{T}(\mathcal{F}, \mathcal{N})$. Indeed, by Definition 3, every ground instance of such subterms is in IP-normal form.

- The second rule narrows the resulting term u in all possible ways in one step, with all possible rewrite rules of the rewrite system $\mathcal{R}$, and all possible substitutions, into terms $v_1, \ldots, v_q$, according to Definition 7. This step is a branching step, creating as many states as there are narrowing possibilities. The substitution σ is integrated to A. If $A \wedge \sigma$ is not satisfiable, the narrowing step with σ is meaningless: it does not represent any rewriting step for the ground instances of u. So it can be discarded. This is the **Narrow** rule.

- We finally have a **Stop** rule halting the proof process on the current branch of the proof tree, when the

Table 1: Inference rules for IP-temination

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

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

Narrow: $\frac{\{t\},\ A,\ C}{\{v_i\},\ A\wedge\sigma,\ C}$ if $t \leadsto^{IP}_{\sigma} v_i$ and $A\wedge\sigma$ is satisfiable

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

$$H_A(t) = \begin{cases} \top & \text{if } t \text{ is in } \mathcal{T}(\mathcal{F},\mathcal{N}) \\ & \text{and is not narrowable} \\ t\downarrow = X & \text{otherwise.} \end{cases} \qquad H_C(t) = \begin{cases} \top & \text{if } IPT(t) \\ t_{ref} > t & \text{otherwise.} \end{cases}$$

ground instances of the current term can be stated as IP-terminating. This happens when the whole current term u can be abstracted i.e., when $C \wedge t_{ref} > u$ is satisfiable.

Let us note that the inductive reasoning can be completed as follows. When the induction hypothesis cannot be applied to a term u, it may be possible to prove IP-termination of every ground instance of u in another way. Let $IPT(u)$ be a predicate that is true iff every ground instance of u is IP-terminating. In the first and third inference rules, we then associate the alternative predicate $IPT(u)$ to the condition $t > u$.

To establish $IPT(u)$, decidable sufficient conditions exist, applicable in practice, because the predicate is only considered for particular terms introduced along the proof, and not for any term.

In particular, $IPT(u)$ is true when every instance of u is in normal form. This is the case when u is in $\mathcal{T}(\mathcal{F},\mathcal{N})$ and is not narrowable. This includes the cases where u itself is an abstraction variable, and where u is a non-narrowable ground term.

We also have $IPT(u)$ for narrowable terms $u \in \mathcal{T}(\mathcal{F},\mathcal{N})$ whose narrowing substitutions are not compatible with A i.e., such that $A\wedge\sigma$ is not satisfiable. As said just before Definition 8, these narrowing possibilities do not represent any reduction step for the ground instances of u, which are then irreducible.

Otherwise, to establish $IPT(u)$, we can use the notion of usable rule, as in [12].

The inference rules are given in Table 1. They use an initial patten $t_{ref} = g(x_1,\ldots,x_m)$, where $x_1,\ldots,x_m \in \mathcal{X}$ and $g \in \mathcal{D}$ (if g is a constant, then $t_{ref} = g$).

We generate the proof trees of $\mathcal{R}$ by applying, for each defined symbol $g \in \mathcal{D}$, the inference rules on the initial set of 3-tuples $\{(\{t_{ref} = g(x_1,\ldots,x_m)\}, \top, \top)\}$, with a specific strategy S, repeating the following steps: first, apply **Abstract**, and then try **Stop**. Then try all possible applications of **Narrow**. Then, try **Stop** again.

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

The process may not terminate if there is an infinite number of applications of **Abstract** and **Narrow** on the same branch of a proof tree. It may stop on the rule **Abstract** when the ordering constraints cannot be proved satisfiable, on the rule **Narrow** when the abstraction constraints cannot be proved satisfiable. Nothing can be said in these cases about IP-termination. The good case is when all branches of the proof trees end with an application of **Stop**: then IP-termination is established.

A finite proof tree is said to be *successful* if its leaves are states of the form $(\emptyset, A, C)$. We write $SUCCESS(g,\succ)$ if the application of S on $(\{g(x_1,\ldots,x_m)\},\top,\top)$ gives a successful proof tree, whose sets C of ordering constraints are satisfied by the same ordering $\succ$.

THEOREM 1. *Let $\mathcal{R}$ be a priority term rewrite system on $\mathcal{T}(\mathcal{F},\mathcal{X})$ having at least one constructor constant. Every term of $\mathcal{T}(\mathcal{F})$ is IP-terminating iff there is a noetherian ordering $\succ$ such that for each symbol $g \in \mathcal{D}$, we have $SUCCESS(g,\succ)$.*

Note that because it is the induction relation, the ordering $\succ$ has to be the same for all proof trees.

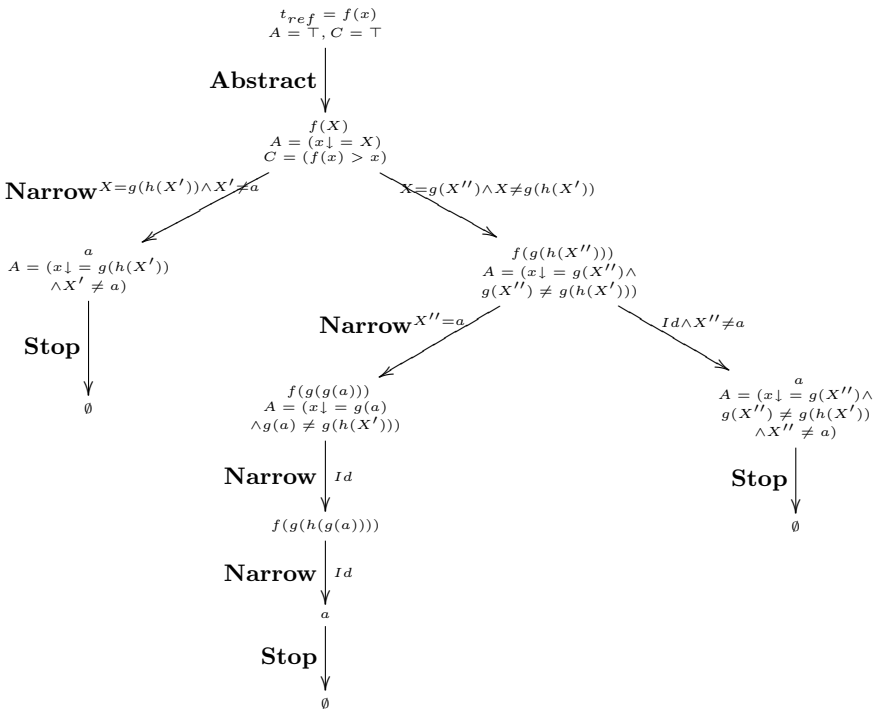
Example: Let us consider the PRS

$$\left\downarrow \begin{array}{l} f(g(h(x))) \to a \\ h(a) \to g(a) \\ f(g(x)) \to f(g(h(x))) \end{array}\right.$$

whose underlying TRS is neither terminating, nor innermost terminating. Theorem 1 is applied to prove that it is IP-terminating.

For a better readability of the proof, whenever we have $A\wedge\sigma$, we propagate σ into A, by applying the substitution part of σ to A. Moreover, the sets A and C are not repeated on a branch, when they do not change. The proof tree of f is given in Figure 1.

Figure 1: Proof tree for symbol f



Abstract applies on $f(x)$ because the ordering constraint $f(x) > x$ is satisfiable by any noetherian ordering having the subterm property. Then, **Narrow** applies on $f(X)$ using the first and third rules, according to Definition 7.

On the second branch, the term $f(g(h(X'')))$ narrows into $f(g(g(a)))$ with the second rule, and $\sigma = (X'' = a)$, into a with the first rule and $\sigma = (Id \wedge X'' \neq a)$, but does not narrow with the third rule: the negation of Id does not exist.

The set A after the **Abstract** step is trivially satisfied by the instantiation $\theta = (x = X = a)$. One can take $\theta = (x = g(h(g(a))), X' = g(a))$ for the next set A on the first branch, $\theta = (x = g(a), X = X' = X'' = a)$ for the next set A on the second branch, and $\theta = (x = g(a), X = X' = a)$ for the last set A on the second branch.

In the proof tree of h, we just have an **Abstract**, a **Narrow** and a **Stop** step. The ordering constraints are satisfied by the same noetherian ordering than above. Applying Theorem 1, we conclude that the PRS is IP-terminating.

Example: Consider now the following IP-terminating specification of the *or* operator, whose underlying RS is also neither terminating nor innermost terminating, because of the commutativity rule:

$$\left\downarrow \begin{array}{ll} or(0, y) \to y & (1) \\ or(x, 1) \to 1 & (2) \\ or(x, y) \to or(y, x) & (3). \end{array}\right.$$

The proof tree of *or* is given in Figure 2. The **Stop** rule applies on $Y \in \mathcal{N}$ because, by definition of an abstraction variable, we have $IPT(Y)$. **Stop** applies on $or(Y, X)$ because the term would only be narrowable with the substitution $\sigma = (Id \wedge Y \neq 0 \wedge X \neq 1)$. But $A \wedge \sigma = (x\downarrow = X, y\downarrow = Y) \wedge X \neq 0 \wedge Y \neq 1 \wedge Y \neq 0 \wedge X \neq 1)$ would be unsatisfiable: X and Y could only be of the form $or(X', Y')$, which is impossible since $or(X', Y')$ is always reducible.

The set A after the **Abstract** step is trivially satisfied by the instantiation $\theta = (x = X = y = Y = 0)$. One can take $\theta = (x = y = Y = 0)$ for the next set A on the first branch, $\theta = (x = X = y = 1)$ for the set A on the second branch, and $\theta = (x = X = 1, y = Y = 0)$ for the set A on the third branch.

Note that in the two examples above, the irreducible constants of the algebra, and more generally the constructors, can be used in an automatic way to find a solution of A.

6. DEALING WITH THE UNSATISFIABILITY OF A

We present in this section an alternative approach to our procedure, dealing with the unsatisfiability of A instead of the satisfiability. As said in Section 4.3, instead of testing whether each node generated in the proof tree is relevant i.e., whether A is satisfiable, we test whether we have generated irrelevant nodes.

As explained in Section 5, the satisfiability test of A in the inference rules of our procedure (see Table 1) is given in the condition of **Narrow**. The unsatisfiability test only localizes in the condition of **Stop**, because the underlying idea is to stop the proof process on a branch whose current node is detected to be irrelevant. It is given as an alternative to the initial condition of **Stop**. The new conditions of the inference rules are given in Table 2.

Figure 2: Proof tree for symbol *or*

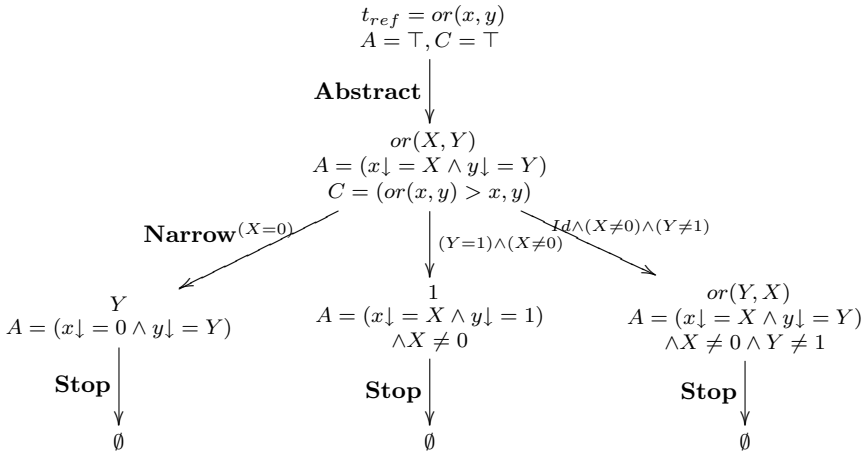


Table 2: Conditions for inference rules dealing with the unsatisfiability of A

$COND{-}ABSTRACT:$	$C \wedge H_C(t\|_{i_1}) \ldots \wedge H_C(\|t_{i_p})$ is satisfiable
$COND{-}NARROW:$	$t \rightsquigarrow_\sigma^{IP} v_i$
$COND{-}STOP:$	$(C \wedge H_C(t))$ is satisfiable or A is unsatisfiable.

The following automatable sufficient conditions for the unsatifiability of an abstraction constraint $t\downarrow = t'$ are often applicable in practice.

Case 1: $t\downarrow = t'$, with t' reducible. Indeed, in this case, any ground instance of t' is reducible, and hence cannot be a normal form.

Case 2: $t\downarrow = t' \wedge \ldots \wedge t'\downarrow = t''$, with t' and t'' not unifiable. Indeed, any ground substitution θ satisfying the above conjunction is such that (1) $\theta t\downarrow = \theta t'$ and (2) $\theta t'\downarrow = \theta t''$. In particular, (1) implies that $\theta t'$ is in normal form and hence (2) imposes $\theta t' = \theta t''$, which is impossible if t' and t'' are not unifiable.

Case 3: $t\downarrow = t'$ where $top(t)$ is a constructor, and $top(t) \neq top(t')$. Indeed, if the top symbol of t is a constructor c, then any normal form of any ground instance of t is of the form $c(u)$, where u is a ground term in normal form. The above constraint is therefore unsatisfiable if the top symbol of t' is g, for some $g \neq c$.

Case 4: $t\downarrow = t'$ with $t, t' \in \mathcal{T}(\mathcal{F}, \mathcal{X}_A)$ not unifiable and $\bigwedge_{t \rightsquigarrow_S v} v\downarrow = t'$ unsatisfiable. This criterion is of interest if the unsatisfiability of each conjunct $v\downarrow = t'$ can be shown with one of the four criteria we present here.

As these four conditions only work on the equality part of A, dealing with the unsatisfiability of A instead of the satisfiability is of particular interest when the abstraction formula A involves many negations of substitutions. The satisfiability test instead requires to verify that the solutions of the equational part verify the disequality part of A.

The unsatisfiability test is precisely advantageous for a succession of priority rules involving more than two or three rules, since narrowing with the nth rule requires to accumulate the negation of the narrowing substitutions of the $n-1$ previous ones.

Moreover, since the unsatisfiability test is an alternative condition of **Stop**, dealing with the unsatisfiability of A instead of the satisfiability is obviously interesting when **Stop** applies with the first condition ($(C \wedge H_C(t))$ is satisfiable). Analyzing A can then be completely avoided. This case is illustrated on the example we give now.

As said in the introduction, rewriting-based specifications with priorities on rules have recently been used to specify security policies, with a concern of verification of consistency, termination and completeness. The example we give below has been proposed in [7], for a conference management system described in [10]. Its termination, due to priority arguments, could not be formally proved until now.

Example: If we do not consider priorities, the following rewrite system is divergent, because of the eighth rule. Let us prove that it is IP-terminating.

$$
\begin{array}{l}
aut(q(author(x), SP, pap(x, z)), SUBMIS, u) \\
\quad \rightarrow PERMIT \\
aut(q(author(x), SP, pap(x, z)), v, u) \\
\quad \rightarrow DENY \\
aut(q(author(x), RSC, pap(x, z)), v, u) \\
\quad \rightarrow DENY \\
aut(q(revr(x), w, p), v, conf(x, p)) \\
\quad \rightarrow DENY \\
aut(q(revr(x), SR, pap(y, z)), REV, ass(x, pap(y, z))) \\
\quad \rightarrow PERMIT \\
aut(q(revr(x), SR, pap(y, z)), v, ass(x, pa(y, z))) \\
\quad \rightarrow DENY \\
aut(q(revr(x), RSC, pap(y, z)), MEE, ass(x, pap(y, z))) \\
\quad \rightarrow PERMIT \\
aut(q(revr(x), w, pap(x, z)), v, u) \\
\quad \rightarrow aut(q(revr(x), w, pap(x, z)), v, conf(x, pap(x, z))) \\
aut(x, y, z) \rightarrow NAPPLIC\ .
\end{array}
$$

Let us apply the strategy S on the initial pattern $t_{ref} = aut(x, y, z)$. The proof tree is given below.

We first have an **Abstract** step. Then, a **Narrow** step gives 9 branches, following the 9 rewrite rules. With the constrained narrowing substitutions $\sigma_1, (\sigma_2 \wedge \overline{\sigma_1}), \ldots (\sigma_7 \wedge \overline{\sigma_1} \wedge \ldots \wedge \overline{\sigma_6})$, the first seven ones give respectively the states $PERMIT$, $DENY$, $DENY$, $DENY$, $PERMIT$, $DENY$, $PERMIT$, on which **Stop** then applies. Indeed, we have $IPT(PERMIT)$ and $IPT(\ DENY)$ because $PERMIT$ and $DENY$ are irreducible constants. The ninth branch gives the state $NAPPLIC$, on which **Stop** applies too.

The interesting branch is the eighth one, giving the state $aut(q(revr(X'), w, pap(X', Z')),\ Y,\ conf(X', pap(X', Z')))$ with the substitution $\sigma_8 = (X = q(revr(X'), w, pap(X', Z')))$ constrained by $\overline{\sigma_1} \wedge \ldots \overline{\sigma_7}$. To lighten the figure, we only specify this branch in the proof tree.

$$
\begin{array}{c}
t_{ref} = aut(x, y, z) \\
A = \top, C = \top \\
\downarrow \ \textbf{Abstract} \\
aut(X, Y, Z) \\
A = (x\downarrow = X, y\downarrow = Y, z\downarrow = Z) \\
C = (aut(x, y, z) > x, y, z) \\
\downarrow \ \textbf{Narrow}_{Rule\ 8} \ \sigma_8 \wedge \overline{\sigma_1} \ldots \overline{\sigma_7} \\
aut(q(revr(X'), w, pap(X', Z')), Y, \\
conf(X', pap(X', Z')))
\end{array}
$$

$$
\begin{array}{c}
A = (x\downarrow = X, y\downarrow = Y, z\downarrow = Z \\
\wedge X = q(revr(X'), w, pap(X', Z')) \\
\wedge \overline{\sigma_1} \ldots \overline{\sigma_7}) \\
\downarrow \ \textbf{Narrow}_{Rule\ 4} \ \sigma_{10} = Id \\
DENY \\
\downarrow \ \textbf{Stop} \\
\emptyset
\end{array}
$$

From this last state, we still apply **Narrow**, with two narrowing possibilities: one, with the fourth rule and the narrowing substitution $\sigma_{10} = Id$, gives the state $DENY$, on which **Stop** then applies, because we have $IPT(DENY)$. The other one, using the ninth rule, has also Id as narrowing substitution, but once constrained by $\overline{\sigma_{10}} = \overline{Id}$, the substitution becomes empty, so this second narrowing possibility is not valid.

Applying the inference rules dealing with the satisfiability of A would have required to perform the satisfiability test for the nine branches of the first **Narrow** step, and on the branch of the second **Narrow** step of the eighth branch, which is avoided here.

As one can see, the rule **Stop** applies on all branches of the proof tree thanks to the predicate IPT. So, on this example, we do not even need to consider A.

As on the examples of Section 6, to satisfy the ordering constraints, any simplification ordering holds. So this example can also be treated in a completely automatic way.

7. CONCLUSION

In this paper, we have proposed an inductive method for proving termination of the decidable innermost priority rewriting relation of C.K. Mohan [16]. This work is an extension to priority rewriting of an inductive approach given in [12] for proving innermost termination of rewriting.

The priority mechanism localizes in the narrowing relation used to model the rewriting relation. We then have generalized the innermost narrowing relation introduced in [12], to model the IP-rewriting relation on ground terms, and showed, through Theorem 1, that the approach of [12] still holds with the generalized narrowing relation. A lifting lemma involving priorities on the rules establishes correctness of this modelization, and then of Theorem 1. For details, see the appendix.

In fact, our innermost termination technique lends itself to this extension: as explained in Section 4.2, the priority mechanism can be expressed through negations of substitutions, then introducing constraints similar to those already required to model ground innermost rewriting.

Constraints are crucial in our approach: ordering constraints guarantee the applicability of the induction principle, abstraction constraints define the ground terms considered at each step of the proof, and help to contain the narrowing mechanism. The first ones are often satisfiable by a Path Ordering. Otherwise, any automatic ordering constraint solver can be used. For the second ones, if the satisfiability is considered, constructor-based solutions can be synthesized in an automatable way, like for the two examples of Section 5. Otherwise, solutions based on the characterization of normal forms of the rewrite system may be considered [11, 6]. If the usatisfiability of abstraction constraints is considered instead, simple automatable sufficient conditions as those given in Section 6 can be used. Analyzing A can even be completely avoided for examples exploiting the first condition of the **Stop** rule.

As termination of the original priority rewriting relation of [4] guarantees a semantics for this relation, one can think that IP-termination guarantees a semantics for the IP-rewriting relation. This has to be investigated. We also plan to generalize our technique to the termination proof of other priority rewriting relations.

8. REFERENCES

[1] L. Augustsson. A compiler for lazy ML. In *LFP '84: Proceedings of the 1984 ACM Symposium on LISP and functional programming*, pages 218–227, New York, NY, USA, 1984. ACM.

[2] F. Baader and T. Nipkow. *Term rewriting and all that.* Cambridge University Press, New York, NY, USA, 1998.

[3] J. C. M. Baeten, J. A. Bergstra, and J. W. Klop. Term rewriting systems with priorities. In *Proceedings of the 2nd International Conference on Rewriting Techniques and Applications*, volume 256 of *Lecture Notes in Computer Science*, pages 83–94. Springer Verlag, 1987.

[4] J. C. M. Baeten, J. A. Bergstra, J. W. Klop, and W. P. Weijland. Term-rewriting systems with rule priorities. *Theoretical Computer Science*, 67(2-3):283–301, 1989.

[5] Home of Clean. http://clean.cs.ru.nl/index.html.

[6] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications, 1997. Release October, 1rst 2002.

[7] A. S. de Oliveira. *Réécriture et Modularité pour les Politiques de Scurité*. PhD thesis, Univesité Henri Poincaré, Nancy, France, 2008.

[8] A. S. de Oliveira, E. K. Wang, C. Kirchner, and H. Kirchner. Weaving rewrite-based access control policies. In *FMSE '07: Proceedings of the 2007 ACM workshop on Formal methods in security engineering*, pages 71–80, New York, NY, USA, 2007. ACM.

[9] N. Dershowitz and D. A. Plaisted. Rewriting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 9, pages 535–610. Elsevier Science, 2001.

[10] D. J. Dougherty, K. Fisler, and S. Krishnamurthi. Specifying and reasoning about dynamic access-control policies. In *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR)*, volume 4130 of *Lecture Notes in Computer Science*, pages 632–646, Seattle, Washington, USA, 2006. Springer.

[11] T. Genet. Decidable approximations of sets of descendants and sets of normal forms. In *Proceedings of the 9th Conference on Rewriting Techniques and Applications, Tsukuba (Japan)*, volume 1379 of *Lecture Notes in Computer Science*, pages 151–165. Springer-Verlag, 1998.

[12] I. Gnaedig and H. Kirchner. Termination of Rewriting under Strategies. *ACM Transactions on Computational Logic*, 2007. To appear. Preliminary version available at http://www.loria.fr/∼gnaedig/PAPERS/REPORTS/-S-termin-2007-preli.pdf.

[13] Wiki homepage of Haskell. http://www.haskell.org/haskellwiki/Haskell.

[14] J. B. Kruskal. Well-quasi ordering, the tree theorem and Vazsonyi's conjecture. *Trans. Amer. Math. Soc.*, 95:210–225, 1960.

[15] N. Marti-Oliet, J. Meseguer, and A. Verdejo. Towards a strategy language for Maude. In *Proceedings of the Fifth International Workshop on Rewriting Logic and Its Applications*, volume 117 of *Electronic Notes in Theoretical Computer Science*, pages 417–441. Elsevier Science, 2004.

[16] C. K. Mohan. Priority rewriting: Semantics, confluence, and conditionals. In *Proceedings of the 3rd International Conference on Rewriting Techniques and Applications*, volume 355 of *Lecture Notes in Computer Science*, pages 278–291. Springer Verlag, 1989.

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

[18] R. Plasmeijer and M. van Eekelen. *Functional Programming and Parallel Graph Rewriting*. Addison Wesley, 1993.

[19] M. Sakai and Y. Toyama. Semantics and strong sequentiality of priority term rewriting systems. *Theoretical Computer Science*, 208(1–2):87–110, 1998.

[20] Terese. *Term Rewriting Systems*. Number 55 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2003.

[21] J. van de Pol. Operational semantics of rewriting with priorities. *Theoretical Computer Science*, 200(1-2):289–312, 1998.

[22] J. van de Pol and H. Zantema. Generalized innermost rewriting. In *Proceedings of the 16th International Conference on Rewriting Techniques and Applications*, volume 3467 of *Lecture Notes in Computer Science*, pages 2–16, Nara, Japan, 2005. Springer.

[23] M. G. van den Brand, P. Klint, and C. Verhoef. Term Rewriting for Sale. In *Proceedings of the First International Workshop on Rewriting Logic and its Applications*, volume 15 of *Electronic Notes in Theoretical Computer Science*, pages 139–161. Elsevier Science, 1998.

APPENDIX

A. THE LIFTING LEMMA

For the proof of Theorem 1, we need the following lifting lemma.

LEMMA 1 (PRIORITY INNER. LIFTING LEMMA). *Let $\mathcal{R}$ be a term rewrite system. Let $s \in \mathcal{T}(\mathcal{F},\mathcal{X})$, α a ground substitution such that αs is IP-reducible at a non variable position p of s, and $\mathcal{Y} \subseteq \mathcal{X}$ a set of variables such that $Var(s) \cup Dom(\alpha) \subseteq \mathcal{Y}$. If $\alpha s \rightarrow^{IP}_{p,l\rightarrow r} t'$, then there exist a term $s' \in \mathcal{T}(\mathcal{F},\mathcal{X})$ and substitutions $\beta,\sigma = \sigma_0 \wedge \bigwedge_{j\in[1..k]} \overline{\sigma_j} \bigwedge_{i\in[1..n]} \overline{\sigma_0^i}$. such that:*

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

where σ_0 is the most general unifier of $s|_p$ and l and $\sigma_j, j \in [1..k]$ are all most general unifiers of $\sigma_0 s|_{p'}$ and a left-hand side l' of a rule of $\mathcal{R}$, for all suffix position p' of p in s, and $\sigma_0^1,\ldots,\sigma_0^n$ are the most general unifiers of $s|_p$ with the left-hand sides of the rules having a greater priority than $l \rightarrow r$.

For the proof of the lemma, we need the two following propositions.

PROPOSITION 1. *Let $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$ and σ be a substitution of $\mathcal{T}(\mathcal{F},\mathcal{X})$. Then $Var(\sigma t) = (Var(t) - Dom(\sigma)) \cup Ran(\sigma_{Var(t)})$.*

PROPOSITION 2. *Suppose we have substitutions σ,μ,ν and sets A,B of variables such that $(B - Dom(\sigma)) \cup Ran(\sigma) \subseteq A$. If $\mu = \nu[A]$ then $\mu\sigma = \nu\sigma[B]$.*

PROOF. Let us consider $(\mu\sigma)_B$, which can be divided as follows: $(\mu\sigma)_B = (\mu\sigma)_{B\cap Dom(\sigma)} \cup (\mu\sigma)_{B-Dom(\sigma)}$.
For $x \in B \cap Dom(\sigma)$, we have $\mathcal{V}ar(\sigma x) \subseteq Ran(\sigma)$, and then $(\mu\sigma)x = \mu(\sigma x) = \mu_{Ran(\sigma)}(\sigma x) = (\mu_{Ran(\sigma)}\sigma)x$. Therefore $(\mu\sigma)_{B\cap Dom(\sigma)} = (\mu_{Ran(\sigma)}\sigma)_{B\cap Dom(\sigma)}$.
For $x \in B - Dom(\sigma)$, we have $\sigma x = x$, and then $(\mu\sigma)x = \mu(\sigma x) = \mu x$. Therefore we have $(\mu\sigma)_{B-Dom(\sigma)} = \mu_{B-Dom(\sigma)}$.
Henceforth we get $(\mu\sigma)_B = (\mu_{Ran(\sigma)}\sigma)_{B\cap Dom(\sigma)} \cup \mu_{B-Dom(\sigma)}$.
By a similar reasoning, we get $(\nu\sigma)_B = (\nu_{Ran(\sigma)}\sigma)_{B\cap Dom(\sigma)} \cup \nu_{B-Dom(\sigma)}$.
By hypothesis, we have $Ran(\sigma) \subseteq A$ and $\mu = \nu[A]$. Then $\mu_{Ran(\sigma)} = \nu_{Ran(\sigma)}$. Likewise, since $B - Dom(\sigma) \subseteq A$, we have $\mu_{B-Dom(\sigma)} = \nu_{B-Dom(\sigma)}$.
Then we have $(\mu\sigma)_B = (\mu_{Ran(\sigma)}\sigma)_{B\cap Dom(\sigma)} \cup \mu_{B-Dom(\sigma)} = (\nu_{Ran(\sigma)}\sigma)_{B\cap Dom(\sigma)} \cup \nu_{B-Dom(\sigma)} = (\nu\sigma)_B$. Therefore $(\mu\sigma) = (\nu\sigma)[B]$. □

PROOF OF LEMMA 1. In the following, we assume that $\mathcal{Y} \cap \mathcal{V}ar(l) = \emptyset$ for every $l \rightarrow r \in \mathcal{R}$.
If $\alpha s \rightarrow^{IP}_{p,l\rightarrow r} t'$, then there is a substitution τ such that $Dom(\tau) \subseteq \mathcal{V}ar(l)$ and $(\alpha s)|_p = \tau l$. Moreover, since p is a non variable position of s, we have $(\alpha s)|_p = \alpha(s|_p)$. Denoting $\mu = \alpha\tau$, we have:

$$\begin{array}{lll} \mu(s|_p) & = \alpha(s|_p) & \text{for } Dom(\tau) \subseteq \mathcal{V}ar(l) \text{ and } \mathcal{V}ar(l) \cap \\ & & \mathcal{V}ar(s) = \emptyset \\ & = \tau l & \text{by definition of } \tau \\ & = \mu l & \text{for } Dom(\alpha) \subseteq \mathcal{Y} \text{ and } \mathcal{Y} \cap \mathcal{V}ar(l) = \emptyset, \end{array}$$

and therefore $s|_p$ and l are unifiable. Let us denote by σ_0 the most general unifier of $s|_p$ and l, and $s' = \sigma_0(s[r]_p)$.

Since σ_0 is more general than μ, there is a substitution ρ such that $\rho\sigma_0 = \mu[\mathcal{Y} \cup Var(l)]$. Let $\mathcal{Y}_1 = (\mathcal{Y} - Dom(\sigma_0)) \cup Ran(\sigma_0)$. We define $\beta = \rho_{\mathcal{Y}_1}$. Clearly $Dom(\beta) \subseteq \mathcal{Y}_1$.
We now show that $\mathcal{V}ar(s') \subseteq \mathcal{Y}_1$, by the following reasoning:

- since $s' = \sigma_0(s[r]_p)$, we have $\mathcal{V}ar(s') = \mathcal{V}ar(\sigma_0(s[r]_p))$;
- the rule $l \rightarrow r$ is such that $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$, therefore we have $\mathcal{V}ar(\sigma_0(s[r]_p)) \subseteq \mathcal{V}ar(\sigma_0(s[l]_p))$, and then, thanks to the previous point, $\mathcal{V}ar(s') \subseteq \mathcal{V}ar(\sigma_0(s[l]_p))$;
- since $\sigma_0(s[l]_p) = \sigma_0 s[\sigma_0 l]_p$ and since σ_0 unifies l and $s|_p$, we get $\sigma_0(s[l]_p) = (\sigma_0 s)[\sigma_0(s|_p)]_p = \sigma_0 s[s|_p]_p = \sigma_0 s$ and, thanks to the previous point: $\mathcal{V}ar(s') \subseteq \mathcal{V}ar(\sigma_0 s)$;
- according to Proposition 1, we have $\mathcal{V}ar(\sigma_0(s)) = (\mathcal{V}ar(s) - Dom(\sigma_0)) \cup Ran(\sigma_{0\mathcal{V}ar(s)})$; by hypothesis, $\mathcal{V}ar(s) \subseteq \mathcal{Y}$. Moreover, since $Ran(\sigma_{0\mathcal{V}ar(s)}) \subseteq Ran(\sigma_0)$, we have $\mathcal{V}ar(\sigma_0(s)) \subseteq (\mathcal{Y} - Dom(\sigma_0)) \cup Ran(\sigma_0)$, that is $\mathcal{V}ar(\sigma_0 s) \subseteq \mathcal{Y}_1$. Therefore, with the previous point, we get $Var(s') \subseteq \mathcal{Y}_1$.

From $Dom(\beta) \subseteq \mathcal{Y}_1$ and $Var(s') \subseteq \mathcal{Y}_1$, we infer $Dom(\beta) \cup Var(s') \subseteq \mathcal{Y}_1$.
Let us now prove that $\beta s' = t'$.
Since $\beta = \rho_{\mathcal{Y}_1}$, we have $\beta = \rho[\mathcal{Y}_1]$. Since $Var(s') \subseteq \mathcal{Y}_1$, we get $\beta s' = \rho s'$. Since $s' = \sigma_0(s[r]_p)$, we have $\rho s' = \rho\sigma_0(s[r]_p) = \mu(s[r]_p) = \mu s[\mu r]_p$. Then $\beta s' = \mu s[\mu r]_p$.
We have $Dom(\tau) \subseteq \mathcal{V}ar(l)$ and $\mathcal{Y} \cap \mathcal{V}ar(l) = \emptyset$, then we have $\mathcal{Y} \cap Dom(\tau) = \emptyset$. Therefore, from $\mu = \alpha\tau[\mathcal{Y} \cup Var(l)]$, we get $\mu = \alpha[\mathcal{Y}]$. Since $\mathcal{V}ar(s) \subseteq \mathcal{Y}$, we get $\mu s = \alpha s$.
Likewise, by hypothesis we have $Dom(\alpha) \subseteq \mathcal{Y}$, $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$ and $\mathcal{Y} \cap \mathcal{V}ar(l) = \emptyset$, then we get $Var(r) \cap Dom(\alpha) = \emptyset$, and then we have $\mu = \tau[Var(r)]$, and therefore $\mu r = \tau r$.
From $\mu s = \alpha s$ and $\mu r = \tau r$ we get $\mu s[\mu r]_p = \alpha s[\tau r]_p$. Since, by hypothesis, $\alpha s \rightarrow^p t'$, with $\tau l = (\alpha s)|_p$, then $\alpha s[\tau r]_p = t'$. Finally, as $\beta s' = \mu s[\mu r]_p$, we get $\beta s' = t'$ (2).
Next let us prove that $\beta\sigma_0 = \alpha[\mathcal{Y}]$. Reminding that $\mathcal{Y}_1 = (\mathcal{Y} - Dom(\sigma_0)) \cup Ran(\sigma_0)$, Proposition 2 (with the notations A for $\mathcal{Y}_1$, B for $\mathcal{Y}$, μ for β, ν for ρ and σ for σ_0) yields $\beta\sigma_0 = \rho\sigma_0[\mathcal{Y}]$. We already noticed that $\mu = \alpha[\mathcal{Y}]$. Linking these two equalities via the equation $\rho\sigma_0 = \mu$ yields $\beta\sigma_0 = \alpha[\mathcal{Y}]$ (3).
Let us now suppose that there exist a rule $l' \rightarrow r' \in \mathcal{R}$, a suffix position p' of p and a substitution σ_i such that $\sigma_i(\sigma_0(s|_{p'})) = \sigma_i l'$.
Let us now suppose that β does not satisfy $\bigwedge_{j\in[1..k]} \overline{\sigma_j}$. There is $i \in [1..k]$ such that β satisfies $\sigma_i = \bigwedge_{i_l\in[1..n]}(x_{i_l} = u_{i_l})$. So β is such that $\bigwedge_{i_l\in[1..n]}(\beta x_{i_l} = \beta u_{i_l})$.
Thus, on $Dom(\beta) \cap Dom(\sigma_i) \subseteq \{x_{i_l}, i_l \in [1..n]\}$, we have $(\beta x_{i_l} = \beta u_{i_l})$, so $\beta\sigma_i = \beta$. Moreover, as β is a ground substitution, $\sigma_i\beta = \beta$. Thus, $\beta\sigma_i = \sigma_i\beta$.
On $Dom(\beta) \cup Dom(\sigma_i) - (Dom(\beta) \cap Dom(\sigma_i))$, either $\beta = Id$, or $\sigma_i = Id$, so $\beta\sigma_i = \sigma_i\beta$.
As a consequence, $\alpha(s) = \sigma_i\alpha(s) = \sigma_i\beta\sigma_0(s) = \beta\sigma_i\sigma_0(s)$ is reducible at position p' with the rule $l' \rightarrow r'$, which is impossible by definition of innermost reducibility of $\alpha(s)$ at position p. So the ground substitution β satisfies $\bigwedge_{i\in[1..k]} \overline{\sigma_i}$ for all most general unifiers σ_i of $\sigma_0 s$ and a left-hand side of a rule of $\mathcal{R}$ at suffix positions of p.
Let us now suppose that there exist a rule $l' \rightarrow r' \in \mathcal{R}$ of higher priority than $l \rightarrow r$ and a substitution σ_0^i such that $\sigma_0^i(s|_p) = \sigma_0^i l'$. With a similar reasoning than previously, we get that $\alpha(s)$ is reducible at position p with the rule $l' \rightarrow r'$, which has higher priority than $l \rightarrow r$. This is impossible by definition of IP-reducibility of $\alpha(s)$ by $l \rightarrow r$ at position

p. So the ground substitution β also satisfies $\bigwedge_{i\in[1..n]} \overline{\sigma_0^i}$ where $\sigma_0^1, \ldots, \sigma_0^n$ are the most general unifiers of $s|_p$ with the left-hand sides of rules having a greater priority than $l \to r$ (4).

Therefore, denoting $\sigma = \sigma_0 \wedge \bigwedge_{j\in[1..k]} \overline{\sigma_j} \wedge \bigwedge_{i\in[1..n]} \overline{\sigma_0^i}$, from the beginning of the proof, we get $s \leadsto^{IP}_{p,l\to r,\sigma} s'$, and then the point (1) of the current lemma holds. □

B. THE IP-TERMINATION THEOREM

Theorem 1. *Let $\mathcal{R}$ be a priority term rewrite system on $\mathcal{T}(\mathcal{F},\mathcal{X})$ having at least one constructor constant. Every term of $\mathcal{T}(\mathcal{F})$ is IP-terminating iff there is a noetherian ordering $\succ$ such that for each symbol $g \in \mathcal{D}$, we have $SUCCESS(g,\succ)$.*

Proof. Let us suppose that every ground term is IP-terminating and show that the construction of the proof trees always terminate. Let $f(x_1,\ldots,x_m)$, $f \in \mathcal{D}$ any initial pattern of a proof tree.

The **Abstract** rule applies to give $f(X_1,\ldots,X_m), X_1, \ldots, X_m \in \mathcal{N}$. Indeed, by hypothesis, we have $IPT(x_i)$. Then **Stop** applies, because we also have $IPT(f(X_1,\ldots,X_m))$. So any proof tree is finite, and $SUCCESS(f,\succ)$ for every $f \in \mathcal{D}$, with any noetherian ordering $\succ$.

For the converse part, we prove by induction on $\mathcal{T}(\mathcal{F})$ that any ground instance $\theta f(x_1,\ldots,x_m)$ IP-terminates for any term $f(x_1,\ldots,x_m) \in \mathcal{T}(\mathcal{F},\mathcal{X})$ with $f \in \mathcal{F}$. We use an abstraction lemma, a narrowing lemma, and a stopping lemma, which are given after this main proof.

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 by $\succ$.

If f is a defined symbol, let us denote it by g and prove that $g(\theta x_1,\ldots,\theta x_m)$ is IP-terminating for any θ satisfying $A = \top$ if we have $SUCCESS\ (h,\succ)$ for every defined symbol h. Note that g may be a reducible constant. Let us denote $g(x_1,\ldots,x_m)$ by t_{ref} in the sequel of the proof.

To each state s of the proof tree of g, characterized by a current term t and the set of constraints A, we associate the set of ground terms $G = \{\alpha t \mid \alpha \text{ satisfies } A\}$, that is the set of ground instances represented by s. When t is a reducible constant, the set of ground instances is reduced to t itself.

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

By abstraction (resp. narrowing) Lemma, when applying **Abstract** (resp. **Narrow**), for each reducible αt in G, there is a $\beta t'$ (resp. there are $\beta_i t'_i$) in G' and such that IP-termination of $\beta t'$ (resp. of the $\beta_i t'_i$) implies IP-termination of αt.

When the **Stop** inference rule applies on $(\{t\},A,C)$, by stopping lemma, every term of $G = \{\alpha t \mid \alpha \text{ satisfies } A\}$ is IP-terminating. Therefore, IP-termination is ensured for all terms in all sets G in the proof tree.

As the process is initialized with $\{t_{ref}\}$ and a set A of abstraction constraints satisfiable by any ground substitution, we get that $g(\theta x_1,\ldots,\theta x_m)$ is IP-terminating, for any $t_{ref} = g(x_1,\ldots,x_m)$, and any ground instance θ.

If f is a constructor, either it is a constant, which is irreducible, and then IP-terminating, or we consider the pattern $f(x_1,\ldots,x_m)$. The proof then works like in the case of defined symbols, but with just an application of **Abstract** and **Stop**. Indeed, $f(x_1,\ldots,x_m)$ always abstracts into $f(X_1,\ldots,X_m)$. Then **Stop** applies because $f(X_1,\ldots,X_m)$ is not narrowable and all its variables are in $\mathcal{N}$. □

Lemma 2 (Abstraction lemma). *Let $(\{t\},A,C)$ be a state of any proof tree, giving the state $(\{t' = t[X_j]_{j\in\{i_1,\ldots,i_p\}}\}, A', C')$ by application of* **Abstract**. *For any ground substitution α satisfying A, if αt is reducible, there is β such that IP-termination of $\beta t'$ implies IP-termination of αt. Moreover, β satisfies A'.*

Proof. We prove that $\alpha t \xrightarrow{*}_S \beta t'$, where $\beta = \alpha \cup \bigcup_{j\in\{i_1,\ldots,i_p\}} X_j = \alpha t|_j\downarrow$.

First, the abstraction positions in t are chosen so that the $\alpha t|_j$ can be supposed IP-terminating. Indeed, each term $t|_j$ is such that:

- either $IPT(t|_j)$ is true, and then by definition of the predicate IPT, $\alpha t|_j$ is IP-terminating;
- or $t_{ref} > t|_j$ is satisfiable by $\succ$, and then, by induction hypothesis, $\alpha t|_j$ is IP-terminating.

So, $\alpha t|_j$ reduces to an IP-normal form $\alpha t|_j\downarrow$. Then, whatever the positions $i_1,\ldots,i_p$ in the term t, we have $\alpha t \xrightarrow{*}{}^{IP} \alpha t[\alpha t|_{i_1}\downarrow]_{i_1} \ldots [\alpha t|_{i_p}\downarrow]_{i_p} = \beta t'$.

Thus, $\alpha t \xrightarrow{*}{}^{IP} \beta t'$ for every derivation that normalizes all subtems $\alpha t|_j\downarrow$, for $j \in \{p_1,\ldots,p_k\}$. As every $\beta t'$ represents a reduced form of αt on every possible rewriting branch of αt, then IP-termination of $\beta t'$ implies IP-termination of αt.

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

Lemma 3 (Narrowing lemma). *Let $(\{t\},A,C)$ be a state of any proof tree, giving the states $(\{v_i\}, A'_i, C'_i), i \in [1..l]$, by application of* **Narrow**. *For any ground substitution α satisfying A, if αt is IP-reducible, then, for each $i \in [1..l]$, there is β_i such that IP-termination of the $\beta_i v_i, i \in [1..l]$, implies IP-termination of αt. Moreover, β_i satisfies A'_i for each $i \in [1..l]$.*

Proof. For any rewriting step $\alpha t \to^{IP}_{p,l\to r} t'$, by Lifting Lemma, there is a term $v \in \mathcal{T}(\mathcal{F},\mathcal{X})$ and substitutions $\beta, \sigma = \sigma_0 \wedge \bigwedge_{j\in[1..k]} \overline{\sigma_j} \wedge \bigwedge_{i\in[1..n]} \overline{\sigma_0^i}$ such that:

1. $t \leadsto^{IP}_{p,l\to r,\sigma} v$,
2. $\beta v = t'$,
3. $\beta\sigma_0 = \alpha[\mathcal{Y} \cup Var(l)]$
4. β satisfies $\bigwedge_{j\in[1..k]} \overline{\sigma_j} \wedge \bigwedge_{i\in[1..n]} \overline{\sigma_0^i}$

where σ_0 is the most general unifier of $t|_p$ and l, $\sigma_j, j \in [1..k]$ are all the most general unifiers of $\sigma_0 t|_{p'}$ and a left-hand side l' of a rule of $\mathcal{R}$, for all suffix positions p' of p in s, $\sigma_0^i, i \in [1..n]$ are all the most general unifiers of $t|_p$ with the left-hand sides of the rules having greater priority than $l \to r$.

The narrowing steps are effectively produced in the proof tree by the **Narrow** rule, applied in all possible ways on t. Then, the narrowing step $t \leadsto^{IP}_{p,l\to r,\sigma} v$ is produced. So a term βv is produced for every IP-rewriting branch starting from αt. Then IP-termination of the βv implies IP-termination of αt.

Let us prove that β satisfies $A' = A \wedge \sigma_0 \wedge \bigwedge_{j\in[1..k]} \overline{\sigma_j} \wedge \bigwedge_{i\in[1..n]} \overline{\sigma_0^i}$.

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 $Var(A)$.

More precisely, on $Ran(\sigma_0)$, β is such that $\beta\sigma_0 = \alpha$ and on $Var(A) \setminus Ran(\sigma_0)$, $\beta = \alpha$. As $Ran(\sigma_0)$ only contains fresh variables, we have $Var(A) \cap Ran(\sigma_0) = \emptyset$, so $Var(A) \setminus Ran(\sigma_0) = Var(A)$. So $\beta = \alpha$ on $Var(A)$ 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 $A' = A \wedge \sigma_0 \wedge \bigwedge_{j\in[1..k]} \overline{\sigma_j} \wedge \bigwedge_{i\in[1..n]} \overline{\sigma_0^i}$. □

LEMMA 4 (STOPPING LEMMA). *Let $(\{t\}, A, C)$ be a state of any proof tree, with A satisfiable, and giving the state $(\emptyset, A', C')$ by application of an inference rule. Then for every ground substitution α satisfying A, αt is IP-terminating.*

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

- either $IPT(t)$ and then αt is IP-terminating for every ground substitution α,
- or $(t_{ref} > t)$ is satisfiable. Then, for every ground substitution α satisfying A, $\alpha t_{ref} \succ \alpha t$. By induction hypothesis, αt is IP-terminating.

□