Termination of Priority Rewriting

Isabelle Gnaedig

Termination of Priority Rewriting *

Isabelle Gnaedig

INRIA & LORIA
BP 101, 54602 Villers-lès-Nancy Cedex France

Abstract. Introducing priorities in rewriting increases the expressive power of rules 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 an inductive proof method for termination of priority rewriting, relying on an explicit induction on the termination property and working by generating proof trees, which model the rewriting relation by using abstraction and narrowing.

1 Introduction

In [1, 2], priority rewriting systems (PRSs 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 [3] or Maude [4]. It is also used as a computation model for functional programming [5], and is underlying in the functional strategy, used for example in Lazy ML [6], Clean [7], or Haskell [8]. Let us also cite recent works on specification and correctness of security policies using rewriting with priorities [9, 10].

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 [5], or to use the innermost strategy [11]. But in these cases, normalization can lead to non-termination.

Second, the semantics of a PRS is not always clearly defined. In [1], 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 shown that bounded -the bounded property is weaker than termination- PRSs have a semantics. In [12], 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 [11], a logical semantics of PRSs based on equational logic is given. A particular class of PRSs is proved sound and complete with respect to the initial algebra, provided every priority rewriting sequence from every ground term terminates.

* The original publication is available at www.springerlink.com https://link.springer.com/chapter/10.1007%2F978-3-642-00982-2_33

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 [13], discussing a normalizing strategy of PRSs i.e., a strategy giving only finite derivations for terms having a normal form with usual rewriting, and [14], 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 addressed in [11], 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 the termination proof 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 [11], because it is decidable, easy to manipulate, and the innermost strategy is often used in programming contexts where priorities on rules are considered. The previously cited works on rule-based security policies also generated a need of specific termination tools: the specifications given in [9] have indeed been executed in TOM [15] 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 approach, whose principle has already been applied for termination of rewriting under strategies [16]. The idea is to prove, in developing proof trees simulating the rewriting trees, that every derivation starting from any term terminates, supposing that it is true for terms smaller than the starting terms. We then introduce the priority notion in the generation mechanism of the proof trees, and show how to optimize the technique in this new case.

2 Priority Rewriting

The Background We assume that the reader is familiar with the basic definitions and notations of term rewriting given for instance in [17, 18]. The ones needed in the paper can also be found in [19].

$\mathcal{T}(\mathcal{F}, \mathcal{X})$ 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 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 the most general unifier of $t|_p$ and l, and $t' = \sigma(t[r]_p)$. This is denoted by $t \rightsquigarrow^{\mathcal{R}}_{p,l\to r,\sigma} t'$ 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 monotonic iff for any pair of terms t, t' in $\mathcal{T}(\mathcal{F}, \mathcal{X})$, 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 monotonic and has the subterm property, then it is noetherian [20]. 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.

Priority Rewriting A *priority term rewrite system* is a pair $(\mathcal{R}, \blacktriangleright)$ of a term rewrite system $\mathcal{R}$ (always considered as finite here) 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 ([11]). *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 rewrites into $t : t = s[\sigma r]_p$ with $s|_p = \sigma l$,*
- *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 1. With the PRS $\{f(g(x)) \to b, g(a) \to c \blacktriangleright g(a) \to d\}$, on $f(g(a))$, the first rule should apply, but this would not be an innermost rewrite step. So the second rule applies, but the third one does not, because $g(a) \to c \blacktriangleright g(a) \to d$.

A PRS $\mathcal{R}$ *IP-terminates* if and only if every IP-rewriting chain (IP-derivation) of the rewriting relation induced by $\mathcal{R}$ is finite. If t' is in an IP-derivation issued from t

and t' is IP-irreducible, then t' is called a(n) *(IP-)normal form* of t and is 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 [11], to guarantee stability by substitution of the innermost rewriting relation, the rules without highest priority only can reduce ground terms. Finally, there are TRSs 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 PRS 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 starting from t is finite.

For proving that a term t of $\mathcal{T}(\mathcal{F})$ is IP-terminating, we then 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.

Rewriting derivations are simulated, using a lifting mechanism, by proof trees developed from initial patterns $t_{ref} = g(x_1,\ldots,x_m)$ on $\mathcal{T}(\mathcal{F},\mathcal{X})$, for every $g \in \mathcal{D}$, by alternatively using narrowing and an abstraction mechanism. For a term t of $\mathcal{T}(\mathcal{F},\mathcal{X})$ occurring in a proof tree,

- first, some subterms $\theta t|_j$ of θt are supposed to be IP-terminating for every instantiation θ 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.* Obviously, the whole term t can be abstracted. 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 presented later), then the ground instances of the initial term are IP-terminating. Otherwise,
- the resulting term $u = t[X_j]_{\{i_1,\ldots,i_p\}}$ ($i_1,\ldots,i_p$ 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.

The process is iterated on each 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 [16]. We now give the concepts needed to formalize and automate it.

4 Abstraction, Narrowing, Constraints

Ordering Constraints The induction ordering is not defined a priori but is constrained along the proof by inequalities between terms that must be comparable, called

ordering constraints, each time the induction hypothesis is used for abstraction. More formally, 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 of this form is undecidable. But a sufficient condition for C to be satisfiable is to find a simplification ordering $\succ$ such that $t \succ t'$ for every constraint $t > t'$ of C.

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 $\succ$.

The constraints generated by our approach are often satisfiable by a Recursive Path Ordering or a Lexicographic Path Ordering. Otherwise, automatic constraint solvers can provide adequate polynomial orderings. See [16] for experiments.

Abstraction Let us define the abstraction variables more formally.

Definition 2 (abstraction variable [16]). *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 $\mathcal{T}(\mathcal{F},\mathcal{X}\cup\mathcal{N})$ so that for every substitution σ (resp. instantiation θ) such that $\mathcal{D}om(\sigma)$ (resp. $\mathcal{D}om(\theta)$) contains a variable $X \in \mathcal{N}$, σX (resp. θX) is in IP-normal form.*

Definition 3 (term abstraction [16]). *Let $i_1,\ldots,i_p$ be positions of t such that no one is prefix of another one. 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 on $\mathcal{T}(\mathcal{F},\mathcal{X}\cup\mathcal{N})$. When a subterm $t|_j$ is abstracted by X_j, we state an *abstraction constraint* $t|_j{\downarrow} = X_j$ where $t \in \mathcal{T}(\mathcal{F},\mathcal{X}\cup\mathcal{N})$ and $X \in \mathcal{N}$, to express that its instances can only be normal forms of the corresponding instances of $t|_j$.

Narrowing After abstracting the current term t into $t' = t[X_j]_{j\in\{i_1,\ldots,i_p\}}$, we test whether the possible ground instances of t' are reducible, according to the possible values of the instances of the X_j, by narrowing t' with the PRS.

To schematize innermost rewriting on ground terms, in [16], we introduced a specific narrowing definition involving constrained substitutions. The problem here is to see how to express priorities in the narrowing mechanism and how to integrate them in the previous constraint based definition. In [16], the usual notion of narrowing was refined as follows. 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.

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 4 ([16]). *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 5 (Innermost narrowing [16]). *Let $\mathcal{R}$ be a TRS. A term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$* innermost narrows *into a term $t' \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$ at the non-variable position p of t, using the rule $l \to r \in \mathcal{R}$ with the constrained substitution $\sigma = \sigma_0 \wedge \bigwedge_{j \in [1..k]} \overline{\sigma_j}$, 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 (*mgu*) of $t|_p$ and l and $\sigma_j, j \in [1..k]$ are all* mgus *of $\sigma_0 t|_{p'}$ and a* lhs *l' of a rule of $\mathcal{R}$, for all suffix position p' of p in t.*

It is always assumed that there is no variable in common between the rule and the term i.e., that $Var(l) \cap Var(t) = \emptyset$. In the following, we are only interested in the restriction of the narrowing substitution applied to the current term t. We then omit its definition on the variables of the *lhs* of rules.

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 take into account the narrowing steps with the rules having a higher priority. Like for the innermost mechanism of Definition 5, this requires to use negations of substitutions. Let us consider the PRS $\{f(g(x), y) \to a \blacktriangleright f(x, h(y)) \to b \blacktriangleright f(x, y) \to c\}$. The term $f(x, y)$ innermost narrows into a with the first rule and the *mgu* $\sigma_1 = (x = g(x'))$, into b with the second rule, the *mgu* $\sigma_2 = (y = h(y'))$ and the constraint $x \neq g(x')$ (which is the negation of the *mgu* of $\sigma_2 f(x, y)$ with the *lhs* of the first rule), and finally into c with the third rule, the *mgu* σ_3 equal to Id and the constraint $x \neq g(x') \wedge y \neq h(y')$ (which is the negation of the *mgu* of $\sigma_3 f(x, y)$ with the first rule and of the *mgu* of $\sigma_3 f(x, y)$ with the second rule).

So, applying the rules one after the other on t, with the current *mgu* σ, we have to accumulate the negation of the *mgu*s of σt and the previous rules, without their constraint part. We have now to see how to manage together the constraints due to the innermost mechanism and those due to the priority mechanism.

If the narrowing substitutions are in their full form $\sigma_0 \wedge \bigwedge_{j\in[1..k]} \overline{\sigma_j}$ with a constraint part coming from the innermost mechanism of Definition 5, this constraint part is also ignored by the priority mechanism. Indeed, it is defined from σ_0, and has no meaning for the negation of σ_0. With the PRS $\{f(g(h(x))) \to a \blacktriangleright h(a) \to b \blacktriangleright f(g(x)) \to c\}$, the term $f(x)$ innermost narrows into *with* a 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 $\sigma_3 = (x = g(x') \wedge x' \neq h(x''))$ (where $(x' = h(x'') \wedge x'' \neq a)$ is the narrowing substitution of $\sigma_3 f(x)$ with the first rule.

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 *mgu* of the substitution. With the PRS $\{f(g(h(x,y)),z) \to a \blacktriangleright f(x,y) \to b$,
$h(a,x) \to a \blacktriangleright h(x,b) \to b\}$, the term $f(x,y)$ innermost narrows into a with the first rule and $\sigma_1 = (x = g(h(x',y')) \wedge x' \neq a \wedge y' \neq b)$, because $h(x',y')$ narrows with the third rule and $\sigma = (x' = a)$, and with the fourth rule and $\sigma = (y' = b \wedge x' \neq a)$. Note that the term $f(x,y)$ also innermost narrows into b with the second rule and $\sigma_2 = (Id \wedge x \neq g(h(x',y')))$.

Definition 6 (Innermost priority narrowing).
*Let $\mathcal{R}$ be a PRS. 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 $t \rightsquigarrow^{IP}_{p,l\to r,\sigma} t'$, iff $t' = \sigma_0(t[r]_p)$, where σ_0 is the* mgu *of $t|_p$ and l, $\sigma_j, j \in [1..k]$ are all* mgu*s of $\sigma_0 t|_{p'}$ and a* lhs *l' of a rule of $\mathcal{R}$, for all suffix position p' of p in t, and $\sigma_0^i, i \in [1..n]$ are the* mgu*s of $\sigma_0 t|_p$ with the* lhs *of the rules having a greater priority than $l \to r$.*

The following lifting lemma ensures that the previously defined narrowing relation simulates IP-rewriting on ground terms.

Lemma 1 (Priority Innermost Lifting Lemma).
Let $\mathcal{R}$ be a PRS. 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 $\mathcal{V}ar(s) \cup \mathcal{D}om(\alpha) \subseteq \mathcal{Y}$. If $\alpha s \to^{IP}_{p,l\to r} t'$, 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. $s \rightsquigarrow^{IP}_{p,l\to r,\sigma} s'$,
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 mgu *of $s|_p$ and l and $\sigma_j, j \in [1..k]$ are all* mgu*s of $\sigma_0 s|_{p'}$ and a* lhs *l' of a rule of $\mathcal{R}$, for all suffix position p' of p in s, and $\sigma_0^i, i \in [1..n]$ are the* mgu*s of $\sigma_0 s|_p$ with the* lhs *of the rules having a greater priority than $l \to r$.*

The proof of the Priority Innermost Lifting Lemma is given in the Appendix.

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 σ,

considered as the narrowing constraint attached to the narrowing step, is added to the abstraction constraint, or in practice, propagated into it by applying its substitution part to the variables of the 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 7. *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* $x_j \in \mathcal{X} \cup \mathcal{N}$*,* $t_i, t'_i, u_j, \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$*. It is* satisfiable *iff there is at least one instantiation* θ *such that* $\bigwedge_i(\theta t_i{\downarrow} = \theta t'_i) \wedge \bigwedge_j(\theta x_j = \theta u_j)$*;* θ *is then said to satisfy the ACF* A *and is called solution of* A*.*

Definition 8. *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 [21], are also under study. The unsatisfiability of A is also undecidable in general, but here also, simple automatable sufficient conditions can be used. In Sect. 5, we present the procedure exactly simulating the rewriting trees i.e., dealing with the satisfiability of A. In Sect. 6, we give the alternative approach dealing with the unsatisfiability, present the sufficient conditions to test it, and show why it is particularly advantageous for IP-rewriting.

5 The IP-termination Procedure

The inference rules 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 $i_1, \ldots, i_p$ 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 2, 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 6. 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 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.

Dealing with the predicate IPT Before to give the inference rules, 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. Simple cases often arise like non narrowable terms of $\mathcal{T}(\mathcal{F}, \mathcal{N})$.

Simple cases often arise like non narrowable terms of $\mathcal{T}(\mathcal{F}, \mathcal{N})$ 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 7, these narrowing possibilities do not represent any reduction step for the ground instances of u, which are then irreducible.

The notion of usable rules [22, 16] is also suitable to establish $IPT(u)$and can also be adapted to IP-rewriting. The set of usable rules of a term t is a computable superset of the rewrite rules that are likely to be used in any rewriting chain (for the usual rewriting relation i.e., the rewriting relation without strategy) starting from any ground instance of t, until its ground normal forms are reached, if they exist.

When t is a variable of $\mathcal{X}$, the set of usable rules of t is $\mathcal{R}$ itself. When $t \in \mathcal{N}$, the set of usable rules of t is empty, since the only possible instances of such a variable are ground terms in normal form. Otherwise, it is recursively computed on the term structure.

Definition 9 (Usable rules). *Let $\mathcal{R}$ be a rewrite system on a set $\mathcal{F}$ of symbols. Let $Rls(f) = \{l \to r \in \mathcal{R} \mid root(l) = f\}$. For any $t \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$, the set of usable rules of t, denoted $\mathcal{U}(t)$, is defined by:*

- $\mathcal{U}(t) = \mathcal{R}$ *if* $t \in \mathcal{X}$,
- $\mathcal{U}(t) = \emptyset$ *if* $t \in \mathcal{N}$,
- $\mathcal{U}(f(u_1, \ldots, u_n)) = Rls(f) \cup \bigcup_{i=1}^{n} \mathcal{U}(u_i) \cup \bigcup_{l \to r \in Rls(f)} \mathcal{U}(r)$.

Proving termination of every ground instance of t then comes down to proving termination of its usable rules, which is in general much easier than proving termination of the whole rewrite system $\mathcal{R}$. If there exists a reduction ordering $\succ_N$ that orients these rules, any ground instance αt is bound to terminate for the usual rewriting relation Indeed, if $\alpha t \to t_1 \to t_2 \to \ldots$, then $\alpha t \succ_N t_1 \succ_N t_2 \succ_N \ldots$ and, since the ordering $\succ_N$ is noetherian, the rewriting chain cannot be infinite.

Obviously, termination of the usable rules of t implies innermost termination of t, which in turn implies IP-termination. If termination cannot be proved, one can try to prove innermost termination with any existing technique. If innermost termination cannot be proved, IP-termination of the usable rules may also be proved with our inductive process itself. The fact that the induction ordering used for usable rules is independent of the main induction ordering, makes the proof very flexible.

The termination procedure The inference rules are given in Table 1. For a detailed explanation as well as considerations about the predicate IPT and the usable rules, see [19].

We generate the proof trees of $\mathcal{R}$ by applying, for each symbol $g \in \mathcal{D}$, the inference rules on the initial 3-tuple $(\{t_{ref} = g(x_1, \ldots, x_m)\}, \top, \top)$ (if g is a constant, then $t_{ref} = g$), with a specific strategy $S{-}RULES$:

$$repeat^*(try{-}skip(\textbf{Abstract}), try{-}stop(\textbf{Narrow}), try{-}skip(\textbf{Stop})).$$

"$repeat^*(T_1, \ldots, T_n)$" repeats the control strategies $T_1, \ldots, T_n$ until none of them is applicable anymore, $try{-}skip(T)$ expresses that T is tried and skipped when it cannot be applied, $try{-}stop(T)$ stops $S{-}RULES$ if T cannot be applied.

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 **Abstract** (resp. **Narrow**) when the ordering (resp. 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 $S{-}RULES$ 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 $g \in \mathcal{D}$, we have $SUCCESS(g, \succ)$.*

Table 1. Inference rules for IP-termination

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

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

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

Note that we do not have to consider the proof trees from the patterns $g(x_1,\ldots,x_m)$ where g is a constructor, because the induction reasoning is trivial on them. Applying the inference rules to any constructor pattern then gives rise to a successful proof tree by using once **Abstract** and **Stop**.

The proofs of Lemma 1 and Theorem 1 can be found in the Appendix.

Example 2. Let us consider the PRS

$$\begin{cases} f(g(h(x))) \to a \\ h(a) \to g(a) \\ f(g(x)) \to f(g(h(x))) \end{cases}$$

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

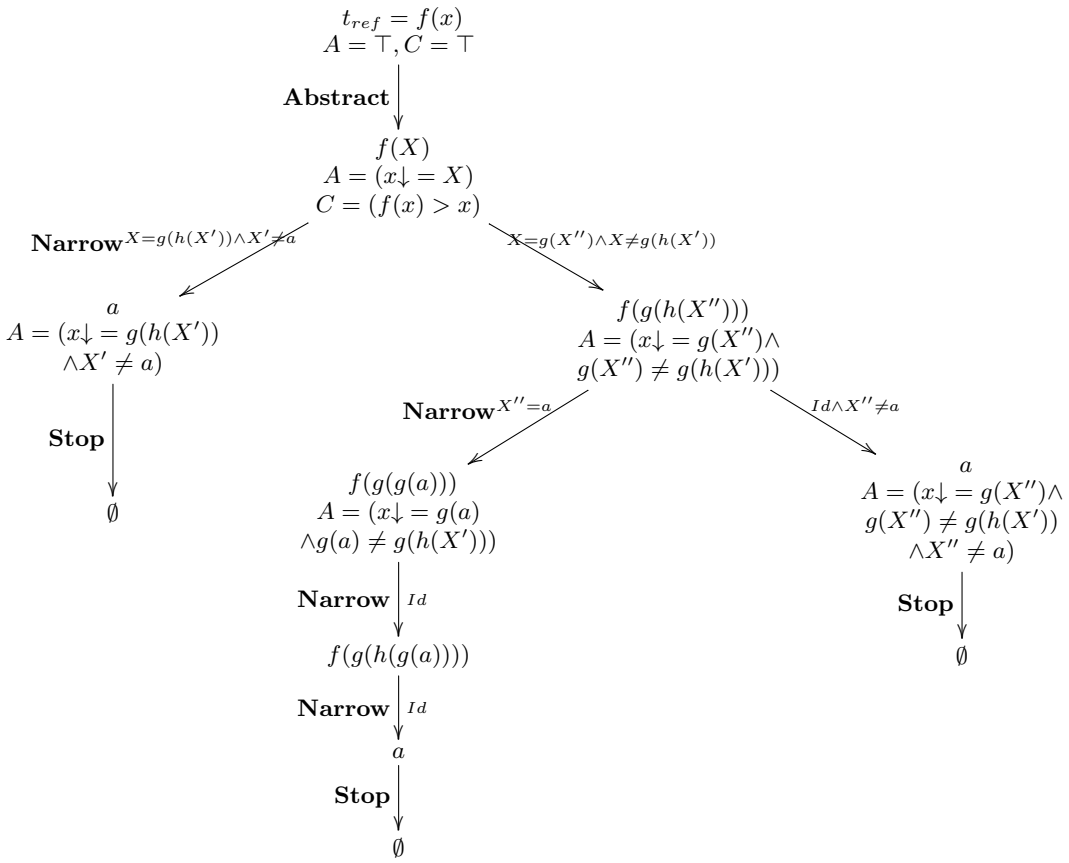
For a better readability of the proof, whenever we have $A\wedge\sigma$, we propagate σ into A, by applying the substitution part of σ to A. Moreover, the sets A and C are not repeated on a branch, when they do not change. The proof tree of f is given in Figure 1.

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 6.

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.

Fig. 1. Proof tree for symbol f

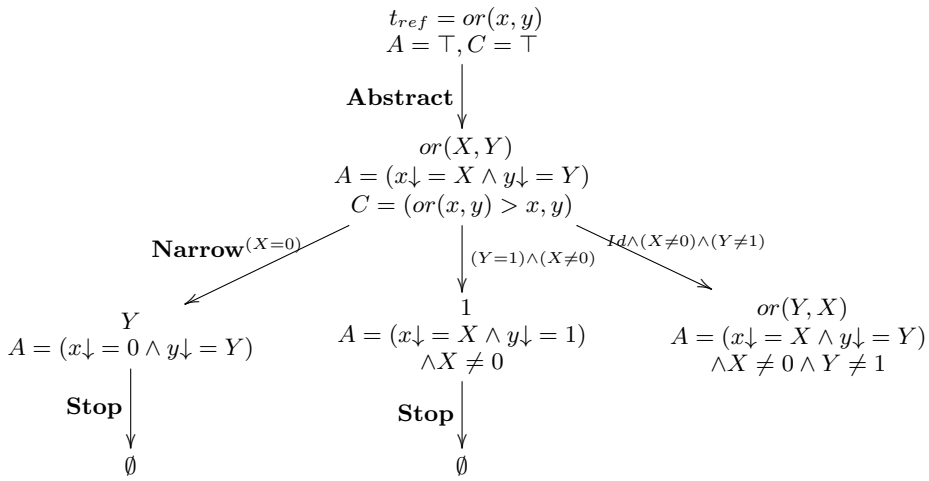


In the proof tree of h, we just have an **Abstract**, a **Narrow** and a **Stop** step. The ordering constraints are satisfied by the same noetherian ordering than above. Applying Theorem 1, we conclude that the PRS is IP-terminating.

Example 3. 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:

$$\begin{array}{l} or(0,y) \to y \quad (1) \\ or(x,1) \to 1 \quad (2) \\ or(x,y) \to or(y,x) \quad (3). \end{array}$$

Fig. 2. Proof tree for symbol *or*



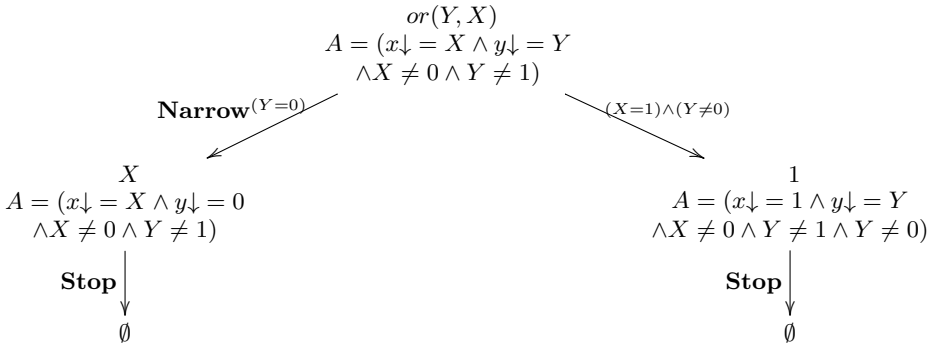
The proof tree of *or* is given in Figures 2 and 3. The **Stop** rule applies on $Y \in \mathcal{N}$ because, by definition of an abstraction variable, we have $IPT(Y)$.

Narrow applies on $or(Y,X)$ (see Figure 3), with $\sigma = (Y = 0)$ and $\sigma = (X = 1 \wedge Y \neq 0)$. There is also the substitution $\sigma = (Id \wedge Y \neq 0 \wedge X \neq 1)$, but it does not generate a new branch. Indeed, $A \wedge \sigma = (x\downarrow = X, y\downarrow = Y) \wedge X \neq 0 \wedge Y \neq 1 \wedge Y \neq 0 \wedge X \neq 1)$ 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.

Example 4. Let us consider the following PRS, whose underlying RS is also neither terminating, nor innermost terminating:

Fig. 3. Proof tree for symbol *or* - Continuation



$$\left\downarrow \begin{array}{ll} f(x, h(y)) \to a & (1) \\ f(g(x), y) \to f(g(x), h(y)) & (2) \\ f(x, y) \to f(g(x), y) & (3). \end{array} \right.$$

The proof tree of the only defined symbol is given in Figure 4.

Abstract applies on $f(x, y)$ because the ordering constraint $f(x) > x, y$ is satisfiable by any noetherian ordering having the subterm property. Then, **Narrow** applies on $f(X, Y)$ using Rules (1), (2), and (3), according to Definition 6.

On the second branch, the term $f(g(X'), h(Y))$ narrows into a with Rule (1) and $\sigma = Id$, so the other rules cannot apply: the negation of Id does not exist. This is coherent with the fact that Rule (1) has the highest priority, and applies for all possible instances of the term.

On the third branch, $f(g(X), Y)$ narrows into $f(g(X), h(Y))$ with Rule (2), but does not narrow with Rule (1), because the narrowing substitution would be $\sigma = (Y = h(Y'))$, and $A \wedge \sigma$ would be unsatisfiable. Indeed, we would have $Y \neq h(Y')$ and $Y = h(Y')$. The term $f(g(X), h(Y))$ does not narrow with Rule (3) either: it first narrows with Rule (2) and Id.

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

So the PRS is IP-terminating.

Example 5. Let us consider the following PRS, whose underlying RS is also neither terminating, nor innermost terminating:

$$\left\downarrow \begin{array}{ll} f(a) \to a & (1) \\ f(x) \to g(f(a)) & (2). \end{array} \right.$$

The proof tree of f is given in Figure 5.

Fig. 4. Proof tree for symbol *or*

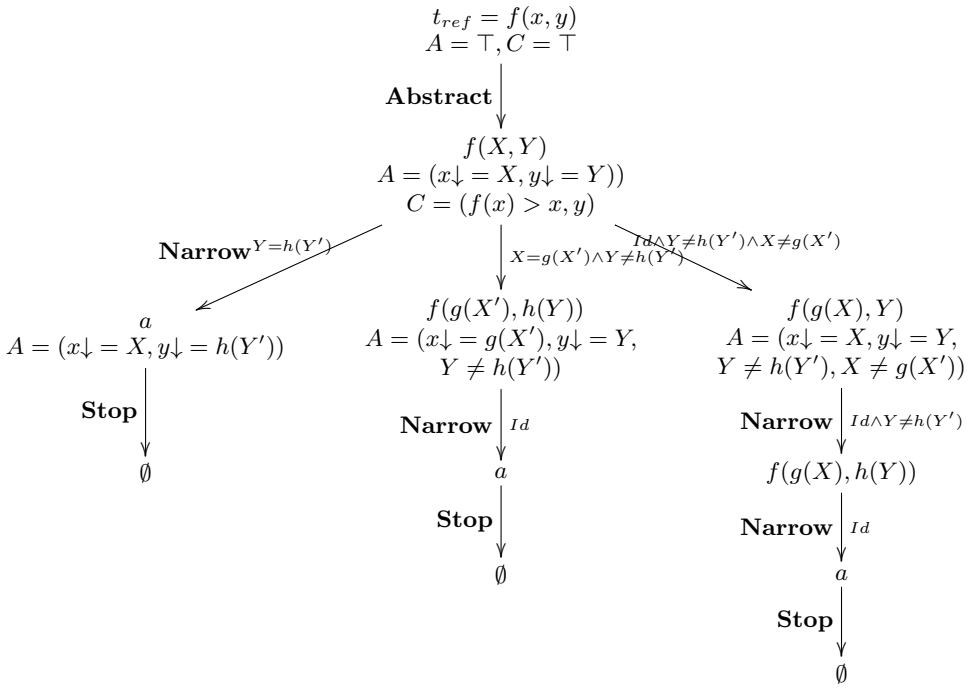
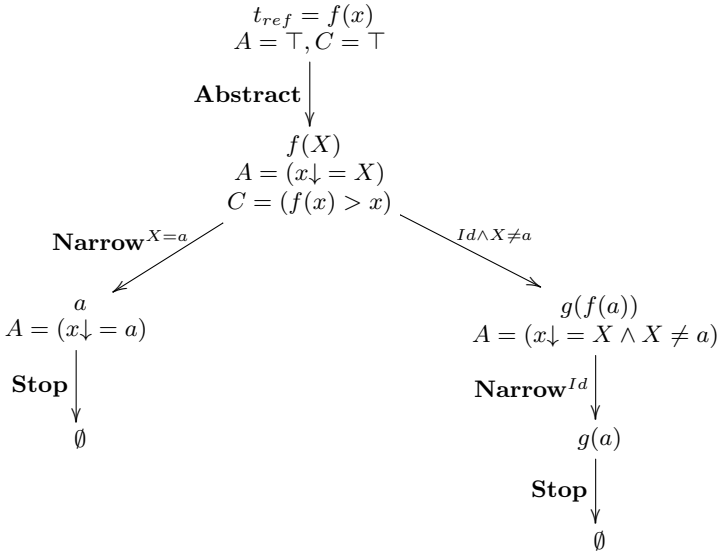


Fig. 5. Proof tree for symbol *or*



Note that in the four 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 Abstraction Constraints for Priority Rewriting

We present in this section an alternative approach to our procedure, dealing with the unsatisfiability of A instead of the satisfiability, and show in comparison why it is suitable for IP-termination. As explained in Sect. 4, 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, and stop the proof process on the irrelevant nodes.

For this, we just have to suppress the satisfiability test of $A \wedge \sigma$ in the condition of **Narrow**, and to add the condition "A is unsatisfiable" as an alternative condition of **Stop**. **Narrow** is then applied with $try{-}skip$ instead of $try{-}stop$.

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 conditions only work on the equalities of A, dealing with the unsatisfiability of A instead of the satisfiability is of particular interest when A involves many negations of substitutions. Testing the satisfiability instead requires to verify that the solutions of the equational part of A verify its disequality part.

Testing the unsatisfiability 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 $n-1$ narrowing substitutions.

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. It is immediate when rules have constant right-hand sides: **Narrow** then generates constant terms, for which the predicate IPT trivially holds.

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. They often have the two characteristics enlightened above. The example we give below has been proposed in [9], for a conference

management system described in [23]. Its termination, due to priority arguments, could not be formally proved until now.

Example 6. 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.

$$\left|\begin{array}{l} aut(q(author(x), SP, pap(x,z)), SUBMIS, u) \quad \to PERMIT \\ aut(q(author(x), SP, pap(x,z)), v, u) \quad \to DENY \\ aut(q(author(x), RSC, pap(x,z)), v, u) \quad \to DENY \\ aut(q(rev(x), w, p), v, conf(x,p)) \quad \to DENY \\ aut(q(rev(x), SR, pap(y,z)), REV, ass(x, pap(y,z))) \quad \to PERMIT \\ aut(q(rev(x), SR, pap(y,z)), v, ass(x, pap(y,z))) \quad \to DENY \\ aut(q(rev(x), RSC, pap(y,z)), MEE, ass(x, pap(y,z))) \quad \to PERMIT \\ aut(q(rev(x), w, pap(x,z)), v, u) \to aut(q(rev(x), w, pap(x,z)), v, conf(x, pap(x,z))) \\ \downarrow aut(x,y,z) \to NAPPLIC\ . \end{array}\right.$$

We apply the strategy $S{-}RULES$ on the initial pattern $t_{ref} = aut(x,y,z)$. The proof tree is given in Fig. 1. We first have an **Abstract** step. Then, a **Narrow** step gives 9 branches, following the 9 above rules. With the constrained narrowing substitutions $\sigma_1, (\sigma_2 \wedge \overline{\sigma_2^1}), \ldots (\sigma_7 \wedge \overline{\sigma_7^1} \wedge \ldots \wedge \overline{\sigma_7^6})$ (where σ_i^j is the mgu of $\sigma_i aut(X,Y,Z)$ with the jth rule), 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 $IPT(\ DENY)$ since $PERMIT$ and $DENY$ are constructor constants. The ninth branch gives the state $NAPPLIC$, on which **Stop** applies too.

The interesting branch is the eighth one, giving the state $aut(q(rev(X'), w, pap(X', Z')), Y, conf(X', pap(X', Z')))$ with the substitution $\sigma_8 = (X = q(rev(X'), w, pap(X', Z')))$ constrained by $\overline{\sigma_8^1} \wedge \ldots \overline{\sigma_8^7}$. To lighten the figure, we only specify this branch in the proof tree.

From this last state, we still apply **Narrow**, with three narrowing possibilities: one, with the fourth rule and the substitution $\sigma_9 = Id$, gives the state $DENY$, on which **Stop** then applies, because we have $IPT(DENY)$. The two other ones are not valid: using the eighth and the ninth rules, we also have the narrowing substitution Id , which becomes empty once constrained by $\overline{\sigma_9}$.

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, 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. To satisfy the ordering constraints, any simplification ordering holds. So this example can 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 [11]. This work is an extension to priority rewriting of an inductive approach given in [16] for proving innermost termination of rewriting.

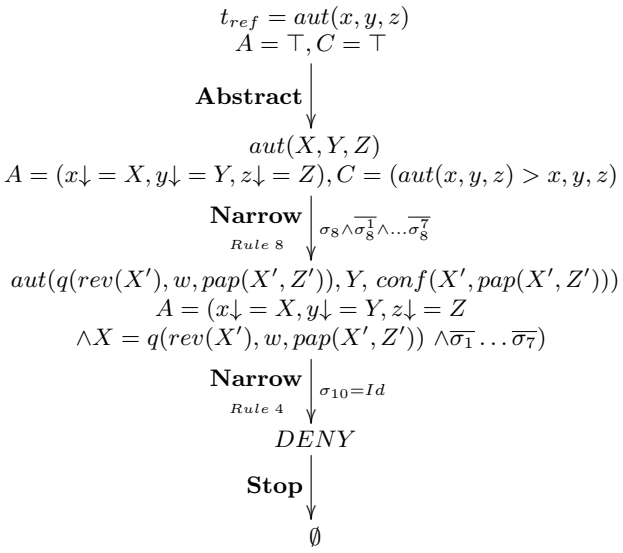


Fig. 6. Proof tree for symbol aut

In our termination proof technique, the priority mechanism localizes in the specific narrowing relation used to model the rewriting relation on ground terms. Moreover, it can be expressed through negations of substitutions, then introducing constraints similar to those already required to model ground innermost rewriting. We then have generalized the innermost narrowing relation introduced in [16], to model the IP-rewriting relation on ground terms and have given a lifting lemma ensuring the correctness of this modelization.

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. When the treatment of the constraints is automatable – sufficient conditions for ordering constraints as well as for abstraction constraints can be given for this – the proof procedure is completely automatable. Considering unsatisfiability of abstraction constraints instead of satisfiability is, in general, particularly suitable for priority rewriting and more precisely for rule-based security policies.

As termination of the original priority rewriting relation of [2] 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, in particular for specifying security policies.

References

1. Baeten, J.C.M., Bergstra, J.A., Klop, J.W.: Term rewriting systems with priorities. In: RTA 1987. Volume 256 of LNCS., Springer Verlag (1987) 83–94

2. Baeten, J.C.M., Bergstra, J.A., Klop, J.W., Weijland, W.P.: Term-rewriting systems with rule priorities. Theoretical Computer Science **67**(2-3) (1989) 283–301
3. van den Brand, M.G., Klint, P., Verhoef, C.: Term Rewriting for Sale. In: WRLA 1998. Volume 15 of ENTCS., Elsevier Science (1998) 139–161
4. Marti-Oliet, N., Meseguer, J., Verdejo, A.: Towards a strategy language for Maude. In: WRLA 2004. Volume 117 of ENTCS., Elsevier Science (2004) 417–441
5. Plasmeijer, R., van Eekelen, M.: Functional Programming and Parallel Graph Rewriting. Addison Wesley (1993)
6. Augustsson, L.: A compiler for lazy ML. In: LFP 1984, New York, NY, USA, ACM (1984) 218–227
7. : Home of Clean http://clean.cs.ru.nl/index.html.
8. : Wiki homepage of Haskell http://www.haskell.org/haskellwiki/Haskell.
9. de Oliveira, A.S.: Réécriture et Modularité pour les Politiques de Sécurité. PhD thesis, Univesité Henri Poincaré, Nancy, France (2008)
10. de Oliveira, A.S., Wang, E.K., Kirchner, C., Kirchner, H.: Weaving rewrite-based access control policies. In: FMSE 2007, ACM (2007) 71–80
11. Mohan, C.K.: Priority rewriting: Semantics, confluence, and conditionals. In: RTA 1989. Volume 355 of LNCS., Springer Verlag (1989) 278–291
12. van de Pol, J.: Operational semantics of rewriting with priorities. Theoretical Computer Science **200**(1-2) (1998) 289–312
13. Sakai, M., Toyama, Y.: Semantics and strong sequentiality of priority term rewriting systems. Theoretical Computer Science **208**(1–2) (1998) 87–110
14. van de Pol, J., Zantema, H.: Generalized innermost rewriting. In: RTA 2005. Volume 3467 of LNCS., Springer (2005) 2–16
15. Moreau, P., Ringeissen, C., Vittek, M.: A Pattern Matching Compiler for Multiple Target Languages. In Hedin, G., ed.: CC 2003. Volume 2622 of LNCS., Springer (May 2003) 61–76
16. Gnaedig, I., Kirchner, H.: Termination of rewriting under strategies. ACM Transactions on Computational Logic **10**(2) (February 2009)
17. Baader, F., Nipkow, T.: Term rewriting and all that. Cambridge University Press (1998)
18. Terese: Term Rewriting Systems. Number 55 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press (2003)
19. Gnaedig, I.: Termination of Priority Rewriting - Extended Version. HAL-INRIA Open Archive Number inria-00349031 (2008)
20. Kruskal, J.B.: Well-quasi ordering, the tree theorem and Vazsonyi's conjecture. Trans. Amer. Math. Soc. **95** (1960) 210–225
21. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata (2007) release October, 12th 2007.
22. Arts, T., Giesl, J.: Proving innermost normalisation automatically. In: RTA 1997. Volume 1232 of LNCS., Springer Verlag (1997) 157–171
23. Dougherty, D.J., Fisler, K., Krishnamurthi, S.: Specifying and reasoning about dynamic access-control policies. In: IJCAR 2006. Volume 4130 of LNCS., Springer (2006) 632–646

Appendix

A The lifting lemma

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

Lemma 2 (Priority Innermost 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:

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

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 $l \to r$ 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, $\alpha s \to^p t'$, 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 $l' \to r' \in \mathcal{R}$, 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 $l' \to r'$, 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 $l' \to r' \in \mathcal{R}$ of higher priority than $l \to r$ 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 $l' \to r'$, which has higher priority than $l \to r$. This is impossible by definition of IP-reducibility of $\alpha(s)$ by $l \to r$ 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 $s \rightsquigarrow^{IP}_{p,l\to r,\sigma} s'$, 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 $SUCCESS(g, \succ)$.*

The proof works like the innermost case of the proof of the termination theorem in [16], by replacing the lifting lemma by the priority innermost lifting lemma. This shows the modular aspect of our approach.