Termination of rewriting with local strategies

Olivier Fissore, Isabelle Gnaedig, Hélène Kirchner

LORIA-INRIA & LORIA-CNRS
P 239 F-54506 Vandœuvre-lès-Nancy Cedex
e-mail: fissore@loria.fr,gnaedig@loria.fr,Helene.Kirchner@loria.fr

Abstract. In this paper, we propose a method for specifically proving termination of rewriting with particular strategies: local strategies on operators. An inductive proof procedure is proposed, based on an explicit induction on the termination property. Given a term, the proof principle relies on alternatively applying the induction hypothesis on its subterms, by abstracting the subterms with induction variables, and narrowing the obtained terms in one step, according to the strategy. The induction relation, an $\mathcal{F}$-stable ordering having the subterm property, is not given a priori, but its existence is checked along the proof, by testing satisfiability of ordering constraints.

Keywords: Rewriting, termination, local strategy on operators, rule-based languages, induction, narrowing, ordering constraints.

1 Introduction

Termination of rewriting is a crucial problem in automated deduction, for equational logic, as well as in programming, for rule-based languages. As it is undecidable in general, it is ensured in particular contexts with sufficient conditions. A lot of termination proof techniques have been proposed, most of them using noetherian orderings on terms. But they usually tackle the property for the standard rewriting relation and essentially work on free term algebras. In the context of rule-based languages such as ASF+SDF [15], OBJ3 [14], Maude [5], CafeOBJ [10], Stratego [19], or ELAN [3], where programs are sets of rules and executions consist in rewriting ground expressions, it would be useful to have more specific termination proof tools: methods allowing to prove termination under specific reduction strategies, or to prove termination on the ground term algebra, for term rewriting systems (TRSs in short) that are not terminating on the free term one. The proof method we propose here, based on an explicit induction on the termination property, enables us to tackle these problems.

In the context of programming, there are sets of rules, that lead to divergent computations when all derivations are considered, but that terminate for particular strategies. A famous example is the evaluation of a recursive function defined with an `if_then_else_` expression, and that can diverge if the first argument is not evaluated first.

Local strategies on operators are used in this context, in particular to force the evaluation of expressions to terminate. This kind of strategy is allowed by languages such that OBJ3, CafeOBJ or Maude, and studied in [7] and [18]. It is defined in the following way: to any operator f is attached an ordered list of integers, giving the positions of the subterms to be evaluated in a given term, whose top operator is f. For example, the TRS

$$2nd(cons(x, cons(y, z))) \to y$$
$$inf(x) \to cons(x, inf(s(x)))$$

does not terminate for the standard rewriting relation, but does wit the following strategy: $LS(cons) = [1]$, $LS(2nd) = [1; 0]$, $LS(inf) = [1; 0]$, $LS(s) = [1]$.

As far as we know, the specific termination problem of rewriting with strategies has only been tackled for the innermost case on free term algebras [1] and for the innermost and the outermost cases for ground term ones [12]. Here, we propose a termination proof method for the case of local strategies on operators, following the induction proof principle proposed in [12]. Note that with our approach we handle the leftmost innermost strategy, which is a particular case of local strategy. Let us also cite termination results for a restricted kind of rewriting called context-sensitive rewriting [17, 16, 20, 11]. In this context, rewriting is allowed only at some specified position in the terms, which is different from local strategies, that are more specific: in the second case, not only allowed rewriting positions are specified, but also the order to consider them. Except for particular cases of local strategies, the two kinds of strategy are distinct.

The main idea of our proof method is to use explicit induction on the termination property in order to prove that any element t of a given set of terms T terminates i.e. there is no infinite derivation chain starting from t. Our induction principle uses an ordering on ground terms having the subterm property. It is based on the simple idea that if reducing a term t according to a given strategy first requires to normalize a subterm t' of t, we can suppose, by induction hypothesis, that t' terminates for the same strategy. If we replace t' by an induction variable X representing any of their normal forms, it then remains to prove that the term u obtained by replacement of t' by X in t is terminating, to prove that t is terminating. A rewriting step is then performed on u following the different possible values of X: it is computed by narrowing. This process is iterated until obtaining a no narrowable term, or a term, the induction hypothesis applies on. Note that the induction ordering is not given a priori but constrained during the proof by setting ordering constraints. Applying the induction hypothesis then lies on testing whether these constraints are satisfiable.

On the previous example, our method consists in proving termination of the constants, and of the terms of the form $s(T), cons(T_1, T_2), inf(T), 2nd(T)$, for the previously given strategy, whatever the values of the ground terms T, T_1, T_2. Let us suppose here we have a constant 0 in the signature. Obviously, 0 is in normal form and then terminating. For $s(T)$, using an induction ordering $\succ$ such that $s(T) \succ T$, by induction hypothesis, we can suppose that T is terminating. So is $s(T)$, since s is a constructor (i.e. s is not a top symbol of left-hand side of rule).

By definition of the strategy, normalizing $cons(T_1, T_2)$ consists in normalizing T_1, into $T_1\downarrow$ if its exists, and then $cons(T_1\downarrow, T_2)$ at the top position. For the same ordering $\succ$ as previously, we have $cons(T_1, T_2) \succ T_1$. Then, by induction hypothesis, T_1 terminates. Let $T_1\downarrow$ be any of its normal forms (there can be several normal forms if the system is not confluent). The termination of $cons(T_1, T_2)$ is then reduced to the termination of $cons(T_1\downarrow, T_2)$, which does not reduce at the top position by definition of the strategy.

Normalizing $2nd(T)$ also first consists in normalizing T. In a similar way than previously, the induction hypothesis can be applied to T. Let $T\downarrow$ be any of its normal forms. Reducing $2nd(T\downarrow)$ at the top position can give two results, according to the form of $T\downarrow$. If $T\downarrow$ is of the form $cons(T_3, cons(T_4, T_5))$, we obtain T_4, which is in normal form since $cons(T_3, cons(T_4, T_5))$ is in normal form. If $T\downarrow$ is not of the form $cons(T_3, cons(T_4, T_5))$, then $2nd(T\downarrow)$ does not rewrite at the top position and then is already in normal form.

Normalizing $inf(T)$ also first consists in normalizing T. Like previously, $T\downarrow$ exists thanks to the induction hypothesis. The resulting term $inf(T\downarrow)$ rewrites into $cons(T\downarrow, inf(s(T\downarrow)))$, whose first subterm has then to be normalized. As this subterm is already in normal form, $cons(T\downarrow, inf(s(T\downarrow)))$ is in normal form for the given strategy. So any ground term is terminating. Our goal here is to provide a procedure implementing such a reasoning.

In Section 2, the background is presented. Section 3 introduces the basic notions formalizing our induction principle. In Section 4, a rule-based algorithm mechanizing the proof principle is given, its correctness is established and examples are given.

2 The background

We assume that the reader is familiar with the basic definitions and notations of term rewriting given for instance in [6]. $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is the set of terms built from a given finite set $\mathcal{F}$ of function symbols having an arity $n \in \mathbb{N}$, and a set $\mathcal{X}$ of variables denoted $x, y \ldots$. $\mathcal{T}(\mathcal{F})$ is the set of ground terms (without variables). The terms composed by a symbol of arity 0 are called constants; $\mathcal{C}$ is the set of constants of $\mathcal{F}$. Positions in a term are represented as sequences of integers. The empty sequence ϵ denotes the top position. The symbol at the top position of a term t is written $top(t)$. Let p and p' be two positions. The position p is said to be prefix of p' (and p' suffix of p) if $p' = p\lambda$, where λ is a non empty sequence of integers. Given a term t, $\mathcal{O}(t)$ is the set of positions in t, inductively defined as follows: $\mathcal{O}(t) = \{\epsilon\}\ if\ t \in \mathcal{X},\ \mathcal{O}(t) = \{\epsilon\} \cup \{i.p \mid 1 \leq i \leq n \text{ and } p \in \mathcal{O}(t_i)\}\ if\ t = f(t_1, \ldots, t_n)$. This set is partitioned into $\overline{\mathcal{O}}(t) = \{p \in \mathcal{O}(t) \mid t|_p \notin \mathcal{X}\}$ and $\mathcal{O}_{\mathcal{V}}(t) = \{p \in \mathcal{O}(t) \mid t|_p \in \mathcal{X}\}$ where the notation $t|_p$ stands for the subterm of t at position p. If $p \in \mathcal{O}(t)$, then $t[t']_p$ denotes the term obtained from t by replacing the subterm at position p by the term t'.

A substitution is an assignment from $\mathcal{X}$ to $\mathcal{T}(\mathcal{F}, \mathcal{X})$, written $\sigma = (x \mapsto t) \ldots (y \mapsto u)$. It uniquely extends to an endomorphism of $\mathcal{T}(\mathcal{F}, \mathcal{X})$. We identify a substitution $\sigma = (x \mapsto t) \ldots (y \mapsto u)$ with the finite set of equations $(x = t) \wedge \ldots \wedge (y = u)$. The result of applying σ to a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is written $\sigma(t)$ or σt. The domain of σ, denoted $Dom(\sigma)$ is the finite subset of $\mathcal{X}$ such that $\sigma x \neq x$. The range of σ, denoted $Ran(\sigma)$, is defined by $Ran(\sigma) = \bigcup_{x \in Dom(\sigma)} Var(\sigma x)$. A ground substitution or instantiation 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 $\sigma_2\sigma_1$. Given two substitutions σ_1 and σ_2, we write $\sigma_1 \leq \sigma_2$ iff $\exists\theta$ such that $\sigma_2 = \theta\sigma_1$. Given a subset $\mathcal{X}_1$ of $\mathcal{X}$, we note $\sigma_{\mathcal{X}_1}$ 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 $\forall x \in Dom(\sigma_{\mathcal{X}_1}) : \sigma_{\mathcal{X}_1} x = \sigma x$.

Given a set R of rewrite rules or term rewriting system on $\mathcal{T}(\mathcal{F}, \mathcal{X})$, a function symbol in $\mathcal{F}$ is called a constructor if it does not occur in R at the top position of the left-hand side of a rule, and is called a defined function symbol otherwise. The set of constructors of $\mathcal{F}$ for R is denoted by $Cons_R$, the set of defined function symbols of $\mathcal{F}$ for R is denoted by Def_R (R is omitted when there is no ambiguity). The rewriting relation induced by R is called standard rewriting relation and is noted $\rightarrow_R$ ($\rightarrow$ if there is no ambiguity on R). We note $s \rightarrow_{p, l \rightarrow r, \sigma} t$ (or $s \rightarrow^{p, l \rightarrow r, \sigma} t$ where either p or $l \rightarrow r$ or σ may be omitted) if s rewrites into t at position p with the rule $l \rightarrow r$ and the substitution σ. The transitive (resp. reflexive transitive) closure of the rewriting relation induced by R is denoted by $\rightarrow_R^+$ (resp. $\rightarrow_R^*$). If it exists, the last term of a finite derivation starting from t is said to be in normal form, and is denoted by $t{\downarrow}$.

An ordering $\succ$ on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is said to be noetherian (or well-founded) iff there is no infinite decreasing chain for this ordering. It is $\mathcal{F}$-stable 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 $\mathcal{T}(\mathcal{F}, \mathcal{X})$, $f(\ldots t \ldots) \succ t$. Note that if $\succ$ is $\mathcal{F}$-stable and has the subterm property, then it is noetherian. If, in addition, $\succ$ is stable by 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. Let t be a term of $\mathcal{T}(\mathcal{F})$; let us recall that t terminates if and only if any rewriting derivation (or derivation chain) starting from t is finite.

3 Induction for termination with local strategies

We now tackle the termination problem for rewriting with local strategies on operators, as expressed in [14] and studied in [7]. A local strategy is defined in the following way.

Definition 1. *An* LS rewriting strategy *(or LS-strategy) on terms of $\mathcal{T}(\mathcal{F}, \mathcal{X})$ (resp. of $\mathcal{T}(\mathcal{F})$) is a function LS from $\mathcal{F}$ to the set of lists of integers $\mathcal{L}(\mathbb{N})$, defining a rewriting strategy as follows.*

Given a LS-strategy such that $LS(f) = [p_1, \ldots, p_k]$, $p_i \in [0..arity(f)]$ for all $i \in [1..k]$, for some symbol $f \in \mathcal{F}$, normalizing a term $t = f(t_1, \ldots, t_m) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ (resp $\in \mathcal{T}(\mathcal{F})$) with respect to $LS(f) = [p_1, \ldots, p_k]$, consists in normalizing all subterms of t at positions $p_1, \ldots, p_k$ successively, according to the strategy. If there exists $i \in [1..k]$ such that $p_1, \ldots, p_{i-1} \neq 0$ and $p_i = 0$, then

- *if the current term t' obtained after normalizing $t|_{p_1}, \ldots, t|_{p_{i-1}}$ is reducible at the top position into a term $g(u_1, \ldots, u_n)$, then $g(u_1, \ldots, u_n)$ is normalized with respect to $LS(g)$ and the rest of the strategy $[p_{i+1}, \ldots, p_k]$ is ignored,*
- *if t' is not reducible at the top position, then t' is normalized with respect to $p_{i+1}, \ldots, p_k$.*

Note that for $x \in \mathcal{X}$, $LS[x] = []$ since a term is not reducible at a variable position.

At each rewriting step, the term t is said to LS-rewrite into a term t'. If t does not rewrite for the LS-strategy, it is said to be in LS-normal form (or in normal form if there is no ambiguity). If any LS-rewriting chain starting from t leads to a LS-normal form then t is said to be LS-terminating (or to LS-terminate). If the evaluation strategy of a term t' is the empty list, then t' is in LS-normal form.

3.1 Induction for local strategies

For proving that a term t of $\mathcal{T}(\mathcal{F})$ LS-terminates, we proceed by induction on $\mathcal{T}(\mathcal{F})$ with a noetherian ordering $\succ$ (more precisely, an $\mathcal{F}$-stable ordering having the subterm property), assuming that for any t' such that $t \succ t'$, t' LS-terminates. We first prove that a basic set of minimal elements for $\succ$ LS-terminates. As the subterm property for $\succ$ is required, the set of minimal elements is a subset of the set of constants of $\mathcal{F}$.

We then consider the case of any term t of $\mathcal{T}(\mathcal{F})$. For that, we observe the rewriting derivation tree for the LS-strategy starting from a term $t_{ref} = g(x_1, \ldots, x_m)$, for any $g \in \mathcal{F}$, where $x_1, \ldots, x_m$ are induction variables that can be instantiated by any ground term. The LS rewriting relation on ground terms is simulated by the two mechanisms below to follow the derivation tree starting from t_{ref}, and whose current term is t. Let $LS(top(t)) = [p_1, \ldots, p_n]$, and p_k the first element of $[p_1, \ldots, p_n]$ such that $p_k = 0$.

- First, the subterms $t|_{p_1}, \ldots, t|_{p_{k-1}}$ of t have to be LS-normalized, by definition of the above LS-strategy. If $t_{ref} \succ t|_{p_1}, \ldots, t|_{p_{k-1}}$ we can suppose, by induction hypothesis, that these subterms are LS-terminating. We then replace them in t by *abstraction variables* X_i representing respectively any of their normal forms $t_i\downarrow$: these variables will only be instantiated by terms in normal form. Reasoning by induction allows us to only suppose the existence of the $t_i\downarrow$ *without explicitly computing them*; this step will be called abstraction step or abstraction of the subterms of t. We also say that t is abstracted into a term v.
- Second, rewriting the resulting term v at position ϵ, following all possible ground instantiations of v. This is computed by a narrowing step on v. Two cases may happen:
  - if v is not narrowable at the top position, the subterms $v|_{p_{k+1}}, \ldots, v|_{p_n}$ of v then have to be LS-normalized, and we try to abstract them like above;
  - if v is narrowable at the top position, the narrowing step is computed with all possible rules and all possible substitutions $\sigma_1, \ldots, \sigma_l$ to give terms $w_1, \ldots, w_l$, that have to be considered respectively with the strategies $LS(top(w_1)), \ldots, LS(top(w_l))$. So the two mechanisms above are again applied on the terms $w_1, \ldots, w_l$. In addition, instances of v that are not considered by the narrowing have to be reduced at the positions $p_{k+1}, \ldots, p_n$. So the two mechanisms described above are also applied on v at positions $p_{k+1}, \ldots, p_n$ for the instances of v that are not instances of $\sigma_i v, i \in [1..l]$.
- The process stops on the current terms t having an empty LS-strategy or on current terms the induction hypothesis can be applied on (i.e. such that $t_{ref} \succ t$; in this case, t is supposed to be LS-terminating).

Note that if there does not exist p_k in $\{p_1, \ldots, p_n\}$ such that $p_k = 0$, then only the first point is processed, abstracting every subterm $t|_{p_i}$ of $t, i \in [1..n]$.

3.2 Abstraction

We now give some new definitions to formalize the above mechanisms. Abstraction needs the use of special variables representing LS-normal forms.

Definition 2. *Let $\mathcal{N}$ be a set of new variables disjoint from $\mathcal{X}$. Symbols of $\mathcal{N}$ are called* NF-variables. *Substitutions and instantiations are extended to $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$ in the following way. Let $X \in \mathcal{N}$; for any substitution σ (resp. instantiation θ) such that $X \in Dom(\sigma)$, σX (resp. θX) is in normal form.*

LS-strategies can be extended to terms of $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$, by stating $LS(X) = []$ for any $X \in \mathcal{N}$.

Note that for abstracting the current term $f(u_1, \ldots, u_m)$, it is not useful to introduce an abstraction variable for the u_j that are ground terms already in normal form, nor for the u_j that are already NF-variables.

Definition 3. *The term $f(u_1, \ldots, u_m)$ is* abstracted *into $f(U_1, \ldots, U_m)$ at positions $\{i_1, \ldots, i_p\} \subseteq [1..m]$ if :*

- *$\{i_1, \ldots, i_p\}$ are the positions of $[1..m]$ such that $u_{i_1}, \ldots, u_{i_p}$ are neither ground terms in normal form, nor NF-variables,*

– *$U_j = X_j$ where X_j is a fresh NF-variable, if $j \in \{i_1, \dots, i_p\}$, $U_i = u_i$ otherwise.*

We will prove LS-termination on $\mathcal{T}(\mathcal{F})$, reasoning on terms with abstraction variables, i.e. on terms of $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$.

3.3 Constraints

Let us now define the different constraints needed by our proof process. Unlike in classical approaches using induction, the induction ordering is not given a priori. Constraints are set along the proof, following the requirements appearing when induction hypotheses have to be applied. Such ordering constraints are cumulated in a set C and the satisfiability of C is tested any time the induction hypotheses have to be applied.

We now formally define the satisfiability of ordering constraints.

Definition 4. An ordering constraint $(t > t')$ *on terms of* $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$ *is* satisfiable *if there exists an ordering $\succ$ and at least one instantiation θ such that $\theta t \succ \theta t'$. We say that $\succ$ and θ satisfy $(t > t')$.*

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

Along our induction process, when abstracting subterms t_i by X_i, we state constraints on NF-variables to express that their instances can only be the normal forms of the corresponding instances of the t_i. They are of the form $t\!\downarrow = X$ where $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, and $X \in \mathcal{N}$, or more generally of the form $t\!\downarrow = t'$ where $t, t' \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$. Let us call such a constraint an abstraction constraint.

Definition 5. *An* abstraction constraint $(t\!\downarrow = t')$ *where $t, t' \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$ is* satisfiable *if there exists at least one instantiation θ such that $\theta t\!\downarrow = \theta t'$. We say that θ satisfies $(t\!\downarrow = t')$.*

A constraint formula *A is a formula of the form $\bigwedge_i (t_i\!\downarrow = t'_i) \bigwedge_j (\bigvee_{k_j} (x_{k_j} \neq u_{k_j}))$, $x_{k_j} \in \mathcal{X} \cup \mathcal{N}, u_{k_j} \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$ and the $(t_i\!\downarrow = t'_i)$ are abstraction constraints. The empty formula is denoted $\top$. A formula A is* satisfiable *if there exists at least one instantiation θ such that $\bigwedge_i (\theta t_i\!\downarrow = \theta t'_i) \bigwedge_j (\bigvee_{k_j} (\theta x_{k_j} \neq \theta u_{k_j}))$. We say that θ satisfies A.*

In this paper, we consider constraint problems composed of 2-tuples (A, C) where A is constraint formula and C is a conjunction of ordering constraints.

Definition 6. *Let A be a constraint formula and C a conjunction of ordering constraints. The constraint problem (A, C) is satisfied by an ordering $\succ$ if A is satisfiable, and for all instantiations θ satisfying A, $\succ$ and θ satisfy C. (A, C) is satisfiable if A is satisfiable and there exists an ordering $\succ$ as above.*

Deciding the satisfiability of (A, C) would require to express all instantiations satisfying A. As we will see later, an interesting point of our method is that we do not need to characterize all those instantiations. It is enough to exhibit one of them to prove the satisfiability of A. In such a case, a sufficient condition for an ordering $\succ$ to satisfy (A, C) is that $\succ$ is stable by substitution (the induction ordering is then a simplification ordering) and $t \succ t'$ for any unequality $t > t'$ of C.

3.4 Narrowing

After the abstraction of the term $f(u_1, \dots, u_m)$ into $f(U_1, \dots, U_m)$ at positions $\{i_1, \dots, i_p\}$, where the u_{i_j} are supposed to have a normal form $u_{i_j}\!\downarrow$, and are replaced by abstraction variables X_{i_j}, we test whether the ground instances of $f(U_1, \dots, U_m)$ are reducible with a case study on the syntactic form of the possible instantiations of the X_{i_j}. This test consists in narrowing $f(U_1, \dots, U_m)$ at position ϵ with all possible substitutions instantiating the X_i only with irreducible terms, and all possible rewrite rules.

Let us now recall the definition of narrowing.

Definition 7. *Let $\mathcal{R}$ be a TRS 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 R and the substitution σ, when σ is a most general unifier of $t|_p$ and l, $t' = \sigma(t[r]_p)$. This is denoted $t \rightsquigarrow_R^{p, l \to r, \sigma} t'$ where either p, or $l \to r$ or σ may be omitted. It is always assumed that there is no variable in common between the rule and the term, i.e. that $Var(l) \cap Var(t) = \emptyset$.*

The requirement of disjoint variables is easily fulfilled by an appropriate renaming of variables in the rules when narrowing is performed. Note that for the most general unifier σ used in the above definition, $Dom(\sigma) \subseteq Var(l) \cup Var(t)$ and we can choose $Ran(\sigma) \cap (Var(l) \cup Var(t)) = \emptyset$, thus introducing in the range of σ only fresh variables. Thus $Var(t) \cap Var(t') = \emptyset$ if in addition, variables of $\mathcal{V}ar(t) - Dom(\sigma)$ are renamed through a substitution denoted σ_{ren}.

Moreover, if narrowing is performed after an abstraction step, the range $Ran(\sigma)$ of the narrowing substitution σ only contains NF-variables. Indeed, abstraction gives a term $f(U_1, \ldots, U_m)$ in which some subterms U_i are abstraction variables $X_{i_1}, \ldots, X_{i_p}$. Thus, on $\{X_{i_1}, \ldots, X_{i_p}\}$, σ is of the form $(X_{i_1} = v_{i_1}, \ldots, X_{i_p} = v_{i_p})$. As $X_{i_1}, \ldots, X_{i_p}$ are NF-variables, all their ground instances must be in normal form, and the same holds for ground instances of the v_i's. So the variables of $v_{i_1}, \ldots, v_{i_p}$ have to be NF-variables.

As we will see below, in our proof process, we will also have to consider the negation of a substitution.

Definition 8. *Let σ be a substitution on $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$ defined by $\bigwedge_i (x_i = t_i)$ $x_i \in \mathcal{X} \cup \mathcal{N}$, $t_i \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$. The* negation *of σ, denoted $\overline{\sigma}$ is the formula $\bigvee_i (x_i \neq t_i)$.*

4 A rule-based algorithm

4.1 The inference rules

Inference rules describing our termination proof mechanism for local strategies work on sets of 4-tuples $T = (\{u\}, [p_1, \ldots, p_m], A, C)$, where:

- $\{u\}$ is a set of terms of $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$, containing the current term u whose ground instances have to be proved LS-terminating. This is either a singleton or the empty set.
- $[p_1, \ldots, p_m]$ is the list of positions with respect to whom the current term u has to be evaluated. This is a sublist of $LS(top(u))$.
- A is a constraint formula. The sub-formulas of the form $u{\downarrow} = X, u \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N}), X \in \mathcal{N}$ are stated each time a subterm u of the current term is abstracted by a new NF-variable X.
- C is a conjunction of ordering constraints completed by the abstraction steps.

Let us now present the inference rules.

- The rule **Abstract** processes the abstracting step. It applies on $(\{f(u_1, \ldots, u_m)\}, [p_1, \ldots, p_n], A, C)$, when there exists $k \in [2..n], p_k = 0$ and $p_1, \ldots, p_{k-1} \neq 0$. The term $u = f(u_1, \ldots, u_m)$ is abstracted at positions $i_1, \ldots, i_p \in \{p_1, \ldots, p_{k-1}\}$ if there exists an $\mathcal{F}$-stable ordering having the subterm property and such that $(A, C \wedge t_{ref} > u_{i_1}, \ldots, u_{i_p})$ is satisfiable. Indeed, by induction hypothesis, all ground instances of $u_{i_1}, \ldots,$ u_{i_p} LS-terminate. So $\{f(u_1, \ldots, u_m)\}$ is replaced by $f(U_1, \ldots, U_m)$. The list of positions then becomes $[0, p_{k+1}, \ldots, p_n]$.
- The rule **Abstract−Stop** processes the abstracting step as above, when there is no position 0 in the strategy of the current term. Any ground instance of the term obtained after abstraction is irreducible, by definition of the LS-strategy, which ends the proof on the current derivation chain. The set containing the current term is then replaced by the empty set.
- The rule **Narrow−Y** processes the narrowing step at position 0 (i.e. the top position) of the current term u. If u is narrowable with a substitution satisfying the current constraint formula A, then u is narrowed in all possible ways in one step, with all possible rewrite rules of the rewrite system R, and all possible substitutions σ_i, into $w_i, i \in [1..l]$. Then $(\{u\}, [0, p_1, \ldots, p_n], A, C)$ is replaced by $\{(\{w_i\}, LS(top(w_i)), \sigma_i A, C), i \in [1..l]\}$, where σ_i is the most general substitution allowing narrowing of u into terms w_i. This narrowing step means that $\sigma_1 u, \ldots, \sigma_l u$ are all the instances of u that are reducible at the top position. It involves that if $\Phi = \overline{\sigma_1} \wedge \ldots \wedge \overline{\sigma_l}$ is satisfiable, for each substitution μ satisfying Φ, μu is not reducible at the top position. Then, as these μu have to be reduced at positions $[p_1, \ldots, p_n]$, if Φ is satisfiable, to the previous set we must add the set : $(\{u\}, [p_1, \ldots, p_n], A \bigwedge_{i=1}^{l} \overline{\sigma_i}, C)\}$. Note that if $\exists i$ such that σ_i is just a renaming of variables, then $\Phi = \emptyset$. Moreover, since the set of variables occuring in A is disjoint from the set of variables occuring in the rewrite rules, we had rather consider $\overline{\sigma_{i\mathcal{V}ar(u)}}$ instead of $\overline{\sigma_i}$.

Table 1. Inference rules for t_{ref} LS-termination

Abstract:
$$\frac{T \cup \{(\{f(u_1,\dots,u_m)\},\ \ [p_1,\dots,p_n],\ \ A,\ \ C)\}}{T \cup \{(\{f(U_1,\dots,U_m)\},\ \ [0,p_{k+1},\dots,p_n],\ \ A \bigwedge_{i\in\{i_1,\dots,i_p\}} (u_i{\downarrow} = X_i),\ \ C \bigwedge_{i\in\{i_1,\dots,i_p\}} t_{ref} > u_i)\}}$$

where $f(u_1,\dots,u_m)$ is abstracted by $f(U_1,\dots,U_m)$ at the positions $i_1,\dots,i_p \in \{p_1,\dots,p_{k-1}\}$
if $\exists k \in [2..n] : p_1,\dots,p_{k-1} \neq 0, p_k = 0$ and $(A, C \bigwedge_{i\in\{i_1,\dots,i_p\}} t_{ref} > u_i)$ is satisfiable

Abstract-Stop:
$$\frac{T \cup \{(\{f(u_1,\dots,u_m)\},\ \ [p_1,\dots,p_n],\ \ A,\ \ C)\}}{T \cup \{(\emptyset,\ \ [],\ \ A,\ \ C \bigwedge_{i\in[i_1..i_p]} t_{ref} > u_i)\}}$$

where $f(u_1,\dots,u_m)$ can be abstracted by $f(U_1,\dots,U_m)$ at the positions $i_1,\dots,i_p \in \{p_1,\dots,p_n\}$
if $p_1,\dots,p_n \neq 0$ and $(A, C \bigwedge_{i\in[i_1..i_p]} t_{ref} > u_i)$ is satisfiable

Narrow-Y:
$$\frac{T \cup \{(\{f(u_1,\dots,u_m)\},\ \ [0,p_1,\dots,p_n],\ \ A,\ \ C)\}}{T \cup_{i\in[1..l]} \{(\{w_i\},\ \ LS(top(w_i)),\ \ \sigma_i A,\ \ C)\} \cup COMPL}$$

if $\exists\sigma$ such that $f(u_1,\dots,u_m) \leadsto_{\epsilon,\sigma} w$ and σA is satisfiable
where $w_i, i \in [1..l]$, are all terms such that $f(u_1,\dots,u_m) \leadsto_{\epsilon,\sigma_i} w_i$ and $\sigma_i A$ is satisfiable,

$$COMPL = \begin{cases} \{(\{f(u_1,\dots,u_m)\},\ \ [p_1..p_n],\ \ A \bigwedge_{i=1}^{l} \overline{\sigma_i \mathcal{V}ar(f(u_1,\dots,u_n))},\ \ C)\} & \text{if } (A \bigwedge_{i=1}^{l} \overline{\sigma_i \mathcal{V}ar(f(u_1,\dots,u_n))}) \text{ satisfiable} \\ \emptyset & \text{otherwise.} \end{cases}$$

Narrow-N:
$$\frac{T \cup \{(\{f(u_1,\dots,u_m)\},\ \ [0,p_1,\dots,p_n],\ \ A,\ \ C)\}}{T \cup \{(\{f(u_1,\dots,u_m)\},\ \ [p_1,\dots,p_n],\ \ A,\ \ C)\}}$$

if $f(u_1,\dots,u_m)$ is not narrowable at the top position
or $\forall\sigma$ narrowing substitution of $f(u_1,\dots,u_m)$ at the top position , σA is not satisfiable.

Stop-Ind:
$$\frac{T \cup \{(\{u\},\ \ [p_1,\dots,p_n],\ \ A,\ \ C)\}}{T \cup \{(\emptyset,\ \ [],\ \ A,\ \ C \wedge t_{ref} > u)}$$

if $(A, C \wedge t_{ref} > u)$ is satisfiable

Stop-A:
$$\frac{T \cup \{(\{f(u_1,\dots,u_m)\},\ \ [p_1,\dots,p_n],\ \ A,\ \ C)\}}{T \cup \{(\{\sharp\},\ \ [],\ \ A,\ \ C)\}}$$

if $p_1 \neq 0$ and neither Abstract nor Abstract$-$Stop applies.

Stop:
$$\frac{T \cup \{(\{f(u_1,\dots,u_m)\},\ \ [],\ \ A,\ \ C)\}}{T \cup \{(\emptyset,\ \ [],\ \ A,\ \ C)\}}$$

– The rule **Narrow−N** handles the case where u is not narrowable at position 0 or is narrowable with a substitution that does not satisfy the current constraint formula A. Then no narrowing is processed and the current term is evaluated at positions following the top position in the strategy. The list of positions then becomes $[p_1, \ldots, p_n]$.
– We also can test for the current term whether there exists an ordering having the subterm property such that $(A, C \wedge t_{ref} > u)$ is satisfiable. Then, by induction hypothesis, any ground instance of u terminates for the LS-strategy, which ends the proof on the current derivation chain. The **Stop−Ind** rule then replaces the set containing the current term by the empty set.
– The rule **Stop−A** allows to stop the inference process when neither **Abstract** nor **Abstract−Stop** applies, replacing u by the particular symbol $\sharp$.
– The rule **Stop** allows to stop the inference process when the list of positions is empty.

The set of inference rules is given in Table 1.

Once **Abstract** is applied, the evaluation list's first element is 0, so the only rule that applies then is one of {**Narrow−Y**, **Narrow−N**}. When **Narrow−Y** does not apply, **Narrow−N** applies. When **Abstract** does not apply, and the evaluation list's first element is not 0, either **Abstract−Stop** or **Stop−A** applies, and then no rule applies anymore. The strategy for applying these rules is:

repeat*(**Stop**; **Stop−Ind**; **Abstract**; **Abstract−Stop**; **Stop−A**; (**Narrow−Y**/**Narrow−N**))

where ";" expresses the sequential application of the rules, $r_1/\ldots/r_n$ expresses that the rules $r_1, \ldots, r_n$ are mutually exclusive and that one of them will be applied, and repeat* $(r_1; \ldots; r_n)$ stops if none of the r_i applies anymore.

We write $SUCCESS(g, \succ)$ if application of the inference rules on $(\{g(x_1, \ldots, x_m)\}, LS(g), \top, \top)$, whose conditions are satisfied by $\succ$, gives a state of the form $(\emptyset, [], A, C)$ on every branch of the derivation tree.

Theorem 1. *Let R be a TRS on $\mathcal{T}(\mathcal{F}, \mathcal{X})$, and $LS : \mathcal{F} \mapsto L(\mathbb{N})$ a LS-strategy such that the constants of $\mathcal{F}$ LS-terminate. If there exists an $\mathcal{F}$-stable ordering $\succ$ having the subterm property, such that for every non constant defined symbol g, $SUCCESS(g, \succ)$, then every term of $\mathcal{T}(\mathcal{F})$ LS-terminates.*

Note that if there exists a simplification ordering $\succ$ such that $l \succ r$ for any rewrite rule $l \to r$ of a TRS $\mathcal{R}$, then we have $SUCCESS(g, \succ')$, for any defined symbol $g \in \mathcal{D}ef_R$ and any $\mathcal{F}$-stable ordering $\succ'$ having the subterm property. Indeed, for any $t_{ref} = g(x_1, \ldots, x_m)$, we have $\mathcal{U}(t_{ref}) = \mathcal{R}$, with every rule oriented by $\succ$. Then, thanks to Proposition 1, we have $TERMIN(t_{ref})$, and then **Stop−Ind** applies.

4.2 Extending the induction principle

When the induction hypothesis cannot be applied on a term u, the inductive reasoning can be completed as follows. It can sometimes be possible to prove termination of any ground instance of u_i (resp. u) by another way. Let $TERMIN(u)$ be a predicate that is true iff any ground instance of u LS-terminates. In the first, second and fifth previous inference rules we can then replace the condition $t > u_i$ for some i (resp. $t > u$) by the alternative predicate $TERMIN(u_i)$ (resp. $TERMIN(u)$). Obviously, in this case, the ordering constraint $t > u_i$ (resp. $t > u$) is not added to C.

As in [13], for establishing that $TERMIN(u)$ is true, in some cases, the notion of usable rules can be used. Given a TRS $\mathcal{R}$ on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$, we determine the only rewrite rules that are likely to apply to any of its ground instances, for the standard rewriting relation, until its ground normal form is reached, if it exists. Then we try to find a simplification ordering $\succ_N$ so that these rules are oriented. Thus any ground instance αt is bound to terminate for the standard rewriting relation: indeed, if $\alpha t \to t_1 \to t_2 \to \ldots$, then, thanks to the previous hypotheses, $\alpha t \succ_N t_1 \succ_N t_2 \succ_N \ldots$ and, since the ordering $\succ_N$ is noetherian, the rewriting chain cannot be infinite. More formally, given a TRS $\mathcal{R}$, we call *usable rules* of a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$, as in [2], a calculable superset of the set of rules of R used in all possible standard derivations starting from t, and defined as follows. When t is a variable of $\mathcal{X}$, then the usable rules of t are $\mathcal{R}$ itself. Moreover, the set of usable rules associated to a NF-variable is empty, since the only possible instances of such a variable are ground terms in normal form.

Definition 9. *Let $\mathcal{R}$ be a TRS on a set $\mathcal{F}$ of symbols. Let $Rls(f) = \{l \to r \in \mathcal{R} \mid top(l) = f\}$. 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* $t \in \mathcal{N}$,
- $\mathcal{U}(f(u_1, \ldots, u_n)) = Rls(f) \bigcup_{i=1}^{n} \mathcal{U}(u_i) \bigcup_{l \to r \in Rls(f)} \mathcal{U}(r)$.

Lemma 1. *[13] Let $\mathcal{R}$ be a TRS on a set $\mathcal{F}$ of symbols and $t \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$. Whatever αt ground instance of t and $\alpha t \to_{p_1, l_1 \to r_1} t_1 \to_{p_2, l_2 \to r_2} t_2 \to \ldots \to_{p_n, l_n \to r_n} t_n$ rewrite chain starting from αt, then $l_i \to r_i \in \mathcal{U}(t)$, $\forall i \in [1..n]$.*

We then can give a sufficient criterion for ensuring termination for the standard rewriting relation (and then LS-termination) of any ground instance of a term t.

Proposition 1. *[13] Let $\mathcal{R}$ be a TRS on a set $\mathcal{F}$ of symbols, and t a term of $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{N})$. If there exists a simplification ordering $\succ$ such that $\forall l \to r \in \mathcal{U}(t) : l \succ r$, then any ground instance of t is terminating.*

Let us now illustrate our method on the example given in the Introduction.

Example 1. Let R be the following TRS. We prove that R is terminating on $\mathcal{T}(\mathcal{F})$ ($\mathcal{F} = \{cons : 2, 2nd : 1, inf : 1, s : 1, 0 : 0\}$) with the following evaluation strategy : $LS(cons) = [1]$, $LS(2nd) = [1; 0]$, $LS(inf) = [1; 0]$, $LS(s) = [1]$.

$$\begin{array}{ll} 2nd(cons(x, cons(y, z))) & \to y \\ inf(x) & \to cons(x, inf(s(x))) \end{array}$$

The defined symbols of $\mathcal{F}$ are $2nd$ and inf. The term 0 is obviously terminating. Applying the rules on $2nd(x_1)$, we get :

$$\begin{array}{ll} 2nd(x_1) & [1;0] \\ & A = \top \\ & C = \top \\ \textbf{Abstract} & \\ 2nd(X_1) & [0] \\ & A = (x_1{\downarrow} = X_1) \\ & C = (2nd(x_1) > x_1) \\ \textbf{Narrow}-\textbf{Y} & \\ X_3 & [] \\ & \sigma = (X_1 = cons(X_2, cons(X_3, X_4)) \\ & \qquad \wedge x = X_2 \wedge y = X_3 \wedge z = X_4) \\ & A = (x_1{\downarrow} = cons(X_2, cons(X_3, X_4))) \\ & C = (2nd(x_1) > x_1) \\ 2nd(X_1) & [] \\ & A = (x_1{\downarrow} = X_1) \\ & \qquad \wedge (X_1 \neq cons(X_2, cons(X_3, X_4))) \\ & C = (2nd(x_1) > x_1) \end{array}$$

The first constraint formula A can be satisfied by $\theta = (x_1 = cons(0, cons(0, 0)) \wedge X_2 = 0 \wedge X_3 = 0 \wedge X_4 = 0)$, the second one by $\theta = (x_1 = 0 \wedge X_1 = 0)$.

$$\begin{array}{ll} \textbf{Stop}\text{ (twice)} & \\ \emptyset & [] \\ & A = (x_1{\downarrow} = cons(X_2, cons(X_3, X_4))) \\ & C = (2nd(x_1) > x_1) \\ \emptyset & [] \\ & A = (x_1{\downarrow} = X_1 \wedge X_1 \neq cons(X_2, cons(X_3, X_4))) \\ & C = (2nd(x_1) > x_1) \end{array}$$

Applying the rules on $inf(x_1)$, we get :

$$\begin{array}{ll}
inf(x_1) & [1;0] \\
 & A = \top \\
 & C = \top \\
\textbf{Abstract} & \\
inf(X_1) & [0] \\
 & A = (x_1{\downarrow} = X_1) \\
 & C = (inf(x_1) > x_1) \\
\textbf{Narrow}-\textbf{Y} & \\
cons(X_2, inf(s(X_2))) & [1] \\
 & \sigma = (X_1 = X_2 \wedge x = X_2) \\
 & A = (x_1{\downarrow} = X_2) \\
 & C = (inf(x_1) > x_1)
\end{array}$$

The constraint formula A is easily satisfied with the substitution $\theta = (x_1 = 0 \wedge X_2 = 0)$.

$$\begin{array}{ll}
\textbf{Abstract} & \\
cons(X_2, inf(s(X_2))) & [] \\
 & A = (x_1{\downarrow} = X_2) \\
 & C = (inf(x_1) > x_1) \\
\textbf{Stop} & \\
\emptyset & [] \\
 & A = (x_1{\downarrow} = X_2) \\
 & C = (inf(x_1) > x_1)
\end{array}$$

Note that the substitution of narrowing used in the application of **Narrow−Y** merely consists of a renaming of variable. Therefore all the instances of $inf(X_1)$ are covered, and there is no need to define an extra derivation.

Another example is the famous case of the conditional expression.

Example 2. Let us consider the following system $\mathcal{R}$ built on $\mathcal{F} = \{f : 1, zero : 1, if\_then\_else\_ : 3, h : 1, s : 1, i : 1, g : 1, 0 : 0\}$, and denote $if\_then\_else\_$ by ite for short :

$$\begin{array}{ll}
f(i(x)) & \rightarrow ite(zero(x), g(x), f(h(x))) \\
zero(0) & \rightarrow true \\
zero(s(x)) & \rightarrow false \\
ite(true, x, y) & \rightarrow x \\
ite(false, x, y) & \rightarrow y \\
h(0) & \rightarrow i(0) \\
h(x) & \rightarrow s(i(x))
\end{array}$$

The LS-strategy is the following : $LS(ite) = [1;0]$, $LS(f) = LS(zero) = LS(h) = [1;0]$ and $LS(g) = LS(i) = [1]$.

Before showing LS-termination of $\mathcal{R}$, let us note a few points :

- because of the first rule and the last one, $\mathcal{R}$ cannot be oriented;
- $\mathcal{R}$ does not terminate in general; indeed, it suffices to consider the following derivation :
  $f(i(0)) \rightarrow_\epsilon$
  $ite(zero(0), g(0), f(h(0))) \rightarrow_{3.1}$
  $ite(zero(0), g(0), f(i(0))) \rightarrow \ldots$

Obviously, the constant 0 LS-terminates. Applying the inference rules on $f(x_1)$, we get :

$$
\begin{array}{ll}
f(x_1) & [1;0] \\
 & A = \top \\
 & C = \top \\
\textbf{Abstract} & \\
f(X_1) & [0] \\
 & A = (x_1{\downarrow} = X_1) \\
 & C = (f(x_1) > x_1)
\end{array}
$$

Abstract applies, since C is satisfiable by any ordering having the subterm property. A is satisfiable with any instantiation θ such that $\theta x_1 = \theta X_1 = 0$.

$$
\begin{array}{ll}
\multicolumn{2}{l}{\textbf{Narrow}-\textbf{Y}} \\
\multicolumn{2}{l}{ite(zero(X_2), g(X_2), f(h(X_2)))\ \sigma = (X_1 = i(X_2)} \\
 & \quad\quad \wedge\, x' = X_2) \\
 & [1;0] \\
 & A = (x_1{\downarrow} = i(X_2)) \\
 & C = (f(x_1) > x_1) \\
f(X_1) & [] \\
 & A = (x_1{\downarrow} = X_1) \\
 & \quad\quad \wedge\, (X_1 \neq i(X_2)) \\
 & C = (f(x_1) > x_1)
\end{array}
$$

Note that x' comes from the renaming of x in the first rule. Here, the first constraint formula A is satisfiable by any instantiation θ such that $\theta X_2 = 0$ and $\theta x_1 = i(0)$. The second constraint formula is satisfied by any instantiation θ such that $\theta x_1 = \theta X_1 = \theta X_2 = 0$.

Narrow−Y expresses the fact that $\sigma f(X_1)$ is reducible if σ is such that $\sigma X_1 = i(X_2)$, and that the other instances ($\sigma' f(X_1)$ with $\sigma' X_1 \neq i(X_2)$) cannot be reduced.

$$
\begin{array}{ll}
\multicolumn{2}{l}{\textbf{Stop}} \\
\multicolumn{2}{l}{ite(zero(X_2), g(X_2), f(h(X_2)))\ \sigma = (X_1 = i(X_2)} \\
 & \quad\quad \wedge\, x' = X_2) \\
 & [1;0] \\
 & A = (x_1{\downarrow} = i(X_2)) \\
 & C = (f(x_1) > x_1) \\
\emptyset & [] \\
 & A = (x_1{\downarrow} = X_1) \\
 & \quad\quad \wedge\, (X_1 \neq i(X_2)) \\
 & C = (f(x_1) > x_1)
\end{array}
$$

Stop applies, ending the second branch.

$$
\begin{array}{ll}
\multicolumn{2}{l}{\textbf{Abstract}} \\
ite(X_3, g(X_2), f(h(X_2))) & [0] \\
 & A = (x_1{\downarrow} = i(X_2) \\
 & \quad\quad \wedge\, zero(X_2){\downarrow} = X_3) \\
 & C = (f(x_1) > x_1) \\
\emptyset & [] \\
 & A = (x_1{\downarrow} = X_1) \wedge (X_1 \neq i(X_2)) \\
 & C = (f(x_1) > x_1)
\end{array}
$$

The constraint formula A is satisfiable with any instantiation θ such that $\theta X_2 = 0$, $\theta X_3 = true$ and $\theta x_1 = i(0)$.

Abstract applies here, since $zero(X_2)$ can be abstracted, thanks to Proposition 1. Indeed, $\mathcal{U}(zero(X_2)) = \{zero(0) \rightarrow true, zero(s(x)) \rightarrow false\}$, and both rules can be oriented by a LPO $\succ$ with the precedence $zero \succ_{\mathcal{F}} true$ and $zero \succ_{\mathcal{F}} false$. Then we have $TERMIN(zero(X_2))$.

Narrow−Y

$g(X_4)$	$\sigma = (X_3 = true \wedge X_2 = X_4$ $\wedge\, x'' = g(X_4) \wedge y'' = f(h(X_4)))$ $[1]$ $A = (x_1{\downarrow} = i(X_4)$ $\wedge\, zero(X_4){\downarrow} = true)$ $C = (f(x_1) > x_1)$
$f(h(X_4))$	$\sigma = (X_3 = false \wedge X_2 = X_4$ $\wedge\, x'' = g(X_4) \wedge y'' = f(h(X_4)))$ $[1; 0]$ $A = (x_1{\downarrow} = i(X_4)$ $\wedge\, zero(X_4){\downarrow} = false)$ $C = (f(x_1) > x_1)$
$ite(X_3, g(X_2), f(h(X_2))$	$[]$ $A = (x_1{\downarrow} = i(X_2) \wedge zero(X_2){\downarrow} =$ $X_3) \wedge (X_3 \neq true \wedge X_3 \neq false)$ $C = (f(x_1) > x_1)$
$\emptyset$	$[]$ $A = (x_1{\downarrow} = X_1) \wedge (X_1 \neq i(X_2))$ $C = (f(x_1) > x_1)$

The first constraint formula A is satisfiable by any instantiation θ such that $\theta X_4 = 0$ and $\theta x_1 = i(0)$. The second one is satisfiable by any instantiation θ such that $\theta X_4 = s(0)$ and $\theta x_1 = i(s(0))$. The third one is satisfiable by any instantiation θ such that $\theta X_3 = zero(i(0))$, $\theta X_2 = i(0)$ and $\theta x_1 = i(i(0))$.

The following step ends the third branch, whose strategy evaluation list is empty.

Stop

$g(X_4)$	$[1]$ $A = (x_1{\downarrow} = i(X_4) \wedge zero(X_4){\downarrow} = true)$ $C = (f(x_1) > x_1)$
$f(h(X_4))$	$[1; 0]$ $A = (x_1{\downarrow} = i(X_4) \wedge zero(X_4){\downarrow} = false)$ $C = (f(x_1) > x_1)$
$\emptyset$	$[]$ $A = (x_1{\downarrow} = i(X_2) \wedge zero(X_2){\downarrow} = X_3)$ $\wedge\, (X_3 \neq true \wedge X_3 \neq false)$ $C = (f(x_1) > x_1)$
$\emptyset$	$[]$ $A = (x_1{\downarrow} = X_1) \wedge (X_1 \neq i(X_2))$ $C = (f(x_1) > x_1)$

Abstract (*twice*)

$g(X_4)$	$[]$ $A = (x_1{\downarrow} = i(X_4) \wedge zero(X_4){\downarrow} = true)$ $C = (f(x_1) > x_1)$
$f(X_5)$	$[0]$ $A = (x_1{\downarrow} = i(X_4) \wedge zero(X_4){\downarrow} = false$ $\wedge\, h(X_4){\downarrow} = X_5)$ $C = (f(x_1) > x_1)$

$$\begin{array}{l} \emptyset\ [] \\ \quad A = (x_1{\downarrow} = i(X_2) \wedge zero(X_2){\downarrow} = X_3) \\ \qquad\quad \wedge (X_3 \neq true \wedge X_3 \neq false) \\ \quad C = (f(x_1) > x_1) \\ \emptyset\ [] \\ \quad A = (x_1{\downarrow} = X_1) \wedge (X_1 \neq i(X_2)) \\ \quad C = (f(x_1) > x_1) \end{array}$$

Abstract trivially applies on $g(X_4)$: since X_4 is an NF-variable, there is no need to abstract it.

The second **Abstract** applies on $f(h(X_4))$, thanks to Proposition 1. Indeed, $\mathcal{U}(h(X_4)) = \{h(0) \rightarrow i(0), h(x) \rightarrow s(i(x))\}$, and both rules can be oriented by the same LPO as previously with the additional precedence $h \succ_{\mathcal{F}} i$ and $h \succ_{\mathcal{F}} s$. Then we have $TERMIN(h(X_4))$.

The first constraint formula has not changed, while the second one is now satisfiable by any instantiation θ such that $\theta X_5 = s(i(s(0)))$, $\theta X_4 = s(0)$ and $\theta x_1 = i(s(0))$.

$$\begin{array}{ll} \textbf{Narrow}-\textbf{N}\ (\textit{on the second branch}) & \\ g(X_4) & [] \\ & A = (x_1{\downarrow} = i(X_4) \wedge zero(X_4){\downarrow} = true) \\ & C = (f(x_1) > x_1) \\ f(X_5) & [] \\ & A = (x_1{\downarrow} = i(X_4) \wedge zero(X_4){\downarrow} = false \\ & \qquad \wedge h(X_4){\downarrow} = X_5) \\ & C = (f(x_1) > x_1) \\ \emptyset & [] \\ & A = (x_1{\downarrow} = i(X_2) \wedge zero(X_2){\downarrow} = X_3) \\ & \qquad \wedge (X_3 \neq true \wedge X_3 \neq false) \\ & C = (f(x_1) > x_1) \\ \emptyset & [] \\ & A = (x_1{\downarrow} = X_1) \wedge (X_1 \neq i(X_2)) \\ & C = (f(x_1) > x_1) \end{array}$$

One could have tried to narrow $f(X_5)$, by using the first rule and the narrowing substitution $\sigma = (X_5 = i(X_6) \wedge x''' = X_6)$. But then $\sigma A = (x_1{\downarrow} = i(X_4) \wedge zero(X_4){\downarrow} = false \wedge h(X_4){\downarrow} = i(X_6))$. For any θ satisfying A, θ must be such that $\theta h(X_4){\downarrow} = h(\theta X_4{\downarrow}){\downarrow} = i(\theta X_6)$. If $\theta X_4{\downarrow} \neq 0$, then, according to $\mathcal{R}$, $h(\theta X_4{\downarrow}) \rightarrow s(i(\theta X_4{\downarrow}))$, where s is a constructor. Then we cannot have $h(\theta X_4{\downarrow}){\downarrow} = i(\theta X_6)$, and therefore θ must be such that $\theta X_4{\downarrow} = 0$. But then $\theta zero(X_4){\downarrow} = true$, which makes A unsatisfied. Therefore there is no narrowing.

The process is ended by a double application of **Stop**.

$$\begin{array}{ll} \textbf{Stop}\ (\textit{twice}) & \\ \emptyset & [] \\ & A = (x_1{\downarrow} = i(X_4) \wedge zero(X_4){\downarrow} = true) \\ & C = (f(x_1) > x_1) \\ \emptyset & [] \\ & A = (x_1{\downarrow} = i(X_4) \wedge zero(X_4){\downarrow} = false \\ & \qquad \wedge h(X_4){\downarrow} = X_5) \\ & C = (f(x_1) > x_1) \\ \emptyset & [] \\ & A = (x_1{\downarrow} = i(X_2) \wedge zero(X_2){\downarrow} = X_3) \\ & \qquad \wedge (X_3 \neq true \wedge X_3 \neq false) \\ & C = (f(x_1) > x_1) \\ \emptyset & [] \\ & A = (x_1{\downarrow} = X_1) \wedge (X_1 \neq i(X_2)) \\ & C = (f(x_1) > x_1) \end{array}$$

As for the defined symbols $ite, zero, h$, the inference rules apply successfully through one **Abstract**, **Narrow−Y**, **Abstract** without abstraction, **Narrow−N** and **Stop** application. Therefore $\mathcal{R}$ is LS-terminating.

5 Conclusion

In this paper, we have proposed a method to prove termination of term rewriting with local strategies on operators by explicit induction on the termination property. Our method works on the ground term algebra using as induction relation an $\mathcal{F}$-stable ordering having the subterm property. The general proof principle relies on the simple idea that for establishing termination of a ground term t, it is enough to suppose that terms smaller than t for this ordering are terminating, and that rewriting the context leads to terminating chains. Iterating this process until obtaining a context which is not reducible anymore establishes the termination of t.

More precisely, the method is applied on terms of the form $g(x_1, \ldots, x_m)$, where g is a defined symbol, and consists in iterating the application of two steps: an abstraction step, replacing immediate subterms by NF-variables, representing any of their normalized instances, and a narrowing step, reducing the resulting term according to the different possible instances of the NF-variables. These two steps are iterated until getting a term for which one can easily say that all ground instances are terminating. The important point to automatize our proof principle is the satisfaction of the ordering constraints for **Abstract** , **Abstract−Stop** and **Stop−Ind**.

On many examples as those shown in the paper, this is immediate since they are ensured by the subterm property. In many other cases, a LPO is sufficient to satisfy these constraints. Note that such an LPO does not suffice when it is used in the classical way (any left-hand side of rule is greater than any corresponding right-hand side) since we can handle systems that are not terminating for standard rewriting. Testing satisfiability of a constraint formula A remains simple to handle in practice.

We now have a semi-automatic implementation of the inference rules and the strategy for the leftmost innermost case, that can be expressed by a local strategy on operators. It has been implemented in ELAN [4], which is a logical environment for specifying and prototyping deduction systems in a rule based language with strategies. In this case, as any list of reduction positions ends with the top position, the formula A never contains disequations. Sufficient conditions are implemented to detect unsatisfiability of A, by identifying the reducible right-hand sides. The subterm property of the induction ordering to be found is also implemented, allowing the first application of the rule **Abstract** to be completely automatic. Given a TRS, the program interacts with the user and builds the derivation tree resulting from the application of the inference rules according to the strategy we have defined in this paper [8]. Execution examples are available [1]. We also have proposed a variant of the previous implementation, reducing the interaction with the user, by ignoring the satisfiability problem of A. In this case, the obtained proof derivation tree contains the tree we would obtain in using A: states for which A is not satisfiable just correspond to empty sets of ground terms. We thus have in general more computations, but with considerably less user interactions. Moreover, for many examples, the subterm property is the only required property on the induction ordering, so there are no user interactions to test the satisfiability of C and the algorithm is completely automatic. We are now working on automatizable sufficient conditions for proving satisfiability of A.

Proofs and additional examples can be found in the full version of the paper [9].

References

1. T. Arts and J. Giesl. Proving innermost normalisation automatically. In *Proceedings 8th Conference on Rewriting Techniques and Applications, Sitges (Spain)*, volume 1232 of *Lecture Notes in Computer Science*, pages 157–171. Springer-Verlag, 1997.
2. T. Arts and Giesl. J. Proving innermost normalization automatically. Technical Report 96/39, Technische Hochschule Darmstadt, Germany, 1996.
3. P. Borovanský, C. Kirchner, H. Kirchner, P.-E. Moreau, and Ch. Ringeissen. An Overview of ELAN. In C. Kirchner and H. Kirchner, editors, *Proc. Second Intl. Workshop on Rewriting Logic and its Applications*, Electronic Notes in Theoretical Computer Science, Pont-à-Mousson (France), September 1998. Elsevier.

[1] `http://www.loria.fr/~fissore/Demo/description.html`

4. Peter Borovanský, Claude Kirchner, Hélène Kirchner, Pierre-Etienne Moreau, and Christophe Ringeissen. An overview of ELAN. In Claude Kirchner and Hélène Kirchner, editors, *Proceedings of the second International Workshop on Rewriting Logic and Applications*, volume 15, `http://www.elsevier.nl/locate/entcs/volume15.html`, Pont-à-Mousson (France), September 1998. Electronic Notes in Theoretical Computer Science. Report LORIA 98-R-316.
5. M. Clavel, S. Eker, P. Lincoln, and J. Meseguer. Principles of Maude. In J. Meseguer, editor, *Proceedings of the 1st International Workshop on Rewriting Logic and its Applications*, volume 5 of *Electronic Notes in Theoretical Computer Science*, Asilomar, Pacific Grove, CA, USA, September 1996. North Holland.
6. Nachum Dershowitz and Jean-Pierre Jouannaud. *Handbook of Theoretical Computer Science*, volume B, chapter 6: Rewrite Systems, pages 244–320. Elsevier Science Publishers B. V. (North-Holland), 1990. Also as: Research report 478, LRI.
7. S. Eker. Term rewriting with operator evaluation strategies. In C Kirchner and H. Kirchner, editors, *Proceedings of the 2nd International Workshop on Rewriting Logic and its Applications*, Pont-à-Mousson, France, September 1998.
8. O. Fissore. Terminaison par induction. Mémoire de DEA, Université Henri Poincaré – Nancy 1, June 2000.
9. O. Fissore, I. Gnaedig, and H. Kirchner. Termination of rewriting with local strategies. Technical report, LORIA, Nancy, France, 2001. http://www.loria.fr/~gnaedig/INDO/STRAT-LO/strat-lo.ps.gz.
10. K. Futatsugi and A. Nakagawa. An overview of CAFE specification environment – an algebraic approach for creating, verifying, and maintaining formal specifications over networks. In *Proceedings of the 1st IEEE Int. Conference on Formal Engineering Methods*, 1997.
11. J. Giesl and Middeldorp A. Transforming Context-Sensitive Rewrite Systems. In *Proceedings of the 10th International Conference on Rewriting Techniques and Applications*, volume 1631 of *Lecture Notes in Computer Science*, pages 271–285, Trento, Italy, 1999. Springer-Verlag.
12. I. Gnaedig, H. Kirchner, and O. Fissore. Induction for Termination. Technical Report A00-R-357, LORIA, Nancy (France), 2000.
13. I. Gnaedig, H. Kirchner, and O. Fissore. Induction for innermost and outermost ground termination. Internal report, submitted, 2001. http://www.loria.fr/~fissore/Papers/in-out.ps.gz.
14. J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.P. Jouannaud. Introducing OBJ3. Technical report, Computer Science Laboratory, SRI International, march 1992.
15. P. Klint. A meta-environment for generating programming environments. *ACM Transactions on Software Engineering and Methodology*, 2:176–201, 1993.
16. S. Lucas. Termination of context-sensitive rewriting by rewriting. In *Proceedings of the 23rd International Colloquium on Automata, Languages and Programming*, volume 1099 of *Lecture Notes in Computer Science*, pages 122–133. Springer-Verlag, 1996.
17. S. Lucas. Context-sensitive rewriting strategies. Technical Report DSIC-II/7/00, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Spain, 2000.
18. Nakamura M. and Ogata K. The evaluation strategy for head normal form with and without on-demand flags. In K. Futatsugi, editor, *Proceedings of the Third International Workshop on Rewriting Logic and its Applications, WRLA'2000*, pages 211–227, Kanazawa City Cultural Halt, Kanazawa, Japan, September 2000. Electronic Notes in Theoretical Computer Science.
19. E. Visser. Stratego: A Language for Program Transformation based on Rewriting Strategies. System Description for Stratego 0.5. In Aart Middeldorp, editor, *Proceedings of the 12th International Conference on Rewriting Techniques and Applications*. Springer-Verlag, 2001. To appear.
20. H. Zantema. Termination of context-sensitive rewriting. In *Proceedings of the 8th International Conference on Rewriting Techniques and Applications*, volume 1232 of *Lecture Notes in Computer Science*, pages 172–186. Springer-Verlag, 1997.