Termination of rewriting strategies: a generic approach

Isabelle Gnaedig, Hélène Kirchner

Termination of rewriting strategies: a generic approach

Isabelle Gnaedig, Hélène Kirchner

LORIA-INRIA & LORIA-CNRS
P 239 F-54506 Vandœuvre-lès-Nancy Cedex
Fax: + 33 3 83 27 83 19
e-mail: gnaedig@loria.fr,Helene.Kirchner@loria.fr

Abstract. We propose a generic termination proof method for rewriting under strategies, based on an explicit induction on the termination property. Rewriting trees on ground terms are modelized by proof trees, generated by alternatively applying narrowing and abstracting steps. The induction principle is applied through the abstraction mechanism, where terms are replaced by variables representing any of their normal forms. The induction ordering is not given a priori, but defined with ordering constraints, incrementally set during the proof. The generic method is then instantiated for the innermost strategy, as an example.

1 Introducing the problem

Rewriting techniques are now widely used in automated deduction, especially to handle equality, as well as in programming, in functional, logical or rule-based languages. Termination of rewriting is a crucial property, important in itself to guarantee a result in a finite number of steps, but it is also required to decide properties like confluence and sufficient completeness, or to allow proofs by consistency. Existing methods for proving termination of term rewriting systems (TRS in short) essentially tackle the termination problem on the free term algebra, for rewriting without strategies. Most are based on syntactic or semantic noetherian orderings containing the rewriting relation induced by the TRS; other ones consist in transforming the termination problem of a TRS $\mathcal{R}$ into the decreasingness problem of another TRS or of pairs of terms, then handled with techniques of the previous category [5, 27].

In the context of proof environments for rule-based programming languages, such as ASF+SDF [20], Maude [3], CafeOBJ [12], ELAN [2], or TOM [25], where a program is a term rewriting system and the evaluation of a query consists in rewriting a ground expression, more specific termination proof tools are required, to allow termination proofs under specific reduction strategies. There are still few results in this domain. To our knowledge, methods have only been given for the innermost strategy [1, 13] and for the context-sensitive rewriting [24], which involves particular kinds of local strategies [23]. In previous works, we have also obtained termination results on ground term algebras for the innermost strategy [15, 8], for general local strategies on the operators [7], and for the outermost strategy [9].

In this paper, we propose a generic proof principle, based on an explicit induction mechanism on the termination property, which is a generalization of our three previous results. We then show how it can be instantiated to give an effective termination proof algorithm for the innermost strategy. The other instances - outermost and local strategies - are presented in the full version of the paper [14]. This generalizing work allowed not only to propose a generic version of our proof method, but also to considerably simplify the technical features of the algorithms initially designed for the different strategies.

The three above strategies have been chosen for their relevance to programming languages. The most widely used innermost strategy consists in rewriting always at the lowest possible positions. It is often used as a built-in mechanism in evaluation of rule-based or functional languages. In addition, for non-overlapping or locally confluent overlay systems [18], or systems satisfying critical peak conditions [19], innermost termination is equivalent to standard termination (i.e. termination for standard rewriting, which consists in rewriting without any strategy). As proved in [21], termination of rewriting is equivalent for the leftmost innermost and the innermost strategies.

The outermost strategy for evaluating expressions in the context of programming is essentially used when one knows that computations can be non-terminating. Outermost computations are of interest in particular for functional languages, where interpreters or compilers generally involve a

strategy for call by name. Often, lazy evaluation is used instead, but lazy evaluation may diverge while the outermost computation terminates. Lazy termination of functional languages has already been studied (see for example [26]), but to our knowledge, except our previously cited work, no termination proof method exists for specifically proving outermost termination of rewriting.

Local strategies on operators are used in particular to force the evaluation of expressions to terminate. A famous example is the evaluation of a recursive function defined with an `if_then_else_` expression, for which evaluating the first argument in priority may allow to avoid divergence. This kind of strategy is provided by languages such that OBJ3, CafeOBJ or Maude, and described in [16].

Local strategies are to be compared with context-sensitive rewriting, where rewriting is also allowed at some specified positions only in the terms: as local strategies specify in addition an ordering on these rewriting positions, they are more specific.

The termination proofs we propose rely on a generic principle and a few common concepts, that are emphasized in this paper. Our approach is based on an explicit induction mechanism on the termination property. The main idea is to proceed by induction on the ground term algebra with a noetherian ordering $\succ$, assuming that for any t' such that $t \succ t'$, t' terminates, i.e. there is no infinite derivation chain starting from t'.

Unlike classical induction proofs, where the ordering is given, we do not need to define it *a priori*. We only have to check its existence by ensuring satisfiability of ordering constraints incrementally set along the termination proof. Thanks to the power of induction, the generated constraints are often simpler to solve than for other approaches, and even, in many cases, do not need any constraint solving algorithm.

Directly using the termination notion on terms has also been proposed in [17], but for inductively proving well-foundedness of binary relations, among which path orderings.

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 f having arity $n \in \mathbb{N}$ (denoted $f:n$), 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 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) 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, $Var(t)$ is the set of variables of t, $\mathcal{O}(t)$ is the set of positions in t, inductively defined as follows: $\mathcal{O}(t) = \{\epsilon\}\ if\ t \in \mathcal{X},\ \mathcal{O}(t) = \{\epsilon\} \cup \{i.p \mid 1 \leq i \leq n \text{ and } p \in \mathcal{O}(t_i)\}\ if\ t = f(t_1,\ldots,t_n)$. 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 $\mathcal{T}(\mathcal{F},\mathcal{X})$. 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 $Ran(\sigma) = \bigcup_{x \in Dom(\sigma)} Var(\sigma x)$. We have in addition $Dom(\sigma) \cap Ran(\sigma) = \emptyset$. A ground substitution or instantiation is an assignment from $\mathcal{X}$ to $\mathcal{T}(\mathcal{F})$. The composition of substitutions σ_1 followed by σ_2 is denoted $\sigma_2\sigma_1$.

Given a set $\mathcal{R}$ of rewrite rules (a set of pairs of terms of $\mathcal{T}(\mathcal{F},\mathcal{X})$, denoted $l \to r$, such that $Var(r) \subseteq Var(l)$) or term rewriting system on $\mathcal{T}(\mathcal{F},\mathcal{X})$, 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 of $\mathcal{F}$ for $\mathcal{R}$ is denoted by $\mathcal{D}ef_R$ ($\mathcal{R}$ is omitted when there is no ambiguity).

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 exists 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 $t = s[\sigma r]_p$. The term $s|_p$ is called a redex. The reflexive transitive closure of the rewriting relation induced by $\mathcal{R}$ is denoted by $\overset{*}{\to}_{\mathcal{R}}$.

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 $\mathcal{R}$ and the substitution σ, when σ is a most general unifier of $t|_p$ and l, and $t' = \sigma(t[r]_p)$. This is denoted $t \rightsquigarrow_R^{p,l\to r,\sigma} t'$ 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 $Var(l) \cap Var(t) = \emptyset$.

An ordering $\succ$ on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is said to be noetherian 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 $\mathcal{T}(\mathcal{F}, \mathcal{X})$, $f(\ldots t \ldots) \succ t$. Observe that, for $\mathcal{F}$ and $\mathcal{X}$ finite, if $\succ$ is $\mathcal{F}$-stable and has the subterm property, then it is noetherian [22]. 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. 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.

Rewriting strategies are in general aimed at reducing the derivation tree (for standard rewriting) of terms. The following definition expresses that rewriting a term with a strategy S can only give a term that would be obtained with the standard rewriting relation.

Definition 1 (rewriting strategy). *Let $\mathcal{R}$ a TRS on $\mathcal{T}(\mathcal{F}, \mathcal{X})$. A* rewriting strategy *S for $\mathcal{R}$ is a mapping $S : \mathcal{T}(\mathcal{F}, \mathcal{X}) \rightarrow \mathcal{T}(\mathcal{F}, \mathcal{X})$ such that for every $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $S(t) = t'$ (we write $t \rightarrow^S t'$) where t' is such that $t \rightarrow_{\mathcal{R}} t'$.*

Let t be a term of $\mathcal{T}(\mathcal{F})$; we say that t terminates (w.r.t. to the strategy S) if and only if every rewriting derivation (or derivation chain) (w.r.t. to the strategy S) starting from t is finite. Given a term t, we call normal form (w.r.t. to the strategy S) or S-normal form of t, and we note it $t{\downarrow}$, any irreducible term, if it exists, such that $t \stackrel{*}{\rightarrow}^S t{\downarrow}$.

3 The inductive proof process

For proving that a term t of $\mathcal{T}(\mathcal{F})$ terminates (for the considered strategy), we proceed by induction on $\mathcal{T}(\mathcal{F})$ with a noetherian ordering $\succ$, assuming that for any t' such that $t \succ t'$, t' terminates. To warrant non emptyness of $\mathcal{T}(\mathcal{F})$, we assume that $\mathcal{F}$ contains at least a constructor constant.

The main intuition is to observe the rewriting derivation tree (for the considered strategy) starting from a ground term $t \in \mathcal{T}(\mathcal{F})$ which is any instance of a term $g(x_1, \ldots, x_m)$, for some defined function symbol $g \in \mathcal{D}ef$, and variables $x_1, \ldots, x_m$. Proving termination on ground terms amounts proving that all rewriting derivation trees have only finite branches, using the same induction ordering $\succ$ for all trees.

Each rewriting derivation tree is simulated, using a lifting mechanism, by a proof tree, developed from $g(x_1, \ldots, x_m)$ on $\mathcal{T}(\mathcal{F}, \mathcal{X})$, for every $g \in \mathcal{D}ef$, by alternatively using two main operations, namely narrowing and abstraction, adapted to the considered rewriting strategy. More precisely, narrowing schematizes all rewriting possibilities of terms. The abstraction process simulates the normalization of subterms in the derivations, according to the strategy. It consists in replacing these subterms by special variables, denoting one of their normal forms, without computing them. This abstraction step is performed on subterms that can be assumed terminating by induction hypothesis.

The schematization of ground rewriting derivation trees is achieved through constraints. The nodes of the developed proof trees are composed of a current term of $\mathcal{T}(\mathcal{F}, \mathcal{X})$, and a set of ground substitutions represented by a constraint progressively built along the successive abstraction and narrowing steps. Each node in a proof tree schematizes a set of ground terms: the ground instances of the current term, that are solutions of the constraint.

The constraint is in fact composed of two kinds of formulas: ordering constraints, set to warrant the validity of the inductive steps, and abstraction constraints combined to narrowing substitutions, which effectively define the relevant sets of ground terms. The latter are actually useful for controlling the narrowing process, well known to easily diverge.

The termination proof procedures given in this paper are described by deduction rules applied with a special control $Strat{-}Rules(S)$, depending on the studied rewriting strategy S. To prove termination of $\mathcal{R}$ on any term $t \in \mathcal{T}(\mathcal{F})$ w.r.t. the strategy S, we consider a generic reference term $t_{ref} = g(x_1, \ldots, x_m)$ for each defined symbol $g \in \mathcal{D}ef$, and empty sets $\top$ of constraints. Applying the deduction rules according to the strategy $Strat{-}Rules(S)$ to the initial state $(\{g(x_1, \ldots, x_m)\}, \top, \top)$ builds a proof tree, whose nodes are the states produced by the inference rules. Branching is produced by the different possible narrowing steps.

Termination is established when the procedure terminates because the deduction rules do not apply anymore and all terminal states of all proof trees have an empty set of terms.

4 Ordering constraints and abstraction

The induction ordering is constrained along the proof by imposing constraints between terms that must be comparable, each time the induction hypothesis is used in the abstraction mechanism. As we are working with a lifting mechanism on the proof trees with terms of $\mathcal{T}(\mathcal{F},\mathcal{X})$, we directly work with an ordering $\succ_{\mathcal{P}}$ on $\mathcal{T}(\mathcal{F},\mathcal{X})$ such that $t \succ_{\mathcal{P}} u$ implies $\theta t \succ \theta u$, for every θ solution of the constraint associated to u.

So inequalities of the form $t > u_1, \ldots, u_m$ are accumulated, which are called *ordering constraints*. Any ordering $\succ_{\mathcal{P}}$ on $\mathcal{T}(\mathcal{F},\mathcal{X})$ satisfying them and which is stable under substitution fulfills the previous requirements on ground terms. The ordering $\succ_{\mathcal{P}}$, defined on $\mathcal{T}(\mathcal{F},\mathcal{X})$, can then be seen as an extension of the induction ordering $\succ$, defined on $\mathcal{T}(\mathcal{F})$. For convenience, the ordering $\succ_{\mathcal{P}}$ will also be written $\succ$.

It is important to remark that, for establishing the inductive termination proof, it is sufficient to decide whether there exists such an ordering.

Definition 2 (ordering constraint). An ordering constraint *is a pair of terms of* $\mathcal{T}(\mathcal{F},\mathcal{X})$ *noted* $(t > t')$. *It is said to be* satisfiable *if there exists 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 exists an ordering satisfying all conjuncts. The empty conjunction, always satisfied, is denoted by* $\top$.

Satisfiability of a constraint conjunction C of this form is undecidable. But a sufficient condition for an ordering $\succ$ to satisfy C is that $\succ$ is stable under substitution and $t \succ t'$ for any constraint $t > t'$ of C.

To abstract a term t at positions $i_1, \ldots, i_p$, where the $t|_j$ are supposed to have a normal form $t|_j\downarrow$, we replace the $t|_j$ by abstraction variables X_j representing respectively one of their possible normal forms. These variables are taken in a set $\mathcal{X}_A$ disjoint from $\mathcal{X}$. Substitutions and instantiations are extended to $\mathcal{T}(\mathcal{F},\mathcal{X}\cup\mathcal{X}_A)$ in the following way. Let $X \in \mathcal{X}_A$; for any substitution σ (resp. instantiation θ) such that $X \in Dom(\sigma)$, σX (resp. θX) is in S-normal form.

Definition 3 (term abstraction). *The term* $t[t|_j]_{j\in\{i_1,\ldots,i_p\}}$ *is said to be* abstracted *into the term* u *(called* abstraction *of* t*) at positions* $\{i_1, \ldots, i_p\}$ *iff* $u = t[X_j]_{j\in\{i_1,\ldots,i_p\}}$, *where the* $X_j, j \in \{i_1, \ldots, i_p\}$ *are fresh distinct abstraction variables.*

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

5 Narrowing and the involved constraints

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

The narrowing relation depends on the considered strategy S and the usual definition needs to be refined. The first idea is to use innermost (resp. outermost) narrowing. Then, if a position p in a term t is a narrowing position, a suffix (resp. prefix) position of p cannot be a narrowing position too. However, if we consider ground instances of t, we can have rewriting positions p for some instances, and p' for some other instances, such that p' is a suffix (resp. a prefix) of p. So, when narrowing at some position p, the set of relevant ground instances of t is defined by excluding the ground instances that would be narrowable at some suffix (resp. prefix) position of p, that we call S-better position: a position S-better than a position p in t is a suffix position of p if S is the

innermost strategy, a prefix position of p if S is the outermost strategy. Note that local strategies are not of the same nature, and there is no S-better position in this case.

Moreover, to preserve the fact that a narrowing step of t schematizes a rewriting step of possible ground instances of t, we have to be sure that an innermost (resp. outermost) narrowing redex in t corresponds to the same rewriting redex in a ground instance of t. This is the case only if, in the rewriting chain of the ground instance of t, there is no rewriting redex at a suffix position of variable of t anymore. So before each narrowing step, we schematize the longest rewriting chain of any ground instance of t, whose redexes occur in the variable part of the instantiation, by a linear variable renaming. Linearity is crucial to express that, in the previous rewriting chain, ground instances of the same variables can be reduced in different ways. For the innermost strategy, abstraction of variables performs this schematization. For the outermost strategy, a reduction renaming is introduced [14]. For local strategies however, this variable renaming is not relevant.

The S-narrowing steps applying to a given term t are computed in the following way. After applying the variable renaming to t, we look at every position p of t such that $t|_p$ unifies with the left-hand side of a rule using a substitution σ. The position p is a S-narrowing position of t, iff there is no S-better position p' of t such that $\sigma t|_{p'}$ unifies with a left-hand side of rule. Then we look for every S-better position p' than p in t such that $\sigma t|_{p'}$ narrows with some substitution σ' and some rule $l' \to r'$, and we set a constraint to exclude these substitutions. So the substitutions used to narrow a term have in general to satisfy a set of disequalities coming from the negation of previous substitutions. To formalize this point, we need the following notations and definitions.

In the following, we identify a substitution $\sigma = (x_1 \mapsto t_1) \dots (x_n \mapsto t_n)$ on $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{X}_A)$ with the finite set of solved equations $(x_1 = t_1) \wedge \dots \wedge (x_n = t_n)$, also denoted by the equality formula $\bigwedge_i (x_i = t_i)$, with $x_i \in \mathcal{X} \cup \mathcal{X}_A$, $t_i \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{X}_A)$, where $=$ is the syntactic equality. Similarly, we call *negation* $\overline{\sigma}$ of the substitution σ the formula $\bigvee_i (x_i \neq t_i)$. 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.

Definition 4 (S-narrowing). *A term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{X}_A)$* S-narrows *into a term $t' \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{X}_A)$ at the non-variable position p of t, using the rule $l \to r \in \mathcal{R}$ with the constrained substitution $\sigma = \sigma_0 \wedge \bigwedge_{j \in [1..k]} \overline{\sigma_j}$, which is written $t \rightsquigarrow^S_{p, l \to r, \sigma} t'$ iff $\sigma_0(l) = \sigma_0(t|_p)$ and $t' = \sigma_0(t[r]_p)$ where σ_0 is the most general unifier of $t|_p$ and l and $\sigma_j, j \in [1..k]$ are all most general unifiers of $\sigma_0 t|_{p'}$ and a left-hand side l' of a rule of $\mathcal{R}$, for all position p' which are S-better positions than p in t.*

Abstraction constraints have to be combined with the narrowing constrained substitutions to characterize the ground terms schematized by the proof trees. Hence the introduction of abstraction constraint formulas.

Definition 5 (abstraction constraint formula). *An* abstraction constraint formula *(ACF in short) is a formula $\bigwedge_i (t_i\downarrow = t'_i) \wedge \bigwedge_j (x_j = t_j) \wedge \bigwedge_k \bigvee_{l_k} (u_{l_k} \neq v_{l_k})$, where $t_i, t'_i, t_j, u_{l_k}, v_{l_k} \in \mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{X}_A)$, $x_j \in \mathcal{X} \cup \mathcal{X}_A$.*

Definition 6 (satisfiability of an ACF). *An* abstraction constraint formula $\bigwedge_i (t_i\downarrow = t'_i) \wedge \bigwedge_j (x_j = t_j) \wedge \bigwedge_k \bigvee_{l_k} (u_{l_k} \neq v_{l_k})$, *is* satisfiable *iff there exists at least one instantiation θ such that $\bigwedge_i (\theta t_i\downarrow = \theta t'_i) \wedge \bigwedge_j (\theta x_j = \theta t_j) \wedge \bigwedge_k \bigvee_{l_k} (\theta u_{l_k} \neq \theta v_{l_k})$. The instantiation θ is then said to satisfy the ACF A and is called solution of A.*

Integrating a constrained substitution $\sigma = \sigma_0 \wedge \bigwedge_i \bigvee_{j_i} (x_{j_i} \neq t_{j_i})$ to an ACF A is done by adding the formula defining σ to A, thus giving the formula $A \wedge \sigma$. For a better readability on examples, we can propagate σ into A (by applying σ_0 to A), thus getting instantiated abstraction constraints of the form $t_i\downarrow = t'_i$ from initial abstraction constraints of the form $t_i\downarrow = X_i$.

An ACF A is attached to each term u in the proof trees; its solutions characterize the interesting ground instances of this term, i.e. the θu such that θ is a solution of A. When A has no solution, the current node of the proof tree represents no ground term. Such nodes are then irrelevant for the termination proof. Detecting and suppressing them during a narrowing step allows to contain the narrowing mechanism.

So we have the choice between generating only the relevant nodes of the proof tree, by testing satisfiability of A at each step, or stopping the proof on a branch on an irrelevant node, by testing unsatisfiability of A. These are both facets of the same question, but in practice, they are handled in different ways.

Checking satisfiability of A is in general undecidable. The disequality part of an ACF is a particular instance of a disunification problem (a quantifier free equational formula), whose satisfiability has been addressed in [4], that provides rules to transform any disunification problem into a solved form. Testing satisfiability of the equational part of an ACF is undecidable in general, but sufficient conditions can be given, relying on a characterization of normal forms.

Unsatisfiability of A is also undecidable in general, but simple sufficient conditions can be used, very often applicable in practice. They rely on reducibility, unifiability, narrowing and constructor tests, and can be found in [14].

So both satisfiability and unsatisfiability checks need to use sufficient conditions. But in the first case, the proof process stops with failure as soon as satisfiability of A cannot be proved. In the second one, it can go on, until A is proved to be unsatisfiable, or until other stopping conditions are fulfilled.

It is important to point out the flexibility of the proof method that allows the combination with auxiliary termination proofs using different techniques: when the induction hypothesis cannot be applied on a term u, i.e. when it is not possible to decide whether the ordering constraints are satisfiable, it is often possible to prove termination (for the considered strategy) of any ground instance of u by another way. In the following we use a predicate $TERMIN(S, u)$ that is true iff every ground instance of u terminates for the considered strategy S.

In particular, $TERMIN(S, u)$ is true when every instance of u is in normal form. This is the case when u is not narrowable, and all variables of u are in $\mathcal{X}_A$. This includes the cases where u itself is an abstraction variable, and where u is a non narrowable ground term.

Every instance of a narrowable u whose variables are all in $\mathcal{X}_A$, and whose narrowing substitutions are not compatible with A, is also in normal form. As said previously, these narrowing possibilities do not represent any reduction step for the ground instances of u, which are then in normal form.

Otherwise, in many cases, for proving that $TERMIN(S, u)$ is true, the notion of usable rules [1] is relevant. Complete results on usable rules for the three considered strategies are respectively given in [14, 9, 7].

6 The termination proof procedure

We are now ready to describe the different steps of the proof mechanism presented in Section 3.

The proof steps generate the proof trees in transforming 3-tuples (T, A, C) where T is a set of terms of $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{X}_A)$ containing the current term u whose termination has to be proved, A is a conjunction of abstraction constraints, C is a conjunction of ordering constraints stated by the abstraction steps.

Starting from initial states $(T = \{t_{ref} = g(x_1, \ldots, x_m)\}, A = \top, C = \top)$, where $g \in \mathcal{D}ef$, the proof process consists in iterating the following generic steps:

- The first step abstracts the current term t at given positions $i_1, \ldots, i_p$. If the conjunction of ordering constraints $\bigwedge_j t_{ref} > t|_j$ is satisfiable for some $j \in \{i_1, \ldots, i_p\}$, we suppose, by induction, the existence of irreducible forms for the $t|_j$. We must have $TERMIN(S, t|_j)$ for the other $t|_j$. Then, $t|_{i_1}, \ldots, t|_{i_p}$ are abstracted into abstraction variables $X_{i_1}, \ldots, X_{i_p}$. The abstraction constraints $t|_{i_1}\downarrow = X_{i_1}, \ldots, t|_{i_p}\downarrow = X_{i_p}$ are added to the ACF A. We call that step the *abstract* step.
- The second step narrows the resulting term u in one step with all possible rewrite rules of the rewrite system $\mathcal{R}$, and all possible substitutions σ, into terms v, according to Definition 4. This step is a branching step, creating as many states as narrowing possibilities. The substitution σ is added to A. This is the *narrow* step.
- We then have a *stop* step halting the proof process on the current branch of the proof tree, when A is detected to be unsatisfiable, or when the ground instances of the current term can be stated terminating for the considered strategy. This happens when the whole current term u can be abstracted, i.e. when the induction hypothesis applies on it, or when we have $TERMIN(S, u)$.

The satisfiability and unsatisfiability tests of A are integrated in the previously presented steps. If testing unsatisfiability of A is chosen, the unsatisfiability test is integrated in the *stop* step. If

testing the satisfiability of A is chosen, the test is made at each attempt of an abstraction or a narrowing step, which are then effectively performed only if A can be proved satisfiable. Otherwise, the proof cannot go on anymore and stops with failure.

As we will see later, for a given rewriting strategy S, these generic proof steps are instantiated by more precise mechanisms, depending on S, and taking advantage of its specificity. We will define these specific instances by inference rules.

Note that there are different ways to simulate the rewriting relation on ground terms, using abstraction and narrowing. For example, the abstraction positions can be chosen so that the abstraction mechanism captures the greatest possible number of rewriting steps. For that, we abstract the greatest possible subterms in the term. We can also choose in priority the smallest possible subterms u_i, that are constants or variables. The ordering constraints $t > u_i$ needed to apply the induction hypothesis, and then to abstract the term, are easier to satisfy than in the previous case since the u_i are smaller.

The previous proof steps, applied to every reference term $t_{ref} = g(x_1, \ldots, x_m)$, where $x_1, \ldots, x_m \in \mathcal{X}$ and $g \in \mathcal{D}ef$, can be combined in the same way whatever $S \in \{Innermost, Outermost, Local{-}Strat\}$:

$$Strat{-}Rules(S) = repeat^*(try(abstract),\ try(narrow),\ try(stop)).$$

"$repeat^*(T_1, \ldots, T_n)$" repeats the strategies of the set $\{T_1, \ldots, T_n\}$ until it is not possible anymore. The operator "try" is a generic operator that can be instantiated, following S, by $try - skip(T)$, expressing that the strategy or rule T is tried, and skipped when it cannot be applied, or by "$try - stop(T)$", stopping the strategy if T cannot be applied.

For each strategy $S \in \{Innermost, Outermost, Local{-}Strat\}$, we write $SUCCESS\ (g, \succ)$ if the application of $Strat{-}Rules(S)$ on $(\{g(x_1, \ldots, x_m)\}, \top, \top)$ gives a finite proof tree, whose sets C of ordering constraints are satisfied by a same ordering $\succ$, and whose leaves are either states of the form $(\emptyset, A, C)$ or states whose set of constraints A is unsatisfiable.

Theorem 1. *Let R be a TRS on a set $\mathcal{F}$ of symbols containing at least a constructor constant. If there exists an $\mathcal{F}$-stable ordering $\succ$ having the subterm property, such that for each symbol $g \in \mathcal{D}ef$, we have $SUCCESS(g, \succ)$, then every term of $\mathcal{T}(\mathcal{F})$ terminates with respect to the strategy S.*

The proof is given in [14].

7 Instantiating the generic process: the innermost case

As an example, here, we instantiate our generic proof process with the innermost strategy, which is the rewriting strategy such that for a given TRS $\mathcal{R}$ and any term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, if $t \rightarrow^S t'$, the rewriting position p in t is such that there is no suffix position p' of p such that t rewrites at position p'.

When rewriting a ground instance of the current term according to the innermost principle, the ground instances of variables in the current term have to be normalized before a redex appears higher in the term. So the variable renaming performed before narrowing corresponds here to abstracting variables in the current term. Then, any innermost proof tree from a pattern $g(x_1, \ldots, x_m)$ begins in abstracting $x_1, \ldots, x_m$. All the following terms in the tree are then in $\mathcal{T}(\mathcal{F}, \mathcal{X}_A)$ [14], which makes superfluous the further renamings.

The inference rules **Abstract**, **Narrow** and **Stop** instantiate respectively the proof steps *abstract*, *narrow*, and *stop* defined in Section 6. They are given in Table 1. Their application conditions depend on whether satisfiability of A or unsatisfiability of A is checked. We present here the conditions where unsatisfiability is checked.

Once instantiated, the generic strategy $Strat{-}Rules(S)$ simply becomes:

repeat*(try-skip(**Abstract**); try-skip(**Narrow**); try-skip(**Stop**)).

The procedure can diverge, with infinite alternate applications of **Abstract** and **Narrow**. In this case, nothing can be said on termination. Termination is proved when, for all proof trees, the procedure stops with an application of **Stop** on each branch, generating only final states of the form $(\emptyset, A, C)$.

Table 1. Inference rules for the innermost strategy

Abstract: $\dfrac{\{t\},\ \ A,\ \ C}{\{u\},\ A \wedge t|_{i_1}\downarrow = X_{i_1} \ldots \wedge\ t|_{i_p}\downarrow = X_{i_p},\ C \wedge H_C(t|_{i_1}) \ldots \wedge H_C(t|_{i_p})}$

where t is abstracted into u at positions $i_1, \ldots, i_p \neq \epsilon$

if COND−ABSTRACT

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

if $t \rightsquigarrow_R^{Inn,\sigma} u$ and COND−NARROW

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

if $COND{-}STOP$

and $H_A(t) = \begin{cases} \top & \text{if any ground instance of } t \\ & \text{is in normal form} \\ t\downarrow = X & \text{otherwise.} \end{cases}$

$H_C(t) = \begin{cases} \top & \text{if } TERMIN(Innermost, t) \\ t_{ref} > t & \text{otherwise.} \end{cases}$

Table 2. Conditions for inference rules dealing with unsatisfiability of A

COND−ABSTRACT : $C \wedge H_C(t|_{i_1}) \ldots \wedge H_C(|t_{i_p})$ is satisfiable
COND−NARROW : $true$
COND−STOP : $(C \wedge H_C(t))$ is satisfiable or A is unsatisfiable.

Example 1. Let us now give an example illustrating how the usable rules can be helpful and why detecting unsatisfiability of A can be important. Let us consider the following system $\mathcal{R}$, which is not terminating but terminates with the innermost strategy:

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

We prove innermost termination of $\mathcal{R}$ on $\mathcal{T}(\mathcal{F})$, where $\mathcal{F} = \{0:0, s:1, plus:2, f:3, g:2\}$. The defined symbols of $\mathcal{F}$ are $f, plus$ and g. Since we are interested in the narrowing substitution applied to the terms of the proof trees, but not in its definition on the variables of the rules, the narrowing substitutions are restricted here to the variables of the narrowed terms.

Let us apply the inference rules checking unsatisfiability of A, whose conditions are given in Table 2. Applying the rules on $f(x_1, x_2, x_3)$, we get the proof tree given in Table 3.

The first **Abstract** applies since $f(x_1, x_2, x_3) > x_1, x_2, x_3$ is satisfiable by any simplification ordering. The second **Abstract** applies by using the $TERMIN$ predicate. Indeed, the usable rules of $plus(X_3, X_3)$ consist of the system $\{plus(x,0) \to x, plus(x,s(y)) \to s(plus(x,y))\}$, that can be proved terminating with any precedence based ordering (independent of the induction ordering) with the precedence $plus \succ_{\mathcal{F}} s$, which ensures the property $TERMIN(Innermost, plus(X_3, X_3))$. Without abstraction here, the process would have generated a branch containing an infinite number of **Narrow** applications.

Finally, **Stop** applies because the constraint A becomes unsatisfiable. Indeed, it contains the abstraction constraint $plus(0,0)\downarrow = s(0)$, which is not true since the unique normal form of $plus(0,0)$ is 0.

Considering now $plus(x_1, x_2)$, we get the proof tree given in Table 4.

Abstract applies since $g(x_1, x_2) > x_1, x_2$ is satisfiable by any simplification ordering. **Stop** applies on the left branch because X_1 is an abstraction variable, hence we trivially have $TERMIN$

Table 3. Proof tree for the symbol f

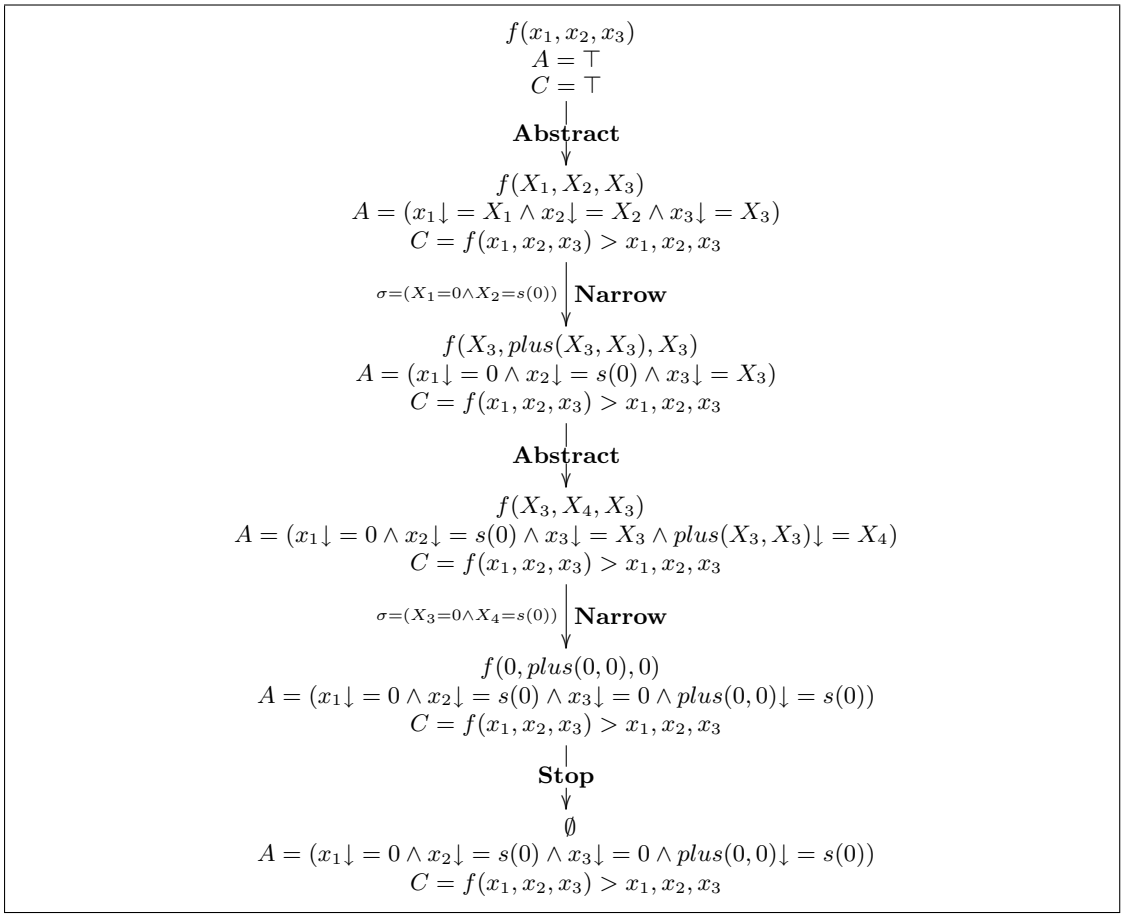
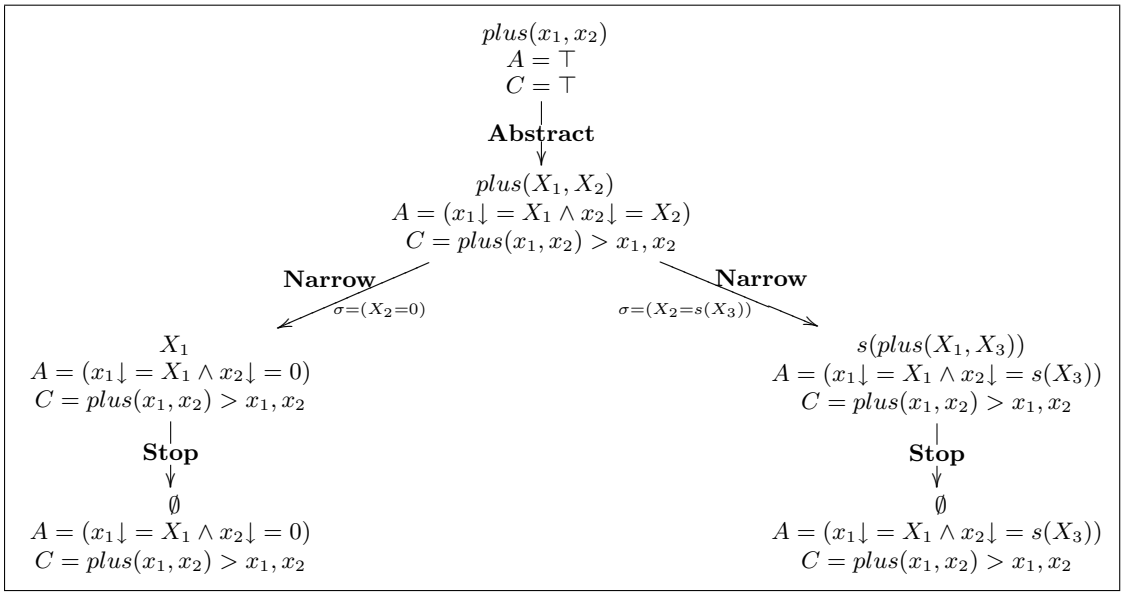


Table 4. Proof tree for the symbol $plus$



$(Innermost, X_1)$. **Stop** applies on the right branch by using the $TERMIN$ predicate. Indeed, the usable rules of $s(plus(X_1, X_3))$ consist of the previous terminating system $\{plus\ (x, 0) \rightarrow x, plus(x, s(y)) \rightarrow s(plus(x, y))\}$.

Applying the inference rules on $g(x_1, x_2)$, we get a similar termination proof tree.

8 Conclusion

The generic termination proof method presented in this paper is based on the simple ideas of schematizing and observing the derivation trees of ground terms and of using an induction ordering to stop derivations as soon as termination is ensured by induction. The method makes clear the schematization power of narrowing, abstraction and constraints. Constraints are heavily used on one hand to gather conditions that the induction ordering must satisfy, on the other hand to represent the set of ground instances of generic terms.

Our technique is implemented in a system named CARIBOO [8, 10], providing a termination proof tool for the innermost, the outermost, and the local strategies [1].

To deal with the generated constraints, the proof process of CARIBOO can use integrated features, like the computation of usable rules, the use of the subterm ordering or the Lexicographic Path ordering to satisfy ordering constraints, and the test of sufficient conditions for detecting unsatisfiability of A.

It can also delegate features, as solving the ordering constraints or orienting the usable rules when the LPO fails, proving termination of a term, or testing satisfiability of A. Delegation is either proposed to the user, or automatically ensured by the ordering constraint solver Cime2.

CARIBOO provides several automation modes for dealing with constraints. Dealing with unsatisfiability of A allows a complete automatic mode, providing a termination proof for a large class of examples (a library is available with the distribution of CARIBOO). It is interesting to note that thanks to the power of induction, and to the help of usable rules, the generated ordering constraints are often simple, and are easily satisfied by the subterm ordering or an LPO.

Finally, the techniques presented here have also been applied to weak termination in [11].

As our proof process is very closed to the rewriting mechanism, it could easily be extended to conditional, equational and typed rewriting, by simply adapting the narrowing definition. Our approach is also promising to tackle inductive proofs of other term properties like confluence or ground reducibility.

References

1. T. Arts and J. Giesl. Proving innermost normalization automatically. Technical Report 96/39, Technische Hochschule Darmstadt, Germany, 1996.
2. P. Borovanský, C. Kirchner, H. Kirchner, P.-E. Moreau, and Ch. Ringeissen. An Overview of ELAN. In C. Kirchner and H. Kirchner, editors, *Proceedings of the 2nd International Workshop on Rewriting Logic and its Applications*, Electronic Notes in Theoretical Computer Science, Pont-à-Mousson (France), September 1998. Elsevier Science Publishers B. V. (North-Holland).
3. 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.
4. H. Comon. Disunification: a survey. In Jean-Louis Lassez and G. Plotkin, editors, *Computational Logic. Essays in honor of Alan Robinson*, chapter 9, pages 322–359. The MIT press, Cambridge (MA, USA), 1991.
5. N. Dershowitz and D.A. Plaisted. Rewriting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 9, pages 535–610. Elsevier Science, 2001.
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. O. Fissore, I. Gnaedig, and H. Kirchner. Termination of rewriting with local strategies. In M. P. Bonacina and B. Gramlich, editors, *Selected papers of the 4th International Workshop on Strategies in Automated Deduction*, volume 58 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers B. V. (North-Holland), 2001.

[1] Available at http://protheo.loria.fr/softwares/cariboo/

8. O. Fissore, I. Gnaedig, and H. Kirchner. CARIBOO : An induction based proof tool for termination with strategies. In *Proceedings of the 4th International Conference on Principles and Practice of Declarative Programming*, pages 62–73, Pittsburgh (USA), October 2002. ACM Press.
9. O. Fissore, I. Gnaedig, and H. Kirchner. Outermost ground termination. In *Proceedings of the 4th International Workshop on Rewriting Logic and Its Applications*, volume 71 of *Electronic Notes in Theoretical Computer Science*, Pisa, Italy, September 2002. Elsevier Science Publishers B. V. (North-Holland).
10. O. Fissore, I. Gnaedig, and H Kirchner. Cariboo, a termination proof tool for rewriting-based programming languages with strategies. Free GPL Licence, APP registration IDDN.FR.001.170013.000.R.P.2005.000.10600, August 2004. `Available at http://protheo.loria.fr/softwares/cariboo/`.
11. O. Fissore, I. Gnaedig, and H Kirchner. A proof of weak termination providing the right way to terminate. In *1st International Colloquium on THEORETICAL ASPECTS OF COMPUTING*, volume 3407 of *Lecture Notes in Computer Science*, pages 356–371, Guiyang, China, September 2004. Springer-Verlag.
12. 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.
13. J. Giesl and A Middeldorp. Innermost termination of context-sensitive rewriting. In *Proceedings of the 6th International Conference on Developments in Language Theory (DLT 2002)*, volume 2450 of *Lecture Notes in Computer Science*, pages 231–244, Kyoto, Japan, 2003. Springer-Verlag.
14. I. Gnaedig and H. Kirchner. Termination of rewriting strategies: a generic approach. Internal Report, Loria, Nancy, France, July 2005. `Available at http://www.loria.fr/~gnaedig/PAPERS/REPORTS/term-generic-internal.pdf`.
15. I. Gnaedig, H. Kirchner, and O. Fissore. Induction for innermost and outermost ground termination. Technical Report A01-R-178, LORIA, Nancy (France), September 2001.
16. J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.P. Jouannaud. Introducing OBJ3. Technical report, Computer Science Laboratory, SRI International, march 1992.
17. Goubault-Larreck. Well-founded recursive relations. In *Proc. 15th Int. Workshop Computer Science Logic (CSL'2001)*, volume 2142 of *Lecture Notes in Computer Science*, Paris, 2001. Springer-Verlag.
18. B. Gramlich. Abstract relations between restricted termination and confluence properties of rewrite systems. *Fundamenta Informaticae*, 24:3–23, 1995.
19. Bernhard Gramlich. On proving termination by innermost termination. In H. Ganzinger, editor, *Proceedings 7th Conference on Rewriting Techniques and Applications, New Brunswick (New Jersey, USA)*, volume 1103 of *Lecture Notes in Computer Science*, pages 93–107. Springer-Verlag, July 1996.
20. P. Klint. A meta-environment for generating programming environments. *ACM Transactions on Software Engineering and Methodology*, 2:176–201, 1993.
21. M.R.K. Krishna Rao. Some characteristics of strong normalization. *Theoretical Computer Science*, 239:141–164, 2000.
22. J. B. Kruskal. Well-quasi ordering, the tree theorem and Vazsonyi's conjecture. *Trans. Amer. Math. Soc.*, 95:210–225, 1960.
23. S. Lucas. Termination of rewriting with strategy annotations. In A. Voronkov and R. Nieuwenhuis, editors, *Proc. of 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR'01*, volume 2250 of *Lecture Notes in Artificial Intelligence*, pages 669–684, La Habana, Cuba, December 2001. Springer-Verlag, Berlin.
24. S. Lucas. Context-sensitive rewriting strategies. *Information and Computation*, 178(1):294–343, 2002.
25. Pierre-Etienne Moreau, Christophe Ringeissen, and Marian Vittek. A Pattern Matching Compiler for Multiple Target Languages. In G. Hedin, editor, *12th Conference on Compiler Construction, Warsaw (Poland)*, volume 2622 of *LNCS*, pages 61–76. Springer-Verlag, May 2003.
26. S. E. Panitz and M. Schmidt-Schauss. TEA: Automatically proving termination of programs in a non-strict higher-order functional language,. In *Proceedings of Static Analysis Symposium'97*, volume 1302 of *Lecture Notes in Computer Science*, pages 345–360. Springer-Verlag, 1997.
27. Terese. Termination. In M. Bezem, J.W. Klop, and R. de Vrijer, editors, *Term Rewriting Systems by "Terese"*, volume I of *Cambridge Tracts in Theoretical Computer Science*, chapter 6, pages 535–610. Cambridge University Press, 2003.