# Termination Proofs using *gpo* Ordering Constraints

Thomas Genet and Isabelle Gnaedig

INRIA Lorraine & CRIN CNRS - BP 101
54602 Villers-lès-Nancy CEDEX FRANCE
Phone: (+33) 3-83-59-30-18 - Fax: (+33) 3-83-27-83-19
E-mail: {Thomas.Genet, Isabelle.Gnaedig}@loria.fr

**Abstract.** We present here an algorithm for proving termination of term rewriting systems by *gpo* ordering constraint solving. The algorithm gives, as automatically as possible, an appropriate instance of the *gpo* generic ordering proving termination of a given system. Constraint solving is done efficiently thanks to a DAG shared term data structure.

## 1 Introduction

To prove termination of a Term Rewrite System (TRS for short), the most commonly used method is to define a well-founded ordering between terms and show that each rewrite step is a strictly decreasing step. In general, the proof is made by verification: orderings are proposed by the user and tested until an appropriate one is found.

Our goal here is to reduce human expertise by working in a constructive way: starting from constraints on a generic ordering, we help the user to build an appropriate specific instance of this ordering by using semi-automatic constraint solving methods.

The generic ordering, we start from, is the general path ordering (*gpo*) designed by Dershowitz and Hoot [2] for expressing in a single notion a large set of well known orderings: syntactic orderings such as *rpo* [10] or *lpo* [8], as well as semantic orderings like *spo* [8] or polynomial orderings [9]. It is based on a lexicographic combination of *termination functions*. Particular orderings, such as those cited above, are obtained by instantiating termination functions with particular values.

Our idea here is to combine the genericity of *gpo* with the constructive power of the constraint approach, to provide a method as automatic as possible for proving termination of TRS. Starting from inequalities on a general path ordering, we reduce the set of possible instantiations of termination functions by constraint solving, until a particular ordering is found when possible.

The problem tackled here is different from the constraint approach of ordering problems already proposed [1, 13, 14, 12, 7]. All are concerned with the satisfiability problem of ordering constraints (existence of a ground substitution validating the ordering constraints). In our approach however, we try to find a *gpo* ordering for validating inequalities between terms with variables, for any value of the variables.

## 2 The starting point: *gpo*

Let $F$ be a set of function symbols with arity, $X$ a set of variable symbols, $T(F,X)$ the set of terms defined on $F$ and $X$, and $T(F)$ the set of ground terms. For definitions of multiset, ordering, quasi-ordering, multiset extension, lexicographic extension, well-founded ordering, rewriting, see [3].

Let us recall the definition of *gpo* ordering from [2]. This definition is based on component orderings defined as follows. A component ordering on $T(F)$ is a pair $\langle\theta_i, \geq_i\rangle$ such that (i) $\theta_i$ is a homomorphism from $T(F)$ to an algebra $A$ and $\geq_i$ is a well-founded quasi-ordering on $A$, or (ii) $\theta_i$ is a function (called multiset extraction function in [2]) from terms to multisets of selected immediate subterms, that is $\theta_i(f(s_1,\ldots,s_n)) = \{s_{j_1},\ldots,s_{j_m}\}$, such that $j_1,\ldots,j_m \in \{1,\ldots,n\}$ and $\geq_i$ is the multiset extension of *gpo* itself.

For any quasi-ordering $\geq_i$, we have: $\simeq_i \ = \ \geq_i \cap \leq_i$ and $>_i \ = \ \geq_i \cap \not\leq_i$. The $\theta_i$ are called *gpo termination functions*. For any term $s \in T(F)$, we denote by $\Theta_{i,j}(s)$ the tuple $\langle\theta_i(s),\ldots,\theta_j(s)\rangle$, where $0 \leq i < j$ and $\theta_i,\ldots,\theta_j$ are *gpo* termination functions. Let $>_{lex}$ be the associated ordering, i.e the lexicographic combination of orderings $>_i,\ldots,>_j$, and $\simeq_{lex}$ be the associated equivalence, i.e. the lexicographic combination of equivalences $\simeq_i,\ldots,\simeq_j$, such that $>_i,\ldots,>_j$ and $\simeq_i,\ldots,\simeq_j$ are respectively related to the homomorphisms $\theta_i,\ldots,\theta_j$. Let $\geq_{lex}$ be the relation $>_{lex} \cup \simeq_{lex}$. We denote $\Theta_{0,k}$ by $\Theta$.

**Definition 1.** (General Path Ordering)(Dershowitz & Hoot [2]). Let $\langle\theta_i, \geq_i\rangle$ be component orderings. The *general path ordering* $\geq_{gpo}$ on $T(F)$ is inductively defined by $\geq_{gpo} \ = \ >_{gpo} \cup \simeq_{gpo}$ where $s = f(s_1,\ldots,s_n) >_{gpo} g(t_1,\ldots,t_m) = t$ iff either (i) $s_i \geq_{gpo} t$ for some subterm $s_i$ of $s$, or (ii) $s >_{gpo} t_1,\ldots,s >_{gpo} t_m$ and $\Theta(s) >_{lex} \Theta(t)$. The equivalence $\simeq_{gpo}$ is defined by $s = f(s_1,\ldots,s_n) \simeq_{gpo} g(t_1,\ldots,t_m) = t$ iff $s >_{gpo} t_1,\ldots,s >_{gpo} t_m$, $t >_{gpo} s_1,\ldots,t >_{gpo} s_n$ and $\Theta(s) \simeq_{lex} \Theta(t)$.

**Theorem 2.** *(Dershowitz & Hoot [2]) Let $\geq_{gpo}$ be a gpo. A rewrite system $R$ terminates on $T(F)$ if (i) $l\sigma >_{gpo} r\sigma$ for all rules $l \to r$ of $R$, all ground substitution $\sigma$ and, (ii) $\forall s,t \in T(F)$, $s \to_R t$ and $s \geq_{gpo} t$ implies $f(\ldots,s,\ldots) \geq_{gpo} f(\ldots,t,\ldots)$ for any ground context $f(\ldots\ \ldots)$.*

Let $\Phi = (\mathcal{T}_{0,k}, \succsim_{lex})$ be a specific instance of $(\Theta, \geq_{lex})$, where $\mathcal{T}_{0,k}$ is the combination of *gpo* termination functions $\tau_0,\ldots,\tau_k$, and $\succsim_0,\ldots,\succsim_k$ are the related quasi-orderings; $\succ_{lex}$ is the lexicographic combination of $\succ_0,\ldots,\succ_k$, $\approx_{lex}$ is the lexicographic combination of $\approx_0,\ldots,\approx_k$, and $\succsim_{lex} \ = \succ_{lex} \cup \approx_{lex}$. When choosing a specific $\Phi = (\mathcal{T}_{0,k}, \succsim_{lex})$ for $(\Theta, \geq_{lex})$, we obtain *instances* of *gpo*, such as for example lexicographic path ordering (Kamin & Lévy [8]), multiset path ordering, polynomial path ordering (Lankford [9]). For more details, see [2]. An instance of the *gpo* based on a particular $\Phi = (\mathcal{T}_{0,k}, \succsim_{lex})$ will be denoted $\succ^{\Phi}_{gpo}$.

For operationally proving termination of TRSs with *gpo*, the usual approach consists in defining a specific *gpo* $\succ^{\Phi}_{gpo}$ on ground terms and using $\succ^{\Phi}_{gpo}$ as an ordering on terms with variables by proving that for all rules $l \to r$ of the TRS, we have $l \succ^{\Phi}_{gpo} r$ and $\forall s,t \in T(F,X)$, $\forall\sigma$ such that $s\sigma, t\sigma \in T(F)$, we have $s \succ_{gpo}^{\Phi} t \implies s\sigma \succ_{gpo}^{\Phi} t\sigma$. We choose here to explicitly define *gpo* on terms with variables. We extend the definition of $\Theta$ and $>_{lex}$ to $T(F,X)$ in the following way:

**Definition 3.** Let $s,t \in T(F,X)$, and let $\Theta$ and $>_{lex}$ be defined on $T(F)$.
(i) $\Theta(s) >_{lex} \Theta(t)$ iff $\forall\sigma$ s.t. $s\sigma, t\sigma \in T(F)$, we have $\Theta(s\sigma) >_{lex} \Theta(t\sigma)$,
(ii) $\Theta(s) \simeq_{lex} \Theta(t)$ iff $\forall\sigma$ s.t. $s\sigma, t\sigma \in T(F)$, we have $\Theta(s\sigma) \simeq_{lex} \Theta(t\sigma)$,
(iii) $\Theta(s) \geq_{lex} \Theta(t)$ iff $\Theta(s) >_{lex} \Theta(t)$ or $\Theta(s) \simeq_{lex} \Theta(t)$.

With this extension of the termination functions to $T(F,X)$, the definition of the general path ordering is extended to $T(F,X)$. From now on, we will use the definition of *gpo* on $T(F,X)$.

**Theorem 4.** *[5] The general path ordering $\geq_{gpo}$ is a quasi-ordering on $T(F,X)$ having the subterm property.*

**Proposition 5.** *[5] (Ground stability) Let $s,t \in T(F,X)$ and $\Phi = (\Theta, \geq_{lex})$. If $s \succ_{gpo}^{\Phi} t$ (resp. $s \approx_{gpo}^{\Phi} t$) then for any substitution $\sigma$ s.t. $s\sigma, t\sigma \in T(F)$, we have $s\sigma \succ_{gpo}^{\Phi} t\sigma$ (resp $s\sigma \approx_{gpo}^{\Phi} t\sigma$).*

Thanks to Proposition 5, for proving the first condition of Theorem 2 (termination theorem), it is enough to prove that: $l \succ_{gpo}^{\Phi} r$ for all rules $l \to r$ of $R$, and for a ground stable instance $\succ_{gpo}^{\Phi}$ of *gpo*. Restricting to ground stable instances of *gpo* is not critical since the instances of *gpo* used in practice are ground stable.

## 3 Solving *gpo* constraints using a term sharing data structure

In this paper, *gpo ordering constraints* are quantifier free first order formulas built on $s > t$ and $s \sim t$ where $s,t \in T(F,X)$. *Solving a gpo ordering constraint* $s > t$ (resp. $s \sim t$) is to decide whether there exists a particular instance $\Phi$ of *gpo* such that $s \succ_{gpo}^{\Phi} t$ (resp. $s \approx_{gpo}^{\Phi} t$), in constructing a specific $\Phi$. Solving *gpo* constraints is undecidable in general, but our goal is to build a procedure giving, when possible, an appropriate ordering in a semi-automatic way. First, our solving process automatically produces constraints on $(\Theta, \geq_{lex})$ from *gpo* constraints. Second, constraints on $(\Theta, \geq_{lex})$ are solved, by finding an appropriate instance for $\Theta$ and $\geq_{lex}$, in a semi-automatic way.

We choose to use a Directed Acyclic Graph (DAG) representation with term sharing for constraints (as in [11] in the context of completion). We thus avoid explosion of the size of the formulas, appearing during the resolution with a classical constraint representation. In this DAG structure, terms are graphs where nodes are labeled by symbols and edges represent the subterm relation. The DAG representation allows sharing of common subterms of distinct terms. On this DAG representation of terms, we additionally define edges representing ordering constraints labeled by logical formulas, called here *proof obligations.* Proof obligations ($\mathcal{O}$-proofs for short) are defined as follows (recall that $\Theta$ denote $\Theta_{0,k} = \langle\theta_0, \ldots, \theta_k\rangle$):

**Definition 6.** Let $\mathcal{X}_{\mathcal{P}}$ be a set of variables called the set of $\mathcal{O}$-proof variables. Let $\top$ be the trivial $\mathcal{O}$-proof, $s, t \in T(F, X)$, $P \in \mathcal{X}_{\mathcal{P}}$. The *set $\mathcal{P}$ of $\mathcal{O}$-proofs* is inductively defined by (i) $\top \in \mathcal{P}$, (ii) $P \in \mathcal{P}$, (iii) $\Theta(s) >_{lex} \Theta(t) \in \mathcal{P}$, and $\Theta(s) \simeq_{lex} \Theta(t) \in \mathcal{P}$, (iv) $A \wedge B \in \mathcal{P}$, if $A, B \in \mathcal{P}$, (v) $A \vee B \in \mathcal{P}$, if $A, B \in \mathcal{P}$.

We now define *satisfiability* of $\mathcal{O}$-proofs.

**Definition 7.** Let $\Phi$ be the pair $(\mathcal{T}_{0,k}, \succsim_{lex})$. Let $P, A, B \in \mathcal{P}$ and $s, t \in T(F, X)$. *$\Phi$ satisfies* $P$, denoted $\Phi \models P$ if (i) $P = \top$, or (ii) $P = A \vee B$ and ($\Phi \models A$ or $\Phi \models B$), or (iii) $P = A \wedge B$ and ($\Phi \models A$ and $\Phi \models B$), or (iv) $P = \Theta(s) >_{lex} \Theta(t)$ and $\mathcal{T}_{0,k}(s) \succ_{lex} \mathcal{T}_{0,k}(t)$, or (v) $P = \Theta(s) \simeq_{lex} \Theta(t)$ and $\mathcal{T}_{0,k}(s) \approx_{lex} \mathcal{T}_{0,k}(t)$.

Let us now define the DAG representation of rewrite rules. We call those graphs *Ordering Constraint Solving Graphs* (OCS graphs for short).

**Definition 8.** An *OCS graph* is a graph $G = (V, E)$ where $V$ is the set of vertices (or nodes) labeled by symbols of $F$ or variables of $X$, and $E \subseteq V \times V$ is the set of edges labeled by S, R, $>$ or $\sim$ for Subterm, Rewrite, inequality and equivalence edges respectively. The S, R, $>$ edges are directed. The $>$, $\sim$ edges are also labeled by an $\mathcal{O}$-proof. The subterm edges are also labeled by a natural $i$ called the *rank* of the subterm edge. For any node $\mathcal{F} \in V$, labeled by $f \in F$ of arity $n$, for all $i = 1 \ldots n$, there exists $\mathcal{G}_i \in V$ and a unique subterm edge $(\mathcal{F}, \mathcal{G}_i) \in E$ of rank $i$.

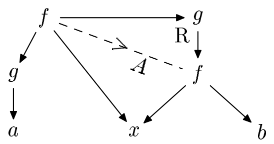
The subterm and the rewrite edges in OCS graphs represent the direct subterm relation in the term and the rewrite relation between terms in rules respectively. The edges labeled by $\mathcal{O}$-proofs represent the constraints on $(\Theta, >_{lex})$, obtained from *gpo* constraints in the first step of our solving process. Let us define the function *Term* mapping any node $\mathcal{F}$ of an OCS graph to a term $t$, such that $\mathcal{F}$ is the top node of the OCS graph representing $t$.

**Definition 9.** Let $G = (V, E)$ be an OCS graph and $\mathcal{F} \in V$. The function $Term$ from $V$ into $T(F, X)$ is inductively defined in the following manner: (i) if $\mathcal{F}$ is labeled by $x \in X$, then $Term(\mathcal{F}) = x$, (ii) if $\mathcal{F}$ is labeled by $f \in F$ of arity $n$, then $Term(\mathcal{F}) = f(Term(\mathcal{T}_1), \ldots, Term(\mathcal{T}_n))$ where for all $i = 1 \ldots n$, $\mathcal{T}_i \in V$, and $(\mathcal{F}, \mathcal{T}_i) \in E$ is a subterm edge of rank $i$.

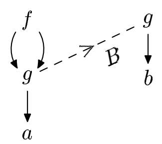
**Definition 10.** Let $l, r \in T(F, X)$. An *OCS representation of the rewrite rule* $l \to r$ is an OCS $G = (V, E)$ such that (i) there exist two nodes $\mathcal{F}, \mathcal{G} \in V$ and a unique rewrite edge $(\mathcal{F}, \mathcal{G}) \in E$ such that $Term(\mathcal{F}) = l$ and $Term(\mathcal{G}) = r$, and (ii) $\forall \mathcal{F}, \mathcal{F}' \in V$ s.t. $\mathcal{F} \neq \mathcal{F}'$, we have $Term(\mathcal{F}) \neq Term(\mathcal{F}')$.

In the previous definition, note that (ii) ensures sharing of subterms in the OCS representation of a rewrite rule. In the following, for any OCS graph $G = (V, E)$ with $\mathcal{F}, \mathcal{G} \in V$, $\mathcal{F} \mathrel{-\!\!>\!\!-} \mathcal{G}$ (resp. $\mathcal{F} \mathrel{-\!\!\sim\!\!-} \mathcal{G}$) denote an inequality edge (resp. equivalence edge) $(\mathcal{F}, \mathcal{G}) \in E$. We note $\mathcal{F} \mathrel{-\!\!\not>\!\!-} \mathcal{G}$ (resp. $\mathcal{F} \mathrel{-\!\!\not\sim\!\!-} \mathcal{G}$) if there is no inequality edge (resp. equivalence edge) $(\mathcal{F}, \mathcal{G}) \in E$. If $Term(\mathcal{F}) = s$ and $Term(\mathcal{G}) = t$, then $\mathcal{F} \mathrel{-\!\!>\!\!-} \mathcal{G}$ (resp. $\mathcal{F} \mathrel{-\!\!\sim\!\!-} \mathcal{G}$) is also denoted by $s \mathrel{-\!\!>\!\!-} t$ (resp. $s \mathrel{-\!\!\sim\!\!-} t$). Inequality and equivalence edges are called *ordering edges*. In the following figures, plain arrows denote subterm edges, plain arrows labeled by $R$ denote rewriting edges and dashed lines denote inequality and equivalence edges. Rank labels are omitted but can be deduced from the figures since subterm edges are always ordered by rank from left to right.

*Example 1.* The OCS representation of the rewrite rule $f(g(a), x) \rightarrow g(f(x, b))$ with an inequality edge labeled by an $\mathcal{O}$-proof label $A$ is presented in Graph 1.1. The OCS Graph 1.2 shows how the constraint $g(a) > g(b)$, duplicated in the previous example of decomposition of $f(g(a), g(a)) > g(b)$ can be represented by a unique edge labeled by an $\mathcal{O}$-proof label $B$.




Graph 1.1

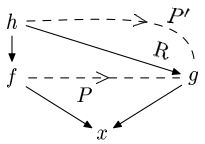



Graph 1.2

An OCS graph allows sharing of terms and sharing of constraints, but it may duplicate $\mathcal{O}$-proofs [5]. To avoid this, we introduce substitutions on $\mathcal{O}$-proofs, called $\mathcal{P}$-substitutions. A $\mathcal{P}$-substitution $\sigma$ is an application from $\mathcal{X}_{\mathcal{P}}$ into $\mathcal{P}$, which can be uniquely extended into a homomorphism $\sigma : \mathcal{P} \mapsto \mathcal{P}$. Our structure for solving *gpo* ordering constraints is composed of an OCS graph representing a rewrite rule and a $\mathcal{P}$-substitution. Ordering edges of the OCS graph are labeled either by the trivial $\mathcal{O}$-proof $\top$ or by an $\mathcal{O}$-proof variable. The application of the substitution to an inequality (resp: equivalence) edge label of the graph gives an $\mathcal{O}$-proof of the corresponding inequality (resp: equivalence).

**Definition 11.** A *Structure for Ordering Constraint Solving* (SOCS for short) of a rule $l \rightarrow r$ is a pair $(G||\sigma)$ where G is an OCS graph representing the rule, and $\sigma$ is a $\mathcal{P}$-substitution.

*Example 2.* Here is a possible SOCS for the rule $h(f(x)) \rightarrow g(x)$:




$P \mapsto \Theta(f(x)) >_{lex} \Theta(g(x))$
$P' \mapsto P$

In this SOCS, the inequality edge between nodes labeled by $f$ and $g$ means that we have **at least one** possible $\mathcal{O}$-proof $P$ for $f(x) > g(x)$. On the right hand side of the SOCS, we find the related $\mathcal{P}$-substitution mapping the variable $P$ to the related $\mathcal{O}$-proof. The mapping $P' \mapsto P$ means that the $\mathcal{O}$-proof $P$ is also an $\mathcal{O}$-proof for edge $h(f(x)) \rightarrow\!\!\!\!> g(x)$.

## 4 The $\mathcal{C}$-deduction rules

We now define the deduction rules, applied on SOCS to infer constraints on $(\Theta, \geq_{lex})$ from *gpo* constraints. Let us first introduce *embedding*, which expresses a notion of sub-formula in $\mathcal{O}$-proofs.

**Definition 12.** Let $P, Q \in \mathcal{P}$ be $\mathcal{O}$-proofs. $P$ is *embedded* in $Q$, denoted $P \trianglelefteq Q$ if $(P = Q)$ or $[Q = A \ \vee \ B$ and $(P \trianglelefteq A$ or $P \trianglelefteq B)]$.

For solving *gpo* constraints on a set of rewrite rules, we start from a set of *initial SOCS*, one for each rule. Initial SOCSs are SOCSs whose OCS graphs have no ordering edge and whose $\mathcal{P}$-substitutions are empty. The *gpo* constraint solving on SOCS is achieved by a set of deduction rules. These rules transform a SOCS by adding ordering edges to the OCS graph and by constructing the corresponding $\mathcal{P}$-substitution, whose application provides the corresponding $\mathcal{O}$-proofs. Solving is processed independently for each SOCS corresponding to each rewrite rule, and ends when no deduction rule applies any longer. Let us denote by $\mathcal{C}$ the set of deduction rules and by $\mathcal{C}$-deduction process the deduction process defined by $\mathcal{C}$. The set of $\mathcal{C}$-deduction rules is given in Figure 1, where an edge $\xrightarrow[\sim]{> P}$ denotes either $\xrightarrow{> P}$ or $\xrightarrow{\sim P}$. Let $(\alpha||\nu), (\beta||\delta)$ be SOCS. A deduction rule $\frac{\alpha||\nu}{\beta||\delta}$ of $\mathcal{C}$ *matches* a SOCS $(G||\sigma)$ if $\alpha$ is a pattern of $G$ (i.e. if $\alpha = (V_\alpha, E_\alpha)$ and $G = (V_G, E_G)$, there exists a bijection from $V_\alpha$ into $V'_G$ and a bijection from $E_\alpha$ into $E'_G$, where $V'_G \subseteq V_G$ and $E'_G \subseteq E_G$), if $\nu$ matches $\sigma$, and if the precondition of $\frac{\alpha||\nu}{\beta||\delta}$ is verified. Then, the *application* of the rule consists in replacing the pattern $\alpha$ of $G$ by the pattern $\beta$ (which can be identical) and by replacing $\nu$ by $\delta$ in $\sigma$. Each time a new ordering edge is constructed in $G$, it is supposed to be labeled by a new $\mathcal{O}$-proof variable. Note that nodes $\mathcal{F}$ and $\mathcal{G}$ of the $\mathcal{C}$-deduction rules must always match distinct nodes of $G$. This prevents from adding cyclic inequality ordering edges (always false w.r.t. *gpo*) and cyclic equivalence edges (always unnecessary for deductions).

Note also that no special strategy is required when rules are applied: neither for the choice of the pair of nodes, nor for the choice of the rule to apply. As a result, the process can be parallelized.

The set of deduction rules $\mathcal{C}$ in Figure 1 is proven sound and complete in [5].

**Theorem 13.** *[5](Complexity) Let $l \rightarrow r$ be a rewrite rule, $(G||\sigma)$ the initial SOCS of $l \rightarrow r$, $N$ the number of nodes of $G$, and $M$ the non-zero maximal arity of function symbols of the rule. The complexity in time and space of the $\mathcal{C}$-deduction process starting from $(G||\sigma)$ is polynomial in $N$ and $M$ in the worst case.*

As explained above, each rule of a rewrite system is treated independently. For constraint solving on the whole set of rules, we have to gather the results relative to rules.

**Definition 14.** Let $R$ be a rewrite system $(l_i \rightarrow r_i,\ i = 1 \ldots n)$ whose SOCSs $(G_i \ || \ \sigma_i)$, representing the rules $l_i \rightarrow r_i$ are in $\mathcal{C}$-normal form. Let $P_i$ be the $\mathcal{O}$-proof label of the edge $l_i \rightarrow\!\!> r_i$ in $G_i$ for any $i = 1 \ldots n$. The *global $\mathcal{O}$-proof* of $R$ is the $\mathcal{O}$-proof: $P_1\sigma_1 \ \wedge \ \ldots \ \wedge \ P_n\sigma_n$.

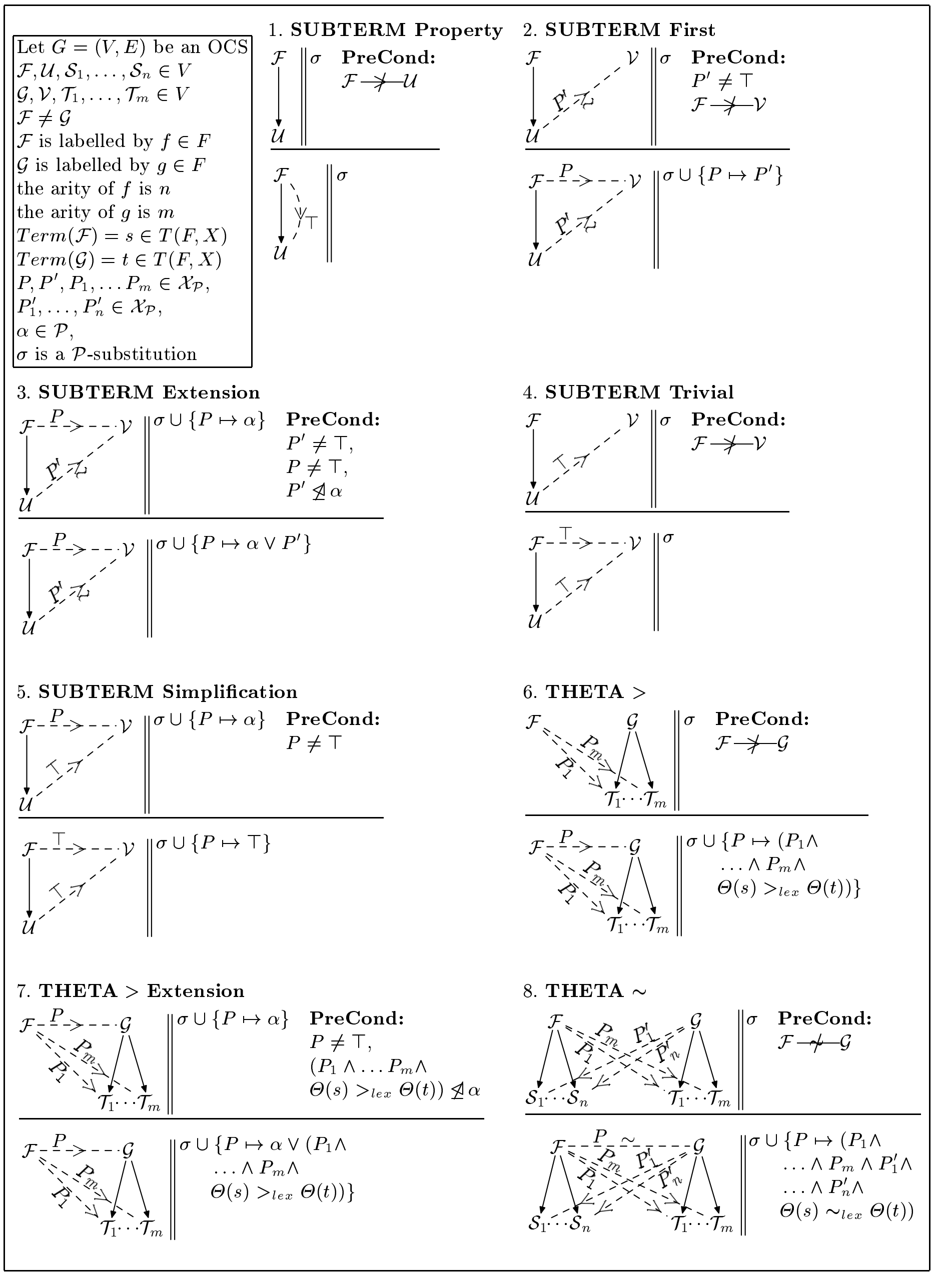



**Figure 1:** The $\mathcal{C}$-deduction rules

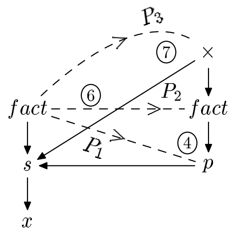
Note that if there is a rule $l_i \to r_i$ such that there is no edge $l_i \xrightarrow{P_i} r_i$ in $G_i$, then there is no possible termination proof with *gpo* for the whole TRS $R$. Definition 14 shows that $\mathcal{O}$-proofs offer a nice method for dealing with the problem of incrementally adding rules in TRS. This feature can be very useful for completion procedures. Note also that, in a SOCS, we generate inequality edges and equivalence edges for the two possible orientations of the rewrite rules (left to right and right to left).

## 5 An example of $\mathcal{C}$-deduction process

Consider the following system, borrowed from [2], for computing factorial in unary arithmetic. Let $R$ be:

$$\begin{array}{llll}
p(s(x)) \to x & (1) & s(x) \times y \to (x \times y) + y & (5) \\
fact(0) \to s(0) & (2) & x + 0 \to x & (6) \\
fact(s(x)) \to s(x) \times fact(p(s(x))) & (3) & x + s(y) \to s(x+y) & (7) \\
0 \times y \to 0 & (4) & &
\end{array}$$

The termination of $R$ cannot be proven with a simplification ordering since rule (3) is self-embedded. However it is possible to prove termination of $R$ with *gpo*. The resolution process on the initial SOCS representing the rule (3) gives the following SOCS:




$$\begin{array}{l}
P_1 \mapsto \Theta(fact(s(x))) >_{lex} \Theta(p(s(x))) \\
P_2 \mapsto P_1 \wedge \Theta(fact(s(x))) >_{lex} \Theta(fact(p(s(x)))) \\
P_3 \mapsto P_2 \wedge \Theta(fact(s(x))) >_{lex} \Theta(s(x) \times fact(p(s(x))))
\end{array}$$

Its complete construction, not detailed here by lack of place, can be found in [5]. In the following, $P_{(1)}, \ldots, P_{(7)}$ denote the $\mathcal{O}$-proofs for rules (1) to (7) respectively and $\sigma$ the $\mathcal{P}$-substitution of the final SOCS for rule (3). Since after deduction on the SOCS for rule (3), we obtain $fact(s(x)) \xrightarrow{P_3} s(x) \times fact(p(s(x)))$, the $\mathcal{O}$-proof $P_{(3)}$ for rule (3) is $P_3\sigma$. Note that $\mathcal{O}$-proofs for rules (1), (4) and (6) are trivial ones. Thus, the global $\mathcal{O}$-proof for the complete TRS is $P_{(2)} \wedge P_{(3)} \wedge P_{(5)} \wedge P_{(7)}$.

## 6 Proving satisfiability of $\mathcal{O}$-proofs

At this stage of the solving process, we have obtained a set of saturated SOCS (one for each rewrite rule), with non-instantiated $\mathcal{O}$-proofs in the $\mathcal{P}$-substitution part: no assumption is made on $\Theta$ nor on $\geq_{lex}$. The next step of our solving process consists of proving the satisfiability of an $\mathcal{O}$-proof $P$ by finding solutions, i.e. particular values $\Phi = (\mathcal{T}_{0,k}, \succsim_{lex})$ of $(\Theta, \geq_{lex})$ such that $\Phi \models P$. Let us show how to proceed in practice for verifying the satisfiability of an $\mathcal{O}$-proof, using the *partial instantiation* process we now define. In the following, an *instantiated literal* (resp. *non-instantiated literal*) of an $\mathcal{O}$-proof is either of the form $\tau_i(s) \succ_i \tau_i(t)$ or $\tau_i(s) \approx_i \tau_i(t)$ (resp. $\Theta_{i,j}(s) >_{lex} \Theta_{i,j}(t)$ or $\Theta_{i,j}(s) \simeq_{lex} \Theta_{i,j}(t)$) where $\tau_i$ are termination functions and $\succ_i$ are associated orderings.

**Definition 15.** Given $0 \leq i < j$, a $\Theta_{i,j}$ *$\mathcal{O}$-proof* is an $\mathcal{O}$-proof whose every non-instantiated literal is either of the form $\Theta_{i,j}(s) >_{lex} \Theta_{i,j}(t)$ or of the form $\Theta_{i,j}(s) \simeq_{lex} \Theta_{i,j}(t)$ where $s, t$ are terms of $T(F, X)$.

**Definition 16.** Given $0 \leq i < j$ and $P$ a $\Theta_{i,j}$ $\mathcal{O}$-proof, a *left partial instantiation* (LPI for short) of $P$ is obtained by instantiating every $\theta_i$ in $P$ by a particular termination function $\tau_i$.

Note that if we consider an $\mathcal{O}$-proof $\Theta_{i,j}(s) >_{lex} \Theta_{i,j}(t)$, its LPI is $\tau_i(s) \succ_i \tau_i(t) \vee [\tau_i(s) \approx_i \tau_i(t) \wedge \Theta_{i+1,j}(s) >_{lex} \Theta_{i+1,j}(t)]$. If we consider an $\mathcal{O}$-proof $\Theta_{i,j}(s) \simeq_{lex} \Theta_{i,j}(t)$, its LPI is $\tau_i(s) \approx_i \tau_i(t) \wedge \Theta_{i+1,j}(s) \simeq_{lex} \Theta_{i+1,j}(t)$. A practical method for finding a solution to our constraint problem in a global $\mathcal{O}$-proof thanks to LPI can be based on DAGs. An *$\mathcal{O}$-proof DAG* is an and-or DAG representing an $\mathcal{O}$-proof where a conjunctive $\mathcal{O}$-proof $\alpha \wedge \beta$ is represented by the DAG $\begin{array}{c}A\\ |\\ B\end{array}$, a disjunctive $\mathcal{O}$-proof $\alpha \vee \beta$ is represented by the DAG $A\ B$ ; $A, B$ are DAGs representing $\alpha$ and $\beta$ respectively.

**Definition 17.** Given $0 \leq i < j$ and $G$ a $\Theta_{i,j}$ $\mathcal{O}$-proof DAG, an *i-path* of $G$ is a pair $(p, A)$ where $p$ is a path from top to bottom of $G$, and $A$ is a tuple of sets $\langle A_0, \ldots, A_{i-1}, A_i \rangle$, where $A_u$ $(0 \leq u \leq i-1)$ is the set $\{\alpha | \alpha \in p, \alpha = \tau_u(s) >_u \tau_u(t)$ or $\alpha = \tau_u(s) \simeq_u \tau_u(t), s, t \in T(F, X)\}$ and $A_i = \{\alpha | \alpha \in p, \alpha = \Theta_{i,j}(s) >_{lex} \Theta_{i,j}(t)$ or $\alpha = \Theta_{i,j}(s) \simeq_{lex} \Theta_{i,j}(t), s, t \in T(F, X)\}$.

**Definition 18.** Let $S$ be a finite set. A set of inequalities and equalities $A = \{\alpha \succ \beta | \alpha, \beta \in S\} \cup \{\alpha \approx \beta | \alpha, \beta \in S\}$ is *compatible* if there exists a quasi-ordering $\succsim_S$ on $S$ such that $\alpha \succ \beta \in A \implies \alpha \succ_S \beta$ and $\alpha \approx \beta \in A \implies \alpha \approx_S \beta$ (where $\succsim_S$ stands for $\succ_S \cup \approx_S$).

Informally, an $\mathcal{O}$-proof contains a solution if its $\mathcal{O}$-proof DAG contains an i-path from top to bottom, whose literals are instantiated and whose sets are compatible. Let us now introduce the notion of *minimal i-path*, minimizing the set of constraints on non-instantiated termination functions $\Theta_{i,k}$ and related ordering $>_{lex}$.

**Definition 19.** Let $(p, A)$ be an i-path, where $A = \langle A_0, \ldots, A_{i-1}, A_i \rangle$. The i-path $(p, A)$ is *minimal* if $A_0, \ldots, A_{i-1}$ are compatible sets, and if there exists no i-path $(p', B)$, such that $B = \langle B_0, \ldots, B_{i-1}, B_i \rangle$, where $B_0, \ldots, B_{i-1}$ are compatible sets and $B_i \subset A_i$.

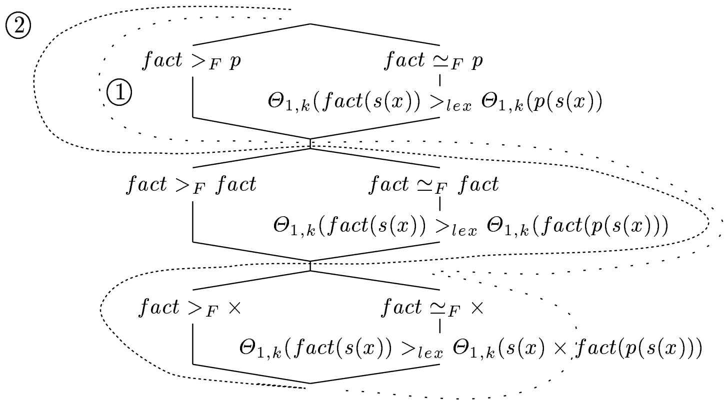
Note that, in general, a minimal i-path is not unique.

**Definition 20.** A *satisfiable i-path* is a minimal i-path $(p, \langle A_0, \ldots, A_i \rangle)$ where $A_i = \emptyset$.

A satisfiable i-path in an $\mathcal{O}$-proof DAG represents a solution of the related $\mathcal{O}$-proof. We now illustrate these definitions, on our previous example. In Section 5, we obtained $P_{(3)} = P_3\sigma$ where $\sigma$ denotes the $\mathcal{P}$-substitution of the final SOCS for rule (3). $P_3\sigma$ can be represented by the $\mathcal{O}$-proof DAG:

$$\begin{array}{c} \Theta(fact(s(x))) >_{lex} \Theta(p(s(x))) \\ | \\ \Theta(fact(s(x))) >_{lex} \Theta(fact(p(s(x)))) \\ | \\ \Theta(fact(s(x))) >_{lex} \Theta(s(x) \times fact(p(s(x)))) \end{array}$$

Recall that $\Theta$ is a simplified notation for $\Theta_{0,k}$. In order to find a specific $\Theta_{0,k}$ and a related $>_{lex}$ satisfying this $\mathcal{O}$-proof, we apply a left partial instantiation on $\Theta_{0,k}$. Let $(\theta_0, \geq_0)$ be a precedence: $\theta_0$ is a function mapping any term to its root symbol, and $\geq_0$ is an ordering on $F$, still unknown, that we want to infer automatically. Left partial instantiation applied to the previous $\mathcal{O}$-proof DAG leads to the following $\mathcal{O}$-proof DAG:




We now search for a solution, which has to be a satisfiable i-path. The paths labeled by ① and ②, among others, are 1-paths. Path ① is associated with the tuple $A = \langle A_0, A_1 \rangle$ where $A_0 = \{fact >_F p, fact \simeq_F fact, fact \simeq_F \times\}$ and $A_1 = \{\Theta_{1,k}(fact(s(x))) >_{lex} \Theta_{1,k}(fact(p(s(x)))), \Theta_{1,k}(fact(s(x))) >_{lex} \Theta_{1,k}(s(x) \times fact(p(s(x))))\}$. Path ② is associated with tuple $B = \langle B_0, B_1 \rangle$ where $B_0 = \{fact >_F p, fact \simeq_F fact, fact >_F \times\}$ and $B_1 = \{\Theta_{1,k}(fact(s(x))) >_{lex} \Theta_{1,k}(fact(p(s(x))))\}$. Sets $A_0$ and $B_0$ are both compatible. However, the 1-path ① is not minimal since $B_1 \subset A_1$. In this particular example, there is a unique minimal 1-path which is ②. Note that there is no satisfiable 1-path in this $\mathcal{O}$-proof DAG since $B_1 \neq \emptyset$. We then search for a satisfiable 2-path. Since ② is the unique minimal 1-path, and since a minimal 2-path is deduced from a minimal 1-path, we start from ② to deduce a minimal 2-path $(p, \langle B_0, B_1', B_2' \rangle)$ by applying an additional LPI.

Note that achieving partial instantiation with precedence, testing the compatibility of $A_0$ and $B_0$, and comparing 1-paths with respect to $\subset$, can be automatized. Thus the deduction of minimal 1-paths can be achieved automatically.

Finding a minimal 1-path allows us to separate the termination proof in two parts: a first part which can be automatically solved (we deduced a precedence), and a second part requiring human expertise. In our example, the proof requiring human expertise is satisfiability of the formula in $B_1$ to infer $B'_1$ and $B'_2$. We then apply left partial instantiation on $B_1$ and search for a satisfiable 2-path. For $\theta_1$, the user may choose the function interpreting *fact* as factorial, $s$ as successor, $p$ as predecessor and 0 as zero, and for $\geq_1$, he may choose $\geq_{\mathcal{N}}$: the greater or equal relation on natural numbers, as in [2]. In the $\mathcal{O}$-proof DAG, $\Theta_{1,k}(fact(s(x))) >_{lex} \Theta_{1,k}(fact(p(s(x))))$ becomes:

$$(x+1)! > x! \qquad (x+1)! = x!$$

$$\Theta_{2,k}(fact(s(x)) >_{lex} \Theta_{2,k}(fact(p(s(x)))$$

Then, validity of $(x+1)! >_{\mathcal{N}} x!$ has to be proved by the user. If we choose an interpretation where constants are interpreted as natural numbers, then $(x+1)! >_{\mathcal{N}} x!$ is valid. Thus, we get a satisfiable 2-path associated with the tuple:
$\langle\{fact >_F p, fact >_F \times\}, \{(x+1)! >_{\mathcal{N}} x!\}, \{\}\rangle$.

If we proceed similarly on the global $\mathcal{O}$-proof DAG for the whole TRS, the algorithm ends with a satisfiable 2-path for the global $\mathcal{O}$-proof DAG, which is: $\langle\{fact >_F s, fact >_F p, fact \simeq_F fact, fact >_F \times, \times >_F +, \times \simeq_F \times, + >_F s, + \simeq_F +\}, \{(x+1)! >_{\mathcal{N}} x!, (x+1) \times y >_{\mathcal{N}} x \times y, x+y+1 >_{\mathcal{N}} x+y\}, \{\}\rangle$. For details, see [5].

Note that in particular cases of *gpo*, like *lpo*, where the compatibility testing of every *gpo* termination function is automatic, the whole *gpo* solving process can be automatically achieved. For the *lpo* case, starting from a set of inequalities representing the rules of a TRS, the algorithm provides a precedence proving termination of the initial TRS (if such a precedence exists). An implementation of the *lpo* case, providing a decision procedure for the existence of a *lpo* for a given TRS has been developed in ECLiPSe[1]. See [5] for examples of execution on big size conditional and unconditional TRSs. Let us cite another approach to find a precedence for syntactical orderings like *lpo* or *rpo* [4]. However, this method, unlike ours, is not goal directed since the search for a precedence is not guided by the inequalities to be proved. In the case where solving cannot be fully automatic, the interest of our approach is that the process focusses user's effort to the key parts of the proof, by automatically proving simple properties and extracting difficult ones.

## 7 Perspectives

In this paper, we proposed a termination proof algorithm for rewrite rule systems using *gpo* constraint solving on OCS graphs, a shared term data structure defined to represent constraints. Next prospects are the improvement of $\mathcal{O}$-proof satisfiability. We are studying how to automatize satisfiability procedures for more

[1] ECRC Common Logic Programming System

syntactic and semantic termination functions. For instance, automatic polynomial termination functions, based on [6, 15], could certainly be integrated. We are also studying how to combine completion on SOUR Graphs [11] with automatic termination proofs, taking advantage of the similarity between the graph deduction process on SOCS and on SOUR.


### Acknowledgments

We would like to thank Hélène Kirchner, Claude Kirchner, Nachum Dershowitz, Christopher Lynch, Polina Strogova and Christophe Ringeissen for comments on this paper.


## References


1. H. Comon. Solving inequations in term algebras. In *Proc. 5th LICS Symp., Philadelphia (Pa., USA)*, pages 62–69, June 1990.
2. N. Dershowitz and C. Hoot. Natural termination. *TCS*, 142(2):179–207, May 1995.
3. N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 6, pages 244–320. Elsevier Science Publishers B. V. (North-Holland), 1990.
4. R. Forgaard and D. Detlefs. An incremental algorithm for proving termination of term rewriting systems. In J.-P. Jouannaud, editor, *Proc. 1st RTA Conf., Dijon (France)*, pages 255–270. Springer-Verlag, 1985.
5. T. Genet and I. Gnaedig. Termination proofs using gpo ordering constraints (extended version). Technical report, INRIA, 1997. RR-3087, available at `http://www.loria.fr/equipe/protheo.html`.
6. J. Giesl. Generating polynomial orderings for termination proofs. In J. Hsiang, editor, *Proc. 6th RTA Conf., Kaiserslautern (Germany)*, volume 914 of *LNCS*. Springer-Verlag, 1995.
7. P. Johann and R. Socher-Ambrosius. Solving simplification ordering constraints. In J.-P. Jouannaud, editor, *Proc. 1st CCL Conf., Munich (Germany)*, volume 845 of *LNCS*, pages 352–367. Springer-Verlag, 1994.
8. S. Kamin and J.-J. Lévy. Attempts for generalizing the recursive path ordering. Unpublished manuscript, 1980.
9. D. S. Lankford. On proving term rewriting systems are noetherian. Technical report, Louisiana Tech. University, Mathematics Dept., Ruston LA, 1979.
10. P. Lescanne. On the recursive decomposition ordering with lexicographical status and other related orderings. *JAR*, 6:39–49, 1990.
11. C. Lynch and P. Strogova. Sour graphs for efficient completion. Technical Report 95-R-343, CRIN, 1995.
12. R. Nieuwenhuis. Simple lpo constraint solving methods. *IPL*, 47(2), 1993.
13. R. Nieuwenhuis and A. Rubio. Theorem proving with ordering constrained clauses. In D. Kapur, editor, *Proc. 11th CADE Conf., Saratoga Springs (N.Y., USA)*, volume 607 of *LNCS*, pages 477–491. Springer-Verlag, 1992.
14. D. Plaisted. Polynomial time termination and constraint satisfaction tests. In C. Kirchner, editor, *Proc. 5th RTA Conf., Montreal (Canada)*, volume 690 of *LNCS*, pages 405–420, Montreal (Québec, Canada), June 1993. Springer-Verlag.
15. J. Steinbach. Generating polynomial orderings. *IPL*, 49:85–93, 1994.