Complete Monotonic Semantic Path Orderings

Cristina Borralleras[1], Maria Ferreira[2] and Albert Rubio[3*]

[1] Univ. de Vic, Dept. L.S.I.,
C/ de Miramarges, s/n 08500 Vic, Spain
`cristina.borralleras@uvic.es`

[2] Univ. Nova de Lisboa, Dept. Informática,
2825-114 Caparica, Portugal
`cf@di.fct.unl.pt`

[3] Universitat Politècnica de Catalunya, Dept. LSI,
Jordi Girona 1, 08034 Barcelona, Spain
`rubio@lsi.upc.es`

Abstract. Although theoretically it is very powerful, the *semantic path ordering* (SPO) is not so useful in practice, since its monotonicity has to be proved *by hand* for each concrete *term rewrite system* (TRS).
In this paper we present a *monotonic* variation of SPO, called MSPO. It characterizes termination, i.e., a TRS is terminating if and only if its rules are included in some MSPO. Hence MSPO is a *complete* termination method.
On the practical side, it can be easily automated using as ingredients standard interpretations and general-purpose orderings like RPO. This is shown to be a sufficiently powerful way to handle several non-trivial examples and to obtain methods like *dummy elimination* or *dependency pairs* (without the dependency graph refinement) as particular cases. Finally, we obtain some positive modularity results for termination based on MSPO.

1 Introduction

Rewrite systems are sets of rules (directed equations) used to compute by repeatedly replacing parts of a given formula with equal ones until the simplest possible form is obtained. Depending on the kind of objects that are rewritten there are different kinds of rewrite systems, like *string rewrite systems* (Thue or semi-Thue systems) or *term rewrite systems* (TRS; see [DJ90,Klo92,BN98] for detailed surveys).

Termination is a fundamental property for most applications of rewrite systems. Termination of a TRS is, in general, an undecidable property, even for one rule TRSs. Termination of TRSs can be proved by showing that the induced rewrite relation is included in a well-founded ordering on terms. If the ordering $\succ$ is also monotonic and stable under substitutions, i.e., a *reduction ordering*, then it suffices to check that $l \succ r$ for every rule $l \rightarrow r$ in the system.

* partially supported by the CICYT project HEMOSS ref. TIC98-0949-C02-01.

Monotonic orderings including the subterm relation are called *simplification orderings*, and their well-foundedness follows from Kruskal's theorem [Kru60]. Inside this class, path orderings, and in particular the *recursive path ordering* (RPO) [Der82], have received a special attention (see [Der87,DJ90,BN98]). Unfortunately, although these orderings are simple and easy to use, they turn out, in many cases, to be a weak termination proving tool, as there are many TRSs that are terminating but are not contained in any simplification ordering, i.e. they are not *simply terminating*.

To avoid this problem, many different transformation methods have been developed, e.g. [BD86,BL90,Zan94,FZ95,Ste95,Xi98,KNT99]. By transforming the TRS into a set of ordering constraints, the *dependency pair* method [AG97,AG00] has become a successful general technique for proving termination of (non-simply terminating) TRSs.

As an alternative to transformation methods, more powerful term orderings can be used. Due to its simplicity, the Semantic Path Ordering (SPO) ([KL80]) becomes a potential well-known candidate: in SPO the scheme of RPO is generalized by replacing the precedence on function symbols by any (well-founded) underlying (quasi-)ordering involving the whole term and not only its head symbol. Although the simplicity of the presentation is kept, this makes the ordering much more powerful. In fact, for every terminating TRS there is some SPO that includes its rewrite relation.

Unfortunately, SPO is not so useful in practice. Due to the generalization, the monotonicity property is lost, even if the underlying ordering is monotonic. Hence, it is not sufficient to check that the rules are included in the ordering to ensure termination, since this does not imply that each rewrite step is inside the ordering. In order to ensure termination of a TRS R, the user is responsible for proving (by hand) monotonicity restricted to all terms s and t such that s rewrites to t with R in one step.

In this paper we present a monotonic version of the SPO. On the one hand, this *monotonic semantic path ordering* (MSPO) is still very powerful for proving termination of TRSs: on the theoretical side, it also characterizes termination, i.e. a TRS is terminating if and only if the rules are included in some MSPO; and on the practical side, it generalizes most of the (automatable) termination proof methods. On the other hand, termination can be automatically checked once the ingredients, i.e. the underlying (base) quasi-orderings, of the MSPO are provided.

The first and only other, as far as we know, monotonic version of SPO is due to Geser [Ges92]. On the one hand, this proposal is not as general as ours (in fact, as we will show, it does not characterize termination), and on the other hand, it is less suitable for practical implementations. In section 5.3 a detailed comparison with this work is provided.

We are not only interested in using MSPO for checking termination, but also for proving termination automatically. Since we cannot expect to automatically generate an adequate MSPO for a given TRS whenever it exists, we have studied particular classes of underlying quasi-orderings, which can be automatically generated. As a hint of the power of the resulting family of MSPO's, it is shown that some known methods like *dummy elimination* [FZ95] or *dependency pairs* [AG97,AG00] (without the dependency graph refinement) are particular instances.

Using these classes of underlying quasi-orderings the termination proofs of several non-simply terminating, but terminating, term rewriting systems can be fully automated. However, some heuristics for choosing the underlying quasi-ordering have to be developed in order to make the method more effective in practice. On the other hand, due to its generality and its additional flexibility, MSPO provides, in some cases, simpler termination proofs than the dependency pair method (see example 6). A first system based on MSPO has been developed by which many examples, including the ones in the paper, have been checked. The software and examples are available at `www.lsi.upc.es/~albert`.

Additionally, applying known abstract sufficient conditions ensuring modularity of termination [Ohl94,Gra94], modularity results for termination based on MSPO are obtained. In particular, for TRSs proved terminating using MSPO with the aforementioned classes of underlying quasi-orderings, termination is proved to be modular for disjoint systems and for finite constructor-sharing systems. Note that these modularity properties are crucial for many practical applications of automatic termination proof systems.

Formal definitions and basic tools are introduced in section 2. In section 3 we present and study the monotonic semantic path ordering. Section 4 is devoted to examples. Other termination methods are analyzed in section 5. Section 6 presents an ordering constraint solving approach for proving termination using MSPO. In section 7 modularity results are presented. Some conclusions are given in section 8.

2 Preliminaries

In the following we consider that $\mathcal{F}$ is a set of function symbols, $\mathcal{X}$ a set of variables and $\mathcal{T}(\mathcal{F},\mathcal{X})$ is the set of terms built from $\mathcal{F}$ and $\mathcal{X}$. Let s and t be arbitrary terms in $\mathcal{T}(\mathcal{F},\mathcal{X})$, let f be a function symbol in $\mathcal{F}$ and let σ be a substitution. A (strict partial) ordering $\succ$ is a transitive irreflexive relation. It is *monotonic* if $s \succ t$ implies $f(\dots s \dots) \succ f(\dots t \dots)$, and *stable under substitution* if $s \succ t$ implies $s\sigma \succ t\sigma$. Monotonic orderings that are stable under substitutions are called *rewrite orderings*. A *reduction ordering* is a rewrite ordering that is *well-founded*: there are no infinite sequences $t_1 \succ t_2 \succ \dots$

The reflexive-transitive closure of a binary relation $\to$ is denoted by $\xrightarrow{*}$ and the transitive closure by $\xrightarrow{+}$.

A term rewrite system (TRS) is a (possibly infinite) set of rules $l \to r$ where l and r are terms. Given a TRS R, s rewrites to t with R, denoted by $s \to_R t$, if there is some rule $l \to r$ in R, $s|_p = l\sigma$ for some position p and substitution σ and $t = s[r\sigma]_p$. A TRS R is terminating if there exists no infinite sequence $t_1 \to_R t_2 \to_R \dots$ Thus, the transitive closure $\xrightarrow{+}_R$ of any terminating TRS is a reduction ordering. Furthermore, reduction orderings characterize termination of TRSs.

Theorem 1. *A rewrite system R is terminating if and only if all rules are contained in a reduction ordering $\succ$, i.e., $l \succ r$ for every $l \rightarrow r \in R$.*

Another interesting property of reduction orderings is that they can be combined with the subterm relation $\rhd$ without loosing well-foundedness. However, note that in general, monotonicity will be lost, since we only add the subterm relation, and not its monotonic closure.

Proposition 1. *If $\succ$ is a reduction ordering then $\succ \cup \rhd$ is well-founded.*

Given a relation $\succ$, the multiset extension of $\succ$ on finite multisets, denoted by $\succ\!\succ$, is defined as the smallest transitive relation containing

$$X \cup \{s\} \ \succ\!\succ X \cup \{t_1, \ldots, t_n\} \qquad \text{if } s \succ t_i \text{ for all } i \in \{1 \ldots n\}$$

If $\succ$ is a well-founded ordering on terms then $\succ\!\succ$ is a well-founded ordering on finite multisets of terms.

A *quasi-ordering* $\succeq$ is a transitive and reflexive binary relation. Its inverse is denoted by $\preceq$. Its *strict part* $\succ$ is the strict ordering $\succeq \setminus \preceq$ (i.e, $s \succ t$ iff $s \succeq t$ and $s \not\preceq t$). Its *equivalence* $\sim$ is $\succeq \cap \preceq$. Note that $\succeq$ is the disjoint union of $\succ$ and $\sim$, and that if $=$ denotes syntactic equality then $\succ \cup =$ is a quasi-ordering whose strict part is $\succ$.

Notation: In the remainder of this paper, $\succeq$ (possibly with subscripts) will always denote a quasi-ordering.

The following definitions for $\succeq$ will be used:

1. $\succeq$ is *monotonic* if $\succ$ is.
2. $\succeq$ is *well-founded* if $\succ$ is.
3. $\succeq$ is *stable under substitutions* if $\succ$ is and $s\sigma \succeq t\sigma$ whenever $s \succeq t$.
4. $\succeq$ is *quasi-monotonic* if $f(\ldots, s, \ldots) \succeq f(\ldots, t, \ldots)$ whenever $s \succeq t$.
5. $\succeq$ is a *quasi-reduction quasi-ordering* if it fulfills the above properties 2, 3 and 4.

Note that if $\succeq$ is quasi-monotonic then $\succ$ is not necessarily monotonic.

The *lexicographic combination* of $\succeq_1, \ldots, \succeq_n$, denoted by $(\succeq_1, \ldots, \succeq_n)_{lex}$, is defined as usual as, $s(\succeq_1, \ldots, \succeq_n)_{lex} t$ iff either $s \succ_i t$ for some i and $s \succeq_j t$ for all $j < i$, or $s \succeq_i t$ for all i.

If all $\succeq_i$ are well-founded (stable under substitutions) then its lexicographic combination also is.

A *precedence* $\succeq_{\mathcal{F}}$ is a well-founded quasi-ordering on $\mathcal{F}$.

Given a precedence $\succeq_{\mathcal{F}}$, the *recursive path ordering* (RPO), denoted by $\succ_{rpo}$, is defined recursively as follows:

$s = f(s_1, \ldots, s_m) \succ_{rpo} t$ iff

1. $s_i \succeq_{rpo} t$, for some $i = 1, \ldots, m$, or
2. $t = g(t_1, \ldots, t_n)$ with $f \succ_{\mathcal{F}} g$ and $s \succ_{rpo} t_i$ for all $i = 1 \ldots n$ or
3. $t = g(t_1, \ldots, t_n)$ with $f \succeq_{\mathcal{F}} g$ and $\{s_1, \ldots, s_m\} \succ\!\succ_{rpo} \{t_1, \ldots, t_n\}$.

where $\succeq_{rpo}$ is defined as $\succ_{rpo} \cup =$.

3 A monotonic Semantic Path Ordering

Let us now recall the definition of the semantic path ordering. Then we will analyze an example of non-monotonicity, which will provide the intuition behind the monotonic version we will propose.

Definition 1. *Given $\succeq_Q$, called the underlying (or base) quasi-ordering, the* semantic path ordering *(SPO) [KL80], denoted as $\succ_{spo}^Q$, is defined as*

$s = f(s_1, \ldots, s_m) \succ_{spo}^Q t$ *iff*

1. $s_i \succeq_{spo}^Q t$, *for some* $i = 1, \ldots, m$, *or*
2. $s \succ_Q t = g(t_1, \ldots, t_n)$ *and* $s \succ_{spo}^Q t_i$ *for all* $i = 1, \ldots, n$, *or*
3. $s \succeq_Q t = g(t_1, \ldots, t_n)$ *and* $\{s_1, \ldots, s_m\} \succ\!\!\succ_{spo}^Q \{t_1, \ldots, t_n\}$,

where $\succeq_{spo}^Q$ is defined as $\succ_{spo}^Q \cup =$.

The semantic path ordering is well-defined, which can be easily proved by induction on sum of the sizes of s and t, and fulfills the following property.

Lemma 1 ([KL80]). *If $\succeq_Q$ is well-founded and stable under substitutions then $\succ_{spo}^Q$ is a well-founded ordering stable under substitutions.*

But, as said before, the semantic path ordering is, in general, non-monotonic, even when $\succeq_Q$ is quasi-monotonic (in fact, the same problem appears if $\succeq_Q$ is monotonic). This is shown in the following example.

Example 1. Consider the following quasi-ordering $\succeq_Q$ defined for all terms s and t as: (i) $f(s) \succeq_Q g(t)$; (ii) $f(s) \succeq_Q f(t)$ iff $s \succeq_Q t$; and (iii) $g(s) \succeq_Q g(t)$, and reflexive in variables and constants.

This quasi-ordering is well-founded, since its strict part is $u[f(s)] \succ_Q u[g(t)]$ for all non-empty contexts u containing only the symbol f and for all terms s and t (note that the length of any decreasing sequence is at most the number of f's above the first g symbol of the initial term in the sequence); it is stable under substitutions and quasi-monotonic. However, the induced SPO is not monotonic: by case 1 of SPO we have $g(f(a)) \succ_{spo}^Q f(a)$, but by adding the context $f([])$ onto both terms we have $f(g(f(a))) \not\succ_{spo}^Q f(f(a))$ (in fact, in this case we can even prove $f(f(a)) \succ_{spo}^Q f(g(f(a)))$).

Analyzing this example we can observe that, even if $\succeq_Q$ is quasi-monotonic (or monotonic), since in case 1 of SPO we do not require $s \succeq_Q t$, it may happen that by adding some context u, if we do not have $u[s] \succeq_Q u[t]$, we cannot apply any of the cases. Then, in order to ensure monotonicity of $\succ_{spo}^Q$, we need to be sure that $u[s] \succeq_Q u[t]$ for any context. A way to obtain this is to require always $s \succeq_Q t$, provided that $\succeq_Q$ is quasi-monotonic, that is defining a new ordering $\succ_M$ as $s \succ_M t$ iff and only if $s \succeq_Q t$ and $s \succ_{spo}^Q t$.

However, requiring always $s \succeq_Q t$ can be a bit too strong and, in fact, from the example above we only need to have $u[s] \succeq_Q u[t]$ for every (non-empty) context u. Hence instead of requiring $s \succeq_Q t$, we will ask for something weaker that ensures $u[s] \succeq_Q u[t]$.

Definition 2. *We say that* $\succeq_I$ *is* quasi-monotonic on $\succeq_Q$ *(or* $\succeq_Q$ *is* quasi-monotonic wrt. $\succeq_I$*) if*

$$s \succeq_I t \quad \textit{implies} \quad f(\ldots s \ldots) \succeq_Q f(\ldots t \ldots)$$

for all terms s *and* t *and function symbols* f.

Definition 3. *A pair* $\langle \succeq_I, \succeq_Q \rangle$ *is called a* quasi-reduction pair *if* $\succeq_I$ *is quasi-monotonic,* $\succeq_Q$ *is well-founded, both are stable under substitutions and* $\succeq_I$ *is quasi-monotonic on* $\succeq_Q$.

Now we define the *monotonic semantic path ordering* (MSPO):

Definition 4. *Let* $\langle \succeq_I, \succeq_Q \rangle$ *be a quasi-reduction pair. The corresponding monotonic semantic path ordering, denoted by* $\succ_{mspo}$*, is defined as:*

$$s \succ_{mspo} t \quad \textit{if and only if} \quad s \succeq_I t \quad \textit{and} \quad s \succ^Q_{spo} t$$

for all terms s *and* t.

Theorem 2. $\succ_{mspo}$ *is a reduction ordering.*

Proof. Well-foundedness follows from the fact that $\succ_{mspo} \subseteq \succ^Q_{spo}$ and $\succ^Q_{spo}$ is well-founded. Transitivity and stability under substitutions follow respectively from the transitivity and the stability of $\succeq_I$ and $\succ^Q_{spo}$.

For monotonicity we have to show that $s \succ_{mspo} t$ implies $f(\ldots, s, \ldots) \succ_{mspo} f(\ldots, t, \ldots)$, that is $f(\ldots, s, \ldots) \succeq_I f(\ldots, t, \ldots)$ and also $f(\ldots, s, \ldots) \succ^Q_{spo} f(\ldots, t, \ldots)$, for all terms s and t and function symbols f.

By definition of $\succ_{mspo}$, $s \succ_{mspo} t$ implies $s \succeq_I t$. Hence, by the quasi-monotonicity of $\succeq_I$, we have $f(\ldots, s, \ldots) \succeq_I f(\ldots, t, \ldots)$. On the other hand, by quasi-monotonicity of $\succeq_I$ on $\succeq_Q$, we have $f(\ldots, s, \ldots) \succeq_Q f(\ldots, t, \ldots)$. Then, by definition of $\succ_{mspo}$, $s \succ_{mspo} t$ implies $s \succ^Q_{spo} t$ and, therefore we have $\{\ldots s \ldots\} \succ\!\succ^Q_{spo} \{\ldots t \ldots\}$, which implies $f(\ldots, s, \ldots) \succ^Q_{spo} f(\ldots, t, \ldots)$ by case (3). □

The previous theorem shows that $\succ_{mspo}$ provides a correct method for proving termination of term rewriting systems. The following result shows that it is also a complete method, i.e., for any terminating term rewriting there is a monotonic semantic path ordering that includes its rules. Therefore $\succ_{mspo}$ characterizes termination. The proof is very similar to the completeness proof of SPO and also to the completeness proof of other methods like semantic labeling in [MHH96].

Theorem 3. *A rewrite system* R *is terminating if and only if there exists some quasi-reduction pair* $\langle \succeq_I, \succeq_Q \rangle$*, s.t.* $l \succ_{mspo} r$ *for every rule* $l \to r$ *in* R.

Proof. The right to left implication follows from Theorem 2. For the left to right, we will build an appropriate pair $\langle \succeq_I, \succeq_Q \rangle$ for any terminating TRS R.

Let $\succeq_I$ be $\rightarrow_R^*$ and let $\succeq_Q$ be $(\rightarrow_R \cup \triangleright)^*$. By definition of rewriting $\succeq_I$ is quasi-monotonic and both $\succeq_I$ and $\succeq_Q$ are stable under substitutions. By termination of R and Property 1, the strict part of $\succeq_Q$, i.e. $(\rightarrow_R \cup \triangleright)^+$, is well-founded. Finally, since $\succeq_I \subseteq \succeq_Q$, and $\succeq_I$ is quasi-monotonic, we have that $\succeq_I$ is quasi-monotonic on $\succeq_Q$.

Therefore, since $\langle \succeq_I, \succeq_Q \rangle$ is a quasi-reduction pair, we only have to prove that $l \succ_{mspo} r$ for every rule $l \rightarrow r$ in R. By definition, we have $l \succeq_I r$ and $l \succ_Q r$, and, moreover, for every subterm r' of r we have $l \succ_Q r'$, since $l \rightarrow_R r \triangleright r'$. Therefore, we have $l \succeq_I r$ and $l \succ_{spo}^Q r$, by repeatedly applying case 2, which implies $l \succ_{mspo} r$. $\square$

The quasi-reduction pair condition is quite tight: as shown in the following example, if $\succeq_Q$ is required to be a quasi-reduction quasi-ordering then even SPO does not characterize termination, i.e. there are terminating sets of rules that are not included in any SPO with an underlying quasi-reduction quasi-ordering.

Example 2. There is no quasi-reduction quasi-ordering $\succeq_Q$ such that the following TRS is included in the generated SPO.

$$\begin{array}{ll} a & \rightarrow b \\ f(b) & \rightarrow g(a) \\ h(a) & \rightarrow h(f(b)) \end{array}$$

First, note that the TRS is terminating (it will be shown using MSPO in example 3). Now we show that these rules cannot be included in SPO with a quasi-reduction quasi-ordering $\succeq_Q$. To this end it is important to remark that in SPO to have $c \succeq_{spo}^Q t$ for some constant c and term t we need $c \succ_Q t'$ for all t' subterm of t.

By definition we need $a \succ_Q b$ to be able to include the first rule. Then to include the second one we need $f(b) \succ_Q g(a)$ and $f(b) \succeq_Q a$, in order to conclude by case 2 first and then by case 2 or 3 for the recursive call with $f(b) \succ_{spo}^Q a$, since any other possibility requires $b \succeq_Q a$, which contradicts the first assumption. Now we have (i) $a \succ_Q b$, (ii) $f(b) \succ_Q g(a)$ and (iii) $f(b) \succeq_Q a$. We proceed with the third rule. If $a \succeq_{spo}^Q h(f(b))$ we need $a \succ_Q f(b)$, which contradicts assumption (iii). Otherwise we need $h(a) \succeq_Q h(f(b))$. By assumption (iii) and quasi-monotonicity of $\succeq_Q$, we have $h(f(b)) \succeq_Q h(a)$, which implies $h(a) \not\succ_Q h(f(b))$. Therefore we have to apply case 3 of SPO, and hence we need $a \succ_{spo}^Q f(b)$, which requires $a \succ_Q f(b)$ contradicting assumption (iii).

Theorem 3 shows the theoretical power of the ordering as a termination proof method, but in order to make it more useful in practice in the following two sections we will show general methods to obtain quasi-reduction pairs $\langle \succeq_I, \succeq_Q \rangle$.

3.1 Building $\succeq_I$

We consider $\succeq_I$ obtained by combining an interpretation I on terms with some quasi-reduction quasi-ordering $\succeq_B$, called the *basic* quasi-ordering, which can be obtained by well-known practical general-purpose methods like the path orderings or polynomial interpretations. For the interpretation I, as a general property, we require the preservation of the quasi-monotonicity and stability under substitutions of the basic quasi-ordering. Below some particular such interpretations which are suitable for practical applications are provided. These interpretations are not original; they have been used in many different transformation based termination methods, and in particular a slightly restricted version of them, called *argument filtering systems* (AFS), has been used in a similar way for the dependency pair method [AG00].

We will consider interpretations as mappings from terms to terms $I : \mathcal{T}(\mathcal{F}, \mathcal{X}) \to \mathcal{T}(\mathcal{F}', \mathcal{X}')$, although, of course, we can also consider interpretation from terms to multisets of terms or any other domain provided that the required properties are fulfilled.

From now on we consider that $\succeq_I$ is defined as follows

$$s \succeq_I t \text{ if and only if } I(s) \succeq_B I(t)$$

Note that this does not imply any loss of generality since I can be the identity mapping. Additionally, it follows that $s \succ_I t$ if and only if $I(s) \succ_B I(t)$.

For any I, if $\succeq_B$ is a quasi-ordering then $\succeq_I$ is a quasi-ordering which is well-founded if $\succeq_B$ also is. For quasi-monotonicity and stability under substitutions this is not always the case. Let us give some examples of interpretations that do preserve these properties.

Each symbol f can be interpreted either by a projection on a single argument, denoted by the pair $(f(x_1, \ldots, x_n), x_i)$, or else by a function symbol f_I applied to an arbitrary sequence obtained from the arguments of f, denoted by the pair $(f(x_1, \ldots, x_n), f_I(x_{i_1}, \ldots, x_{i_k}))$, for some $k \geq 0$ and $i_1, \ldots, i_k \in \{1, \ldots, n\}$. Additionally we consider I to be the identity for variables (although it can be any bijection).

Note that in the AFS the second kind of interpretations are restricted to have $f_I \notin \mathcal{F}$ and all x_{i_j} to be different variables in $\{x_1, \ldots x_n\}$, which is not the case here.

We assume that there is only one pair for each symbol. Usually the identity pairs will be omitted. Thus the interpretation I is recursively defined from these pairs as, $I(x) = x$ and $I(f(t_1, \ldots, t_n))$ is

- $I(t_i)$ if we have the pair $(f(x_1, \ldots, x_n), x_i)$, or
- $f_I(I(t_{i_1}), \ldots, I(t_{i_k}))$ if we have the pair $(f(x_1, \ldots, x_n), f_I(x_{i_1}, \ldots, x_{i_k}))$.

It is easy to show that these interpretations preserve quasi-monotonicity and stability under substitutions (recall that the stability of quasi-orderings requires also the stability of their strict parts).

Proposition 2. *Let I be an interpretation as defined above.*

1. *If $\succeq_B$ is quasi-monotonic then $\succeq_I$ is quasi-monotonic.*
2. *If $\succeq_B$ is stable under substitutions then $\succeq_I$ is stable under substitutions.*

3.2 Building $\langle \succeq_I, \succeq_Q \rangle$

In this section we show how quasi-reduction pairs $\langle \succeq_I, \succeq_Q \rangle$ can be obtained. First, as basic cases, we present two possible quasi-orderings $\succeq_Q$ fulfilling the quasi-monotonicity requirement for a given $\succeq_I$.

Proposition 3. *Let $\succeq_I$ be a quasi-reduction quasi-ordering. Then*

1. *$\langle \succeq_I, \succeq_I \rangle$ is a quasi-reduction pair.*
2. *Let $\succeq_{\mathcal{F}}$ be a precedence on $\mathcal{F}$ and let $\succeq_T^{\mathcal{F}}$ be defined as $s \succeq_T^{\mathcal{F}} t$ iff $top(s) \succeq_{\mathcal{F}} top(t)$. Then $\langle \succeq_I, \succeq_T^{\mathcal{F}} \rangle$ is a quasi-reduction pair.*

Now we show how to obtain new quasi-reduction pairs from one or several given quasi-reduction pairs. Hence, we can start by pairs as in the proposition above and then (repeatedly) apply the following properties to obtain more suitable quasi-reduction pairs.

First we define what a renaming quasi-ordering is.

Definition 5. *Let N be a mapping from $\mathcal{F}$ to $\mathcal{F}$, called a* renaming *in $\mathcal{F}$, and let f_N denote $N(f)$. We extend N to terms, obtaining a head renaming map, in the following way $N(f(t_1, \ldots, t_m)) = f_N(t_1, \ldots, t_m)$ for every symbol f in $\mathcal{F}$.*

Given $\succeq$ and a renaming map in $\mathcal{F}$, the renaming *quasi-ordering $\succeq^N$ is defined as $s \succeq^N t$ if and only if either $s = t \in \mathcal{X}$ or $N(s) \succeq N(t)$.*

Note that the renaming map is only applied to the head symbol of the term, and not to the arguments, and that, to preserve stability under substitutions, it is not defined for variables. This notion of renaming already appears in the dependency pair method, where the head symbol of both terms in each dependency pair is always renamed (see section 5.2 for details).

Proposition 4. *$\langle \succeq_I, \succeq_Q \rangle$ is a quasi-reduction pair if*

1. *$\langle \succeq_I, \succeq_{Q_0} \rangle$ is a quasi-reduction pair and $\succeq_Q$ is well-founded and stable under substitutions and $\succeq_{Q_0} \subseteq \succeq_Q$; or*
2. *$\langle \succeq_I, \succeq_{Q_0} \rangle$ is a quasi-reduction pair and $\succeq_Q$ is $\succeq_{Q_0}^N$ for some renaming map N in $\mathcal{F}$; or*
3. *$\langle \succeq_I, \succeq_{Q_i} \rangle$ is a quasi-reduction pair for all $i \in \{1 \ldots n\}$ and $\succeq_Q$ is $(\succeq_{Q_1}, \ldots, \succeq_{Q_n})_{lex}$.*

Let us show how propositions 3 and 4 can be used to build quasi-reduction pairs. Assume that $\succeq_I$ is a quasi-reduction quasi-ordering, $\succeq_{\mathcal{F}}$ is a precedence on $\mathcal{F}$ and N is a renaming map in $\mathcal{F}$. Then, by proposition 3.1, $\langle \succeq_I, \succeq_I \rangle$ is a quasi-reduction pair and, by proposition 4.2, $\langle \succeq_I, \succeq_I^N \rangle$ is a quasi-reduction pair as well. Now since, by proposition 3.2, $\langle \succeq_I, \succeq_T^{\mathcal{F}} \rangle$ is a quasi-reduction pair, we can conclude by proposition 4.3, that $\langle \succeq_I, (\succeq_I^N, \succeq_T^{\mathcal{F}})_{lex} \rangle$ is a quasi-reduction pair.

4 Examples

In the examples we will always provide the quasi-reduction pair $\langle \succeq_I, \succeq_Q \rangle$ and in some cases the details of the checking of $\succ_{mspo}$ will be included. First we give the definition of $\succeq_Q$ using $\succeq_I$ and then the definition of $\succeq_I$. In all cases we use the methods described in sections 3.1 and 3.2. Since for the basic quasi-ordering $\succeq_B$ of $\succeq_I$ we always use RPO, to avoid confusion, its precedence will be denoted by $\succeq_P$ (note that we can use other precedences to build $\succeq_Q$), and for simplicity we will directly give its strict part $\succ_P$.

Example 3. The following TRS comes from example 2

$$\begin{aligned} a &\to b \\ f(b) &\to g(a) \\ h(a) &\to h(f(b)) \end{aligned}$$

In this case for $\succeq_Q$ we use $\succeq_I^N$ with the renaming map N which is the identity except $N(f) = F$. For $\succeq_I$ we use the interpretation I generated by the pairs: $(f(x), b)$ and $(g(x), b)$; and RPO with the precedence $h \succ_P F$, $F \succ_P g$, $F \succ_P a$, $a \succ_P b$. Note that we have added to the signature the symbol F.

For the first rule we have $a \succeq_I b$ since $I(a) = a \succ_{rpo} b = I(b)$ and since $N(a) = a$ and $N(b) = b$, we have $a \succ_Q b$, which implies $a \succ_{spo}^Q b$ by case 2 of SPO, and hence $a \succ_{mspo} b$.

For the second rule $I(f(b)) = b = I(g(a))$, and hence $f(b) \succeq_I g(a)$. To prove $f(b) \succ_{spo}^Q g(a)$, since $I(N(f(b))) = F(b) \succ_{rpo} b = I(N(g(a)))$, which implies $f(b) \succ_Q g(a)$, by case 2 of SPO we only need to check $f(b) \succ_{spo}^Q a$, which follows again by case 2.

For the third rule $I(h(a)) = h(a) \succ_{rpo} h(b) = I(h(f(b)))$ and hence $h(a) \succeq_I h(f(b))$. To prove $h(a) \succ_{spo}^Q h(f(b))$, since $I(N(h(a))) = h(a) \succ_{rpo} h(b) = I(N(h(f(b))))$, which implies $h(a) \succ_Q h(f(b))$, by case 2 of SPO we only need to check $h(a) \succ_{spo}^Q f(b)$. This follows again by case 2, since $I(N(h(a))) = h(a) \succ_{rpo} F(b) = I(N(f(b)))$, and hence $h(a) \succ_Q f(b)$, and $h(a) \succ_{spo}^Q b$ follows as well by case 2.

Example 4. In the following example of nested recursion (from [FZ95]) we use a precedence $\succeq_{\mathcal{F}}$ as first component in $\succeq_Q$.

$$\begin{aligned} f(g(x)) &\to g(f(f(x))) \\ f(h(x)) &\to h(g(x)) \end{aligned}$$

We can take as $\succeq_Q$ the lexicographic combination $(\succeq_T^{\mathcal{F}}, \succeq_I^N)_{lex}$ with the precedence $f \succ_{\mathcal{F}} g$ and $f \succ_{\mathcal{F}} h$, and the renaming map N which is the identity except $N(f) = F$. For $\succeq_I$, the interpretation is given by $(f(x), x)$ and $(h(x), a)$; and the basic quasi-ordering is RPO generated by the empty precedence. Note that we have added to the signature the function symbol F and the constant a.

We only show here that $f(g(x)) \succ_{mspo} g(f(f(x)))$. First, we have $f(g(x)) \succeq_I g(f(f(x)))$, since $I(f(g(x))) = g(x) = I(g(f(f(x))))$. To prove $f(g(x)) \succ_{spo}^Q$

$g(f(f(x)))$ we apply case 2, since $f \succ_{\mathcal{F}} g$ and hence $f(g(x)) \succ_Q g(f(f(x)))$. For the recursive call $f(g(x)) \succ_{spo}^Q f(f(x))$ we apply case 2 as well, since $I(N(f(g(x)))) = F(g(x)) \succ_{rpo} F(x) = I(N(f(f(x))))$ and hence $f(g(x)) \succ_Q g(f(f(x)))$. For the recursive call $f(g(x)) \succ_{spo}^Q f(x)$ we apply again case 2, since $I(N(f(g(x)))) = F(g(x)) \succ_{rpo} F(x) = I(N(f(x)))$, and for the recursive call $f(g(x)) \succ_{spo}^Q x$ we apply twice case 1.

Example 5. In the following non-simply terminating example (from [AG97]) we use a precedence $\succeq_{\mathcal{F}}$ as second component in $\succeq_Q$.

$$
\begin{array}{ll}
le(0,y) & \to true \\
le(s(x),0) & \to false \\
le(s(x),s(y)) & \to le(x,y) \\
minus(0,y) & \to 0 \\
minus(s(x),y) & \to if(le(s(x),y),s(x),y) \\
if(true,s(x),y) & \to 0 \\
if(false,s(x),y) & \to s(minus(x,y)) \\
quot(0,s(y)) & \to 0 \\
quot(s(x),s(y)) & \to s(quot(minus(x,y),s(y)))
\end{array}
$$

We can take as $\succeq_Q$ the lexicographic combination $(\succeq_I, \succeq_T^{\mathcal{F}})_{lex}$ with the precedence $le \succ_{\mathcal{F}} true$, $le \succ_{\mathcal{F}} false$, $minus \succ_{\mathcal{F}} if$ and $if \succ_{\mathcal{F}} s$. For $\succeq_I$, the interpretation is given by $(le(x,y),b)$, $(true,b)$, $(false,b)$, $(if(x,y,z),y)$ and $(minus(x,y),x)$; and the basic quasi-ordering is RPO generated by the precedence $quot \succ_P s$, $s \succ_P b$ and $s \succ_P 0$.

We only show here that $minus(s(x),y) \succ_{mspo} if(le(s(x),y),s(x),y)$. First $minus(s(x),y) \succeq_I if(le(s(x),y),s(x),y)$ holds since $I(minus(s(x),y)) = s(x) = I(if(le(s(x),y),s(x),y))$. To prove $minus(s(x),y) \succ_{spo}^Q if(le(s(x),y),s(x),y)$, we have $minus(s(x),y) \succ_Q if(le(s(x),y),s(x),y)$, since $minus \succ_{\mathcal{F}} if$. Then by case 2 of SPO we only need to check the recursive call $minus(s(x),y) \succ_{spo}^Q le(s(x),y)$, $minus(s(x),y) \succ_{spo}^Q s(x)$ and $minus(s(x),y) \succ_{spo}^Q y$. The last two follow from case 1 of SPO. For the first one we have $minus(s(x),y) \succ_Q le(s(x),y)$, since $I(minus(s(x),y) = s(x) \succ_{rpo} b = I(le(s(x),y))$, and as before we can conclude by case 2.

Example 6. The following system is an automatic translation of a prolog program that computes the Ackermann function. This example comes from Claus Claves Master's thesis, in the context of the development of TALP [OCM00], an automated termination proof tool for logic programs based on rewriting techniques [1]. This TRS can be proved terminating by the dependency pair method (see section 5.2), but using a further refinement of the dependency graph introduced in [AG98] (see also [AG00]).

[1] This example was posed to us by Claude Marche.

As shown below, this example can be easily handled by MSPO with a very simple quasi-reduction pair.

$$
\begin{array}{ll}
ack\_in(0, n) & \to ack\_out(s(n)) \\
ack\_in(s(m), 0) & \to u11(ack\_in(m, s(0))) \\
u11(ack\_out(n)) & \to ack\_out(n) \\
ack\_in(s(m), s(n)) & \to u21(ack\_in(s(m), n), m) \\
u21(ack\_out(n), m) & \to u22(ack\_in(m, n)) \\
u22(ack\_out(n)) & \to ack\_out(n)
\end{array}
$$

We can take as $\succeq_Q$ the lexicographic combination $(\succeq_T^{\mathcal{F}}, \succeq_I^N)_{lex}$ with the precedence $ack\_in \succ_{\mathcal{F}} ack\_out$, $ack\_in \succ_{\mathcal{F}} u11$, $ack\_in \succ_{\mathcal{F}} s$, $ack\_in \succ_{\mathcal{F}} s$, $ack\_in \sim_{\mathcal{F}} u21$ and $u21 \succ_{\mathcal{F}} u22$; and with the renaming map N which is the identity except $N(ack\_in) = Ack\_in$ and $N(u21) = U21$. For $\succeq_I$, the interpretation is given by $(ack\_out(x), a)$, $(ack\_in(x, y), a)$, $(Ack\_in(x, y), acku(x))$, $(u21(x, y), a)$, $(U21(X, Y), acku(Y))$,$(u11(X), a)$, and $(u22(X), a)$ and the basic quasi-ordering is RPO generated by the precedence $s \succ_P a$ and $0 \succ_P a$.

5 Generalizing other termination proof methods

As an application of the provided methods to generate suitable quasi-orderings $\succeq_Q$, we will show how two known termination proof methods, namely *dummy elimination* [FZ95] and the *dependency pairs* [AG97], can be seen as a particular instance of the monotonic semantic path ordering. Note that as a side effect, this provides a new simple proof of their correctness. Finally we study Geser's proposal and show that it is strictly weaker than ours.

5.1 Dummy elimination

Dummy elimination consists of a transformation which eliminates function symbols from a signature replacing them by a constant ($\diamond$ in our notation); terms and rewrite rules are transformed accordingly. The soundness result states that a TRS R, defined over $\mathcal{T}(\mathcal{F} \cup \mathcal{F}_a, \mathcal{X})$, where $\mathcal{F}_a$ contains symbols of arity ≥ 1 which are to be eliminated, is terminating if the transformed TRS $E(R)$, defined over $\mathcal{T}(\mathcal{F} \cup \{\diamond\}, \mathcal{X})$, is terminating.

Let I_a be the interpretation defined by the pairs $(g(x_1, \ldots, x_n), \diamond_g)$ for every symbol g to be eliminated (and the identity pair for all other symbols). The system $E(R)$ is given by

$$E(R) = \{\mathtt{cap}(l) \to u \mid u \in \{\mathtt{cap}(r)\} \cup \mathtt{dec}(r),\ l \to r \in R\}$$

where $\mathtt{cap}(s) = I_a(s)$ and $\mathtt{dec}(s)$ contains $I_a(s')$ for all s' subterm of s just below a function symbol to be eliminated (for details see [FZ95]). For example, $R = \{f(f(x)) \to f(g(f(x)))\}$ is transformed, via the elimination of g, in the system $E(R) = \{f(f(x)) \to f(\diamond);\ f(f(x)) \to f(x)\}$, where $\diamond$ is the constant replacing g.

In the following we show that whenever termination of R can be shown using dummy elimination, we can find a simple quasi-reduction pair $\langle \succeq_I, \succeq_Q \rangle$ s.t. R is contained in MSPO.

Let $\succ$ be the reduction ordering containing $E(R)$, and $\succeq$ be $\succ \cup =$. Then we define $\succeq_I$ by means of the interpretation I_a defined above and $\succeq$ as basic quasi-ordering, i.e. $s \succeq_I t$ iff $I_a(s) \succeq I_a(t)$. On the other hand, we define $\succeq_Q$ as $s \succeq_Q t$ iff $I_a(s)(\succeq \cup \rhd)^* I_a(t)$. Note that if $\succ$ is a simplification ordering like RPO, the subterm relation is already included.

Now we show that $\langle \succeq_I, \succeq_Q \rangle$ is a quasi-reduction pair. Since $\succ$ is a reduction ordering, $\succeq$ is a quasi-reduction quasi-ordering, and, by proposition 2, $\succeq_I$ also is. As already said, since $\succ$ is a reduction ordering, we have that $(\succeq \cup \rhd)^*$ is a well-founded quasi-ordering stable under substitutions, and since $\succeq_I \subseteq \succeq_Q$, by propositions 3.1 and 4.1, we conclude that $\langle \succeq_I, \succeq_Q \rangle$ is a quasi-reduction pair.

Finally we show that if $E(R)$ is contained in $\succ$ then R is contained in $\succ_{mspo}$. Since $\mathtt{cap}(l) \to \mathtt{cap}(r) \in E(R)$ we have $I_a(l) = \mathtt{cap}(l) \succ \mathtt{cap}(r) = I_a(r)$ and hence $l \succeq_I r$. To show that $l \succ^Q_{spo} r$ we prove that $l \succ^Q_{spo} r'$ for all r' subterm of r, by induction on $|r'|$. We have that $u \unrhd \mathtt{cap}(r')$ for some $u \in \{\mathtt{cap}(r)\} \cup \mathtt{dec}(r)$. By definition, $\mathtt{cap}(l) \to u \in E(R)$, and hence $\mathtt{cap}(l) \succ u$. Consequently, since the strict part of $(\succeq \cup \rhd)^*$ includes $(\succ \cup \rhd)^+$, we have $I_a(l) = \mathtt{cap}(l)(\succ \cup \rhd)^+ \mathtt{cap}(r') = I_a(r')$ and thus, $l \succ_Q r'$. By induction hypothesis $l \succ^Q_{spo} r'_i$ for all arguments r'_i of r', and therefore, by case 2, we have $l \succ^Q_{spo} r'$.

Recently, the *argument filtering* transformation method [KNT99] has been proposed. The basic idea of this method is the same as in dummy elimination but using more general term transformations, like the AFS in the dependency pair method, or the ones presented here. In fact, they coincide with the interpretations we have given but requiring that f and f_I are the same symbol. Therefore, we can prove, in the same way as for dummy elimination, that this method is a particular instance of MSPO, since we can take $\succeq_I$ and $\succeq_Q$ as before, but with this new interpretations.

Finally, let us remark that, by using interpretation from terms to multisets of terms, we can also show that the *distribution elimination* method [Zan94] is a particular case of MSPO. In our case the restrictions imposed on this method to be correct are necessary to assure that $\succeq_I$ and $\succeq_Q$ are stable under substitutions.

5.2 Dependency Pairs

We consider here the plain dependency pair method, i.e. the method without using what is called the *dependency graph* refinement (see [AG97,AG00] for details). In section 6 some ideas about how this refinement can be incorporated to our method will be given.

In this method for a given TRS R the signature $\mathcal{F}$ is split into two sets: the constructor symbols set C and the defined symbols set D. Defined symbols are those heading the left hand side of a rule in R and constructor symbols are all others.

Let N be a renaming map in $\mathcal{F}$ defined as $N(f) = F$ for all symbols $f \in D$ and the identity for the others. The dependency pairs of a rule $l \to r$ in R

is the set of pairs $\langle N(l), N(r')\rangle$ for every subterm r' of r headed by a symbol in D. For example, the rule $f(f(x)) \rightarrow f(g(f(x)))$ has the dependency pairs $\langle F(f(x)), F(g(f(x)))\rangle$ and $\langle F(f(x)), F(x)\rangle$.

Then R is terminating if there is a quasi-reduction quasi-ordering $\succeq$ such that for every rule $l \rightarrow r$ we have $l \succeq r$ and for every dependency pair $\langle s, t\rangle$ of $l \rightarrow r$ we have $s \succ t$. Note that in practice these conditions are expressed as an ordering constraint, called in the rest of the paper the *dependency pair constraint*.

In the following we show that whenever termination of R can be shown using dependency pairs, we can a find a simple quasi-reduction pair $\langle \succeq_I, \succeq_Q\rangle$ obtained from the ordering used by the dependency method, s.t. R is contained in MSPO.

Let $\succeq_{\mathcal{F}}$ be a precedence on $\mathcal{F}$ where $f \succeq_{\mathcal{F}} g$ iff $f \in D$ (hence $f \succ_{\mathcal{F}} g$ iff $f \in D$ and $g \in C$). Then we take $\succeq_I$ as $\succeq$ and $\succeq_Q$ as $(\succeq_T^{\mathcal{F}}, \succeq_I^N)_{lex}$. By propositions 3 and 4 (2 and 3), we have that $\langle \succeq_I, \succeq_Q\rangle$ is a quasi-reduction pair.

Finally we show that R is contained in $\succ_{mspo}$. We have directly that $l \succeq_I r$ since $l \succeq r$ in the dependency pairs proof. To show that $l \succ_{spo}^Q r$ we prove that $l \succ_{spo}^Q r'$ for all r' subterm of r, by induction on $|r'|$. If $top(r') \in C$ then $top(l) \succ_{\mathcal{F}} top(r')$ and hence $l \succ_Q r'$. Otherwise $top(l) \succeq_{\mathcal{F}} top(r')$ and there is a dependency pair $\langle N(l), N(r')\rangle$, s.t. $N(l) \succ N(r')$, which implies $l \succ_I^N r'$, and hence $l \succ_Q r'$. Since, by induction hypothesis $l \succ_{spo}^Q r_i'$ for all arguments r_i' of r', by case 2 of SPO, we have $l \succ_{spo}^Q r'$.

Note that we have only used case 2 of SPO for proving that R is contained in $\succ_{mspo}$. Moreover, note that the precedence we have used to build $\succeq_Q$ is quite weak in its strict part. We believe that by using better precedences as a first component of $\succeq_Q$, e.g. adding strict comparisons between the defined symbols, we can capture easily part of the power of the dependency graph (see section 6).

5.3 Geser's monotonic semantic path ordering

Now we analyze Geser's proposal [Ges92] for a monotonic SPO. We give here the strict part of his definition.

Definition 6. *Let $\succeq_Q$ be a quasi-reduction quasi-ordering.*

$$s \succ_G t \quad \textit{iff} \quad s \succ_{spo}^Q t \textit{ and } f(\ldots, s, \ldots) \succeq_Q f(\ldots, t, \ldots) \textit{ for all } f \in \mathcal{F}$$

Although this version is an important step in the right direction, it has two main weaknesses. First of all, the requirement of $\succeq_Q$ to be a quasi-reduction quasi-ordering is too strong. As shown in example 2, this makes SPO loose completeness, and hence Geser's proposal cannot be complete. Furthermore, in a more practical view, with such a $\succeq_Q$ neither the dummy elimination technique nor the dependency pair technique can be included (note that, for instance, the renaming mapping does not preserve quasi-monotonicity). On the other hand, with respect to efficiency of an implementation of the method, a termination proof requires a huge number of comparisons with $\succeq_Q$, since for every rule $l \rightarrow r$, every symbol f and every argument position of this symbol $f(\ldots,l,\ldots) \succeq_Q f(\ldots,r,\ldots)$ has to be checked. Note that instead we only have to check $l \succeq_I r$.

Now we show that $\succ_G$ is included in $\succ_{mspo}$.

Lemma 2. *Let $\succeq_Q$ be a quasi-reduction quasi-ordering. There exists some $\succeq_I$, s.t. $\langle \succeq_I, \succeq_Q \rangle$ is a quasi-reduction pair and $\succ_G \subseteq \succ_{mspo}$.*

Proof. We take $\succeq_I$ as $s \succeq_I t$ iff $f(\ldots,s,\ldots) \succeq_Q f(\ldots,t,\ldots)$ for all $f \in \mathcal{F}$. It is obvious that in this case $\succ_G \subseteq \succ_{mspo}$. Then, we have to prove that $\langle \succeq_I, \succeq_Q \rangle$ is a quasi-reduction pair, that is, $\succeq_I$ is quasi-monotonic on $\succeq_Q$, which follows by definition of $\succeq_I$, and $\succeq_I$ is a quasi-monotonic quasi-ordering stable under substitutions, which follows directly from the properties of $\succeq_Q$. □

6 Constraints

Using MSPO, we can translate our termination problem into an *ordering constraint solving problem* (similar to the ones given in [Com90] for lexicographic path ordering, except that here variables are universally quantified), which is, in general, more suitable for automation. This translation is simply based on applying the definition of MSPO, and SPO, to the rules of the TRS.

Let R be a set of rules $\{l_i \rightarrow r_i \mid 1 \leq i \leq n\}$. We consider the following initial MSPO-constraint:

$$l_1 \succ_{mspo} r_1 \wedge \ldots \wedge l_n \succ_{mspo} r_n$$

Which is transformed by applying the definition of MSPO into the conjunction of two constraints I_C and SPO_C

$$\begin{array}{l} I_C : \quad\;\; l_1 \succeq_I r_1 \wedge \ldots \wedge l_n \succeq_I r_n \\ SPO_C : l_1 \succ^Q_{spo} r_1 \wedge \ldots \wedge l_n \succ^Q_{spo} r_n \end{array}$$

Now the definition of SPO is applied to the second part of the constraint. This is formalized by means of constraint transformation rules:

$$\begin{array}{ll} s \succeq^Q_{spo} t \Longrightarrow \top & \text{if } s \equiv t \\ s \succeq^Q_{spo} t \Longrightarrow s \succ^Q_{spo} t & \text{if } s \not\equiv t \\ x \succ^Q_{spo} t \Longrightarrow \bot & \\ s \succ^Q_{spo} x \Longrightarrow \top & \text{if } s \not\equiv x \in Vars(s) \\ s = f(s_1,\ldots,s_m) \succ^Q_{spo} g(t_1,\ldots,t_n) = t \Longrightarrow & \\ \quad s_1 \succeq^Q_{spo} t \vee \ldots \vee s_m \succeq^Q_{spo} t \vee & \\ \quad (s \succ_Q t \wedge s \succ^Q_{spo} t_1 \wedge \ldots \wedge s \succ^Q_{spo} t_n) \vee & \\ \quad (s \succeq_Q t \wedge \{s_1,\ldots,s_m\} \succ\!\!\succ^Q_{spo} \{t_1,\ldots,t_n\}) & \end{array}$$

Where $\{s_1,\ldots,s_m\} \succ\!\!\succ^Q_{spo} \{t_1,\ldots,t_n\}$ is translated into a constraint over $\succ^Q_{spo}$ and $\succeq^Q_{spo}$.

It is easy to see that these transformation rules are terminating and confluent. Moreover, the resulting normal form is an ordering constraint over $\succ_Q$ and $\succeq_Q$. Then after computing the disjunctive normal form, the initial constraint SPO_C

has been translated into a disjunction of constraints over $\succ_Q$ and $\succeq_Q$ each one of the form

$$Q_C: \quad s_1 \succ_Q t_1 \wedge \ldots \wedge s_p \succ_Q t_p \wedge s'_1 \succeq_Q t'_1 \wedge \ldots \wedge s'_q \succeq_Q t'_q$$

where none of the terms are variables.

Now we have to find a reduction pair $\langle \succeq_I, \succeq_Q \rangle$ satisfying I_C and one of these constraints Q_C. Note that, from what we have seen in section 5.2, the dependency pair constraint can be obtained by applying always case 2 of SPO, which means that some Q_C obtained above represents this path. Since there are several possible Q_C, it may happen, as in example 6, that the path chosen by the dependency pair method is not the easiest one.

To solve the constraints I_C and Q_C some simplification techniques are necessary. A first simple example of such a simplification, is obtained by considering that $\succeq_Q$ is a lexicographic combination $\langle \succeq_T^{\mathcal{F}}, \succeq_M \rangle$ with $\langle \succeq_I, \succeq_M \rangle$ being a quasi-reduction pair. Then we take the constraint Q_C, and define $top(s) \succeq_{\mathcal{F}} top(t)$ iff either $s \succeq_Q t$ or $s \succ_Q t$ is in Q_C. Now, if $\succ_{\mathcal{F}}$ is the strict part of $\succeq_{\mathcal{F}}$ and $\sim_{\mathcal{F}}$ its equivalence, we can simplify the constraint Q_C into M_C by the following rules:

$$\begin{array}{ll}
s \succ_Q t \Longrightarrow \top & \text{if } top(s) \succ_{\mathcal{F}} top(t) \\
s \succeq_Q t \Longrightarrow \top & \text{if } top(s) \succ_{\mathcal{F}} top(t) \\
s \succ_Q t \Longrightarrow s \succ_M t & \text{if } top(s) \sim_{\mathcal{F}} top(t) \\
s \succeq_Q t \Longrightarrow s \succeq_M t & \text{if } top(s) \sim_{\mathcal{F}} top(t)
\end{array}$$

At this point, in general, it is interesting to define $\succeq_M$ by means of a renaming N of all symbols heading s or t in some $s \succ_M t$ or $s \succeq_M t$ in M_C, which leads to a final renamed constraint called N_C.

Solving the constraints I_C and N_C, i.e., finding an adequate quasi-reduction pair, is very similar to solve the dependency pair constraints. The simplest solution is to consider the quasi-reduction pair $\langle \succeq_I, \succeq_I \rangle$, but we believe that, if necessary, we will be able to use refinements like the dependency graph in our constraint solver.

7 Modularity

In this section we present some modularity results for MSPO which are obtained applying known abstract sufficient conditions ensuring modularity of termination. We consider disjoint unions of TRSs, i.e., systems that do not share any symbol, and constructor sharing unions of TRSs, i.e., systems which share only constructors.

A TRS R is called terminating under non-deterministic collapses, denoted $\mathcal{C}_\varepsilon$-terminating, if $R \cup \{G(x, y) \rightarrow x, G(x, y) \rightarrow y\}$ terminates for some new symbol G. For $\mathcal{C}_\varepsilon$-termination we have the following results:

- [Ohl94] $\mathcal{C}_\varepsilon$-termination is a modular property for disjoint unions of TRSs.
- [Gra94] $\mathcal{C}_\varepsilon$-termination is a modular property for constructor-sharing unions of finite TRSs.

Lemma 3. *If* $\trianglerighteq \subseteq \succeq_B$ *and* R *is included in* $\succ_{mspo}$ *then* R *is* $\mathcal{C}_\varepsilon$*-terminating.*

Note that since $\succeq_B$ includes the subterm relation and G is a new symbol, we have $I(G(x,y)) = G(x,y)$ and hence $G(x,y) \succeq_I x$ and $G(x,y) \succeq_I y$. Therefore, since by case 1 both rules are included in SPO, we can conclude that $G(x,y) \succ_{mspo} x$ and $G(x,y) \succ_{mspo} y$.

Corollary 1. *Let* $\succ^1_{mspo}$ *and* $\succ^2_{mspo}$ *be two MSPO whose basic orderings include the subterm relation and let* R_1 *and* R_2 *be TRSs that are included in* $\succ^1_{mspo}$ *and* $\succ^2_{mspo}$ *respectively.*

- *If* R_1 *and* R_2 *are disjoint then* $R_1 \cup R_2$ *is* $\mathcal{C}_\varepsilon$*-terminating, and thus terminating.*
- *If* R_1 *and* R_2 *share only constructors and are finite then* $R_1 \cup R_2$ *is* $\mathcal{C}_\varepsilon$*-terminating, and thus terminating.*

Similarly in [GO98] modularity results for disjoint and constructor sharing unions of TRSs proved terminating using the dependency pair method are presented. For disjoint systems the results are the same, but for constructor sharing unions the restriction is not imposed on the finiteness of the systems but on the treatment of the shared symbols in the termination proof.

8 Conclusion

In this paper we have described a new ordering-based general method for proving termination of TRSs. MSPO is based on the well-known SPO, but unlike SPO it is monotonic, which makes it useful in practice. It is a complete method, i.e., it characterizes termination. The method generalizes, in a simple way, many known methods based on transformations. In the case of the dependency pairs method, which is by now one of the most successful general methods applied in practice, we have only shown that we generalize it without the "dependency graph" refinement. These kind of "operational" refinements do not fit so well in our framework, but, by considering the termination of TRSs as an ordering constraint solving problem over MSPO, we believe that, if necessary, we will be able to use these refinements in our constraint solver. On the other hand, due to its additional flexibility, we have seen that MSPO provides, in some cases, simpler termination proofs than the dependency pair method. Thus, in order to study the behavior of MSPO, we are developing a termination system based on this method. Currently this system can only check termination once the ingredients for the MSPO are provided, but our aim is to fully automate the termination proofs, by using some heuristics for selecting ingredients to be tried.

Besides its application to first-order term rewriting, the fact that the method is defined by means of orderings, for which the properties to be fulfilled are well-known, opens the door to other important classes like AC-rewriting (i.e., rewriting modulo associativity and commutativity axioms) or higher-order rewriting, for which the lack of general methods is more important. The idea is to combine with MSPO some recent results obtained for RPO in the AC-case [Rub99] and in the HO-case [JR99], since all them share the same structure and are based on orderings.

Finally, apart from the presented modularity results, we are studying other kinds of combinations. In particular, we are interested in the so called *hierarchical combinations* (see [AG98] related results for the dependency pair method), since reusing termination proofs or proving the termination of the TRS by splitting it in different parts may be crucial in practice.

Acknowledgments: We want to thank Pilar Nivela, Roberto Nieuwenhuis, Enno Ohlebusch and Claude Marche for their help in the development of this work and to Thomas Arts, Jürguen Giesl and Aart Middeldorp for their comments on an earlier version of this work.

References

[AG97] T. Arts and J. Giesl. Automatically proving termination where simplification orderings fail. In *TAPSOFT: 7th International Joint Conference on Theory and Practice of Software Development*, LNCS 1214, pages 261–272. Springer-Verlag, 1997.

[AG98] T. Arts and J. Giesl. Modularity of termination using dependency pairs. In T. Nopkow, editor, *9th International Conference on Rewriting Techniques and Applications (RTA)*, LNCS 1379, pages 226–240, Tsukuba, Japan, 1998. Springer-Verlag.

[AG00] T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.

[BD86] Leo Bachmair and Nachum Dershowitz. Commutation, transformation, and termination. In Jörg H. Siekmann, editor, *8th International Conference on Automated Deduction (CADE)*, LNCS 230, pages 5–20, Oxford, England, 1986. Springer-Verlag.

[BL90] Françoise Bellegarde and Pierre Lescanne. Termination by completion. *Applicable Algebra in Engineering, Communication and Computing*, 1:79–96, 1990.

[BN98] F. Baader and T. Nipkow. *Term Riwriting and all that.* Cambridge University Press, 1998.

[Com90] Hubert Comon. Solving symbolic ordering constraints. *International Journal of Foundations of Computer Science*, 1(4):387–411, 1990.

[Der82] Nachum Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17(3):279–301, 1982.

[Der87] Nachum Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3:69–116, 1987.

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 6, pages 244–320. Elsevier Science Publishers B.V., 1990.

[FZ95] Maria Ferreira and Hans Zantema. Dummy elimination: making termination easier. In H. Reichel, editor, *10th Int. Conf. on Fundamentals of Computation Theory*, LNCS 965, pages 243–252. Springer-Verlag, 1995.

[Ges92] Alfons Geser. On a monotonic semantic path ordering. Technical Report 92-13, Ulmer Informatik-Berichte, Universität Ulm, Ulm, Germany, 1992.

[GO98] J. Giesl and E. Ohlebusch. Pushing the frontiers of combining rewrite systems farther outwards. In *Proceedings of the Second International Workshop on Frontiers of Combining Systems (FroCoS '98)*, Logic and Computation Series, pages 141–160, Amsterdam, The Netherlands, 1998. Research Studies Press, John Wiley & Sons.

[Gra94] B. Gramlich. Generalized sufficient conditions for modular termination of rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 5:131–158, 1994.

[JR99] Jean-Pierre Jouannaud and Albert Rubio. The higher-order recursive path ordering. In *14th IEEE Symposium on Logic in Computer Science (LICS)*, pages 402–411, Trento, Italy, 1999.

[KL80] S. Kamin and J.-J. Levy. Two generalizations of the recursive path ordering. Unpublished note, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, 1980.

[Klo92] J.W. Klop. Term rewriting systems. In S. Abramsky, D.M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 1–116. Oxford University Press, 1992.

[KNT99] K. Kusakari, M. Nakamura, and Y. Toyama. Argument filtering transformation. In *Proceedings of the International Conference on Principles and Practice of Declarative Programming (PPDP'99)*, LNCS 1702, pages 47–61. Springer-Verlag, 1999.

[Kru60] Joseph B. Kruskal. Well-quasi-ordering, the Tree Theorem, and Vazsonyi's conjecture. *Transactions of the American Mathematical Society*, 95:210–225, May 1960.

[MHH96] A. Middeldorp, Ohsaki H., and Zantema H. Transforming termination by self-labelling. In *Proceedings of the 13th International Conference on Automated Deduction*, volume 1104 of *LNAI*, pages 373–386, New Brunswick, 1996. Springer-Verlag.

[OCM00] E. Ohlebusch, C. Claves, and C. Marché. Talp: a tool for the termination analysis of logic programs. In *Proceedings of the 11th conference on Rewriting Techniques and Applications*, LNCS. Springer-Verlag, 2000.

[Ohl94] E. Ohlebusch. On modularity of termination of term rewriting systems. *Theoretical Computer Science*, 136(2):333–360, 1994.

[Rub99] A. Rubio. A fully syntactic AC-RPO. In P. Narendran and M. Rusinowitch, editors, *Tenth International Conference on Rewriting Techniques and Applications (RTA)*, LNCS 1631, pages 133–147, Trento, Italy, 1999. Springer-Verlag.

[Ste95] J. Steinbach. Automatic termination proofs with transformation orderings. In J. Hsiang, editor, *6th International Conference on Rewriting Techniques and Applications (RTA)*, LNCS 914, pages 11–25, Kaiserslautern, Germany, 1995.

[Xi98] H. Xi. Towards automated termination proofs through freezing. In *9th Int. Conf. on Rewriting Techniques and Applications (RTA)*, LNCS 1379, pages 271–285, Tsukuba, Japan, 1998.

[Zan94] H. Zantema. Termination of term rewriting: interpretation and type elimination. *Journal of Symbolic Computation*, 17:23–50, 1994.