Orderings and Constraints: Theory and Practice of proving termination

Cristina Borralleras[1] and Albert Rubio[2]*

[1] Universitat de Vic, Spain
Email: cristina.borralleras@uvic.es
[2] Universitat Politècnica de Catalunya, Barcelona, SPAIN
Email: rubio@lsi.upc.es

Abstract. In contrast to the current general way of developing tools for proving termination automatically, this paper intends to show an alternative program based on using on the one hand the theory of term orderings to develop powerful and widely applicable methods and on the other hand constraint based techniques to put them in practice.
In order to show that this program is realizable a constraint-based framework is presented where ordering based methods for term rewriting, including extensions like Associative-Commutative rewriting, Context-Sensitive rewriting or Higher-Order rewriting, as well as the use of rewriting strategies, can be put in practice in a natural way.

1 Introduction

In this paper we show how to translate into a constraint solving problem any termination proof using the *Monotonic Semantic Path Ordering* (MSPO) [BFR00] and its variants for Associative-Commutative (AC) rewriting [BR03], Higher-Order (HO) rewriting [BR01] and Context-Sensitive (CS) rewriting [Bor03]. By using the definition of MSPO a disjunction of constraints is obtained, such that, if any of these constraints can be solved, then the TRS is proved to be terminating.

Our constraints have the same semantics as the ones obtained in the dependency pair method (DP) [AG00], and, in particular, one of the constraints obtained from the definition of the MSPO coincides with the one given by DP method (and it is unclear whether this one is always the best to be solved). Moreover, since both kind of constraints share the same semantics, we can reuse all techniques developed to solve DP constraints like the DP graph or many other further developments [AG00,GTSK04,HM05,GTSKF06].

The framework we propose was first described in [Bor03]. A similar framework for the DP-method was independently proposed in [GTSK04]. These results show that MSPO can be seen as an ordering-based way to understand the DP-method, and that the key point for the success of this method is the use, in

* Supported by spanish project LOGICTOOLS, ref. TIN2004-07925.

practice, of ordering constraints, for which a wide variety of sound solvers have been developed.

Additionally, we study the application of our techniques to prove termination of innermost rewriting. In order to reuse our framework, the TRS is modified by adding constraints to the rules, which approximate the restrictions imposed by the strategy. A constrained rule can be applied if the substitution satisfies the constraint. Hence, a TRS is innermost terminating if its constrained version is terminating.

A constrained TRS is terminating if all instances of each constrained rule (i.e. the instances satisfying the constraint) are included in a reduction ordering. Therefore, we can apply MSPO but taking the constraints of the rules into account. This is done by inheriting the constraints when applying MSPO. As a result, we obtain a disjunction of constrained constraints, which, to avoid confusion, will be called *decorated constraints* (note that the constraints coming from the rules are applied to the ordering constraints coming from MSPO). Using these decorated constraints we can cover all techniques applied in the DP method for innermost rewriting. Furthermore, these decorated constraints can be used to store other information which can be relevant for the termination proof. The same ideas applied to the innermost strategy can be applied, as well, to other strategies that can be approximated by means of constrained rules, like, for instance, rewriting with priorities [vdP98].

Finally, we show, as an example, how our framework also extends to the higher-order version of the MSPO, which can also be done for the AC-version and the CS-version.

These results should be seen as a proof of the thesis that developing results at the ordering level and implementing and applying them at the constraint level is an appropriate program to obtain a general purpose tool for proving termination.

Our method has been implemented in a system called *Termptation* which automatically proves termination of rewriting and innermost rewriting. The implementation does not cover any of the extensions of MSPO to associativity-commutativity, higher-order or context-sensitive rewriting, which is planed for future development.

Basic notions and definitions are given in section 2. In sections 3 and 4 we revise the dependency pair method and the monotonic semantic path ordering respectively. Section 5 is devoted to present and apply our constraint framework. In section 6 we adapt our framework to deal with innermost rewriting and in section 7 we consider higher-order rewriting. Some conclusions are given in section 8. An extended version of these results and all proofs can be found in [Bor03].

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 $f(\ldots s \ldots) \succ f(\ldots t \ldots)$, 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 $t_1 \succ t_2 \succ \ldots$

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 $t = s[r\sigma]_p$. The defined symbols D are the root symbols of left-hand sides of rules. All other function symbols are called constructors.

A TRS R is terminating if there exists no infinite sequence $t_1 \to_R t_2 \to_R \cdots$ Thus, the transitive closure $\xrightarrow{+}_R$ of any terminating TRS is a reduction ordering. Furthermore, reduction orderings characterize termination of TRSs, i.e. 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 $l \to r \in R$.

Given a relation $\succ$, the multiset extension of $\succ$ on finite multisets, denoted by $\succ\!\succ$, is defined as the smallest transitive relation containing $X \cup \{s\} \succ\!\succ X \cup \{t_1, \ldots, t_n\}$ if $s \succ t_i$ 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 $\succ$. $\succeq$ is *monotonic* if $f(\ldots, s, \ldots) \succeq f(\ldots, t, \ldots)$ whenever $s \succeq t$. An ordering $\succ_2$ is compatible with a quasi-ordering $\succeq_1$ if $\succeq_1 \cdot \succ_2 \subseteq \succ_2$.

Let $\succeq_1$ be a quasi-ordering and let $\succ_2$ be an ordering. Then $\langle \succeq_1, \succ_2 \rangle$ is a *compatible ordering pair* if $\succ_2$ and $\succeq_1$ are stable under substitutions, $\succ_2$ is compatible with $\succeq_1$ and $\succ_2$ is well-founded.

A *precedence* $\succeq_{\mathcal{F}}$ is a well-founded quasi-ordering on $\mathcal{F}$. It is extended to a quasi-ordering on terms as $s \succeq_{\mathcal{F}} t$ iff $top(s) \succeq_{\mathcal{F}} top(t)$.

3 The Dependency Pair Method

For every defined symbol $f \in D$, we introduce a fresh tuple symbol $f^\#$ (sometimes written as F for simplicity) with the same arity. If $t = f(\ \bar{t}\ )$ then $t^\#$ denotes the term $f^\#(\ \bar{t}\ )$. If $l \to r \in R$ and t is a subterm of r with a defined root symbol, then $\langle l^\#, t^\# \rangle$ is a dependency pair of R. The set of all dependency pairs of R is denoted $DP(R)$. An R-chain is a sequence $\langle s_1, t_1 \rangle, \langle s_2, t_2 \rangle, \ldots$ of pairs in $DP(R)$ such that there is a substitution σ where $t_i\sigma \to_R^* s_{i+1}\sigma$.

Theorem 1 (Termination Criterion [AG00]). *A TRS R is terminating if and only if no infinite R-chain exists.*

Definition 1. *A pair $\langle \succeq_I, \succ_q \rangle$ is called a* DP-reduction pair *if $\succeq_I$ is monotonic, and $\langle \succeq_I, \succ_q \rangle$ is a compatible ordering pair.*

Then a TRS R is terminating if there is a DP-reduction pair $\langle \succeq_I, \succ_q \rangle$ such that $l \succeq_I r$ for every rule in R and $s \succ_q t$ for every dependency pair $\langle s, t \rangle$ in $DP(R)$.

Many refinements and improvements for solving the obtained constraints have been described. See [GTSK04] for a general framework for solving DP-constraints.

4 The Monotonic Semantic Path Order

Let us now recall the definition of the *semantic path ordering* (SPO) [KL80], with a slight modification, since we use a compatible ordering pair instead of a quasi-ordering as underlying (or base) ordering:

Definition 2. *Given a* compatible ordering pair $\langle \succeq_Q, \succ_q \rangle$*, the SPO, denoted as* $\succ_{spo}$*, is defined as* $s = f(s_1, \ldots, s_m) \succ_{spo} t$ *iff*

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

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

The semantic path ordering is well-defined, but, in general, it is not monotonic, even when $\succeq_Q$ is monotonic (in fact, the same problem appears if $\succ_Q$ is monotonic).

Definition 3. *We say that* $\succeq_I$ *is* monotonic on $\succeq_Q$ *(or* $\succeq_Q$ *is* monotonic wrt. $\succeq_I$*) if* $s \succeq_I t$ *implies* $f(\ldots s \ldots) \succeq_Q f(\ldots t \ldots)$ *for all terms* s *and* t *and function symbols* f*.*

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

A triplet $\langle \succeq_I, \succeq_Q, \succ_q \rangle$ *is called a* reduction triplet *if* $\succeq_I$ *is monotonic,* $\succeq_I$ *is stable under substitutions,* $\langle \succeq_Q, \succ_q \rangle$ *is a compatible ordering pair and* $\succeq_I$ *is monotonic on* $\succeq_Q$*.*

Note that in particular, if $\langle \succeq_I, \succeq_Q \rangle$ is a reduction pair then $\langle \succeq_I, \succeq_Q, \succ_Q \rangle$ is a reduction triplet.

Now we define the *monotonic semantic path ordering* (MSPO) [BFR00]:

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

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

Theorem 2. $\succ_{mspo}$ *is a reduction ordering. Furthermore, MSPO characterizes termination.*

5 Reduction constraints

In this section we present the constraint framework where our termination problems are translated to. We will first present the syntax and semantics of our constraints. Then, we show how the termination problems are translated into constraint problem through the definition of the MSPO. Then, we present some transformation techniques in order to solve the obtained constraints and show how the DP method is included in ours. Finally, we show how all other transformation techniques applied in the DP framework apply to ours.

5.1 Syntax and Semantics

Definition 5. *A* reduction constraint *is a pair $\langle C_1, C_2\rangle$, where C_1 is a conjunction of positive literals over the relation $\sqsupseteq_1$ and C_2 is conjunction of positive literals over $\sqsupset_2$ and $\sqsupseteq_2$.*

Now we provide the notion of satisfiability for reduction constraints, which is based on reduction triplets. Hence, it is easy to show that these kind of constraints are the ones obtained by applying MSPO.

Definition 6. *A reduction constraint $\langle C_1, C_2\rangle$ is satisfiable iff there exists a reduction triplet $\langle \geq_1, \geq_2, >_2\rangle$ such that $\geq_1$ satisfies C_1 and $\langle \geq_2, >_2\rangle$ satisfies C_2, i.e. $\geq_2$ satisfies all literals $s \sqsupseteq_2 t$ in C_2 and $>_2$ satisfies all literals $s \sqsupset_2 t$ in C_2.*

The following definition and theorem allow us to connect the reduction triplet semantics given above for reduction constraints with the R-chain semantics given for the constraints obtained by the dependency pair method.

In what follows, we will speak about the relation $\sqsupseteq_1$ in C_1, or simply $\sqsupseteq_1$, as the relation defined by all instances of all $s \sqsupseteq_1 t$ in C_1. We have not used a new relation symbol to ease the reading. The same will be done for the relation $\sqsupseteq_2$ in C_2 (or simply $\sqsupseteq_2$) and the relation $\sqsupset_2$ in C_2 (or simply $\sqsupset_2$).

Then, we use $\xrightarrow{\not\Lambda\ *}_{\sqsupseteq_1}$ for the reflexive and transitive closure of the monotonic with non-empty contexts (and stable under substitution) closure of $\sqsupseteq_1$ in C_1.

Definition 7. *Let $\langle C_1, C_2\rangle$ be a reduction constraint. A pair $\langle s, t\rangle_O$, with O being either $\sqsupseteq_2$ or $\sqsupset_2$, is said to be in C_2 iff $s\ O\ t$ occur in C_2 (up to renaming of variables). It is said to be strict if O is $\sqsupset_2$.*

A sequence of pairs of terms $\langle s_1, t_1\rangle_{O_1}, \langle s_2, t_2\rangle_{O_2}, \ldots$ is a chain *in $\langle C_1, C_2\rangle$ if every $\langle s_i, t_i\rangle_{O_i}$ is in C_2 and there exists a substitution σ such that $t_i\sigma \xrightarrow{\not\Lambda\ *}_{\sqsupseteq_1} s_{i+1}\sigma$ holds for all consecutive pairs $\langle s_i, t_i\rangle_{O_i}$ and $\langle s_{i+1}, t_{i+1}\rangle_{O_{i+1}}$ in the sequence.*

Lemma 1. *If $C_1 = \{l \sqsupseteq_1 r \mid \forall l \to r \in R\}$ the notion of R-chain using dependency pairs coincides with the notion of chain using our pairs $\langle s, t\rangle_O$.*

Theorem 3. *A reduction constraint $\langle C_1, C_2\rangle$ is satisfiable iff there is no chain in $\langle C_1, C_2\rangle$ with infinitely many strict pairs.*

From this theorem it follows that the constraints obtained by the dependency pair method are reduction constraints, as we will show in detail in section 5.4.

5.2 Translating MSPO-termination of rewriting into reduction constraint solving

Using MSPO, we can translate our termination problem into a reduction constraint solving problem. 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 $\{l_i \to r_i \mid 1 \leq i \leq n\}$. 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 a conjunction of two constraints, C_I and C_{SPO}

$$\begin{array}{ll} C_I : & l_1 \succeq_I r_1 \wedge \ldots \wedge l_n \succeq_I r_n \\ C_{SPO} : & l_1 \succ_{spo} r_1 \wedge \ldots \wedge l_n \succ_{spo} r_n \end{array}$$

Now the definition of SPO given in Section 4 is applied to the second part of the constraint. This is formalized by means of correct constraint transformation rules:

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

where $\{s_1, \ldots, s_m\} \succ\!\succ_{spo} \{t_1, \ldots, t_n\}$ is translated into a constraint over $\succ_{spo}$ and $\succeq_{spo}$.

It is easy to see that these transformation rules are correct, terminating and confluent. Moreover, the resulting normal form is an ordering constraint over $\succ_q$ and $\succeq_Q$ which represents the conditions on $\succ_q$ and $\succeq_Q$ that are necessary to show that C_{SPO} is true. Then after computing the disjunctive normal form, the initial constraint C_{SPO} has been translated into a disjunction of constraints over $\succ_q$ and $\succeq_Q$ each one of the form

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

where none of the terms are variables.

Now we have to find a reduction triplet $\langle \succeq_I, \succeq_Q, \succ_q \rangle$ satisfying C_I and one of these constraints C_Q, which means that this is a reduction constraint satisfaction problem as the following theorem states.

Theorem 4. *Let R be a TRS and let C_I be the constraint over $\succeq_I$ and let $C_Q^1 \vee ... \vee C_Q^k$ be the disjunction of constraints over $\succ_q$ and $\succeq_Q$ obtained by applying the MSPO method. Then R is terminating iff some reduction constraint $\langle C_I, C_Q^i \rangle$ is satisfiable.*

From the previous theorem, in order to prove termination, we have to show that some reduction constraint $\langle C_I, C_Q^i \rangle$ is satisfiable. To this end, we have to provide some correct (wrt. satisfiability) constraint transformation techniques which allow us to simplify the constraints until they can directly be shown to be satisfiable by building an actual reduction triplet (or pair).

From now on, we will assume that we have a reduction constraint $\langle C_1, C_2 \rangle$, which is transformed step by step, preserving satisfiability, into one or several simpler reduction constraints of the form $\langle S_1, S_2 \rangle$. After this simplification process, each resulting reduction constraint $\langle S_1, S_2 \rangle$ is proved satisfiable separately by building an appropriate reduction quasi-ordering $\succeq$ (or a compatible ordering pair $\langle \succeq, \succ \rangle$ where $\succeq$ is monotonic), which includes all literals in S_1 and S_2. Note that, if $\succeq$ is a reduction quasi-ordering and $\succ$ is the strict part of it, then $\langle \succeq, \succeq, \succ \rangle$ is a reduction triplet.

5.3 Constraint transformations

In this section we propose some basic techniques for simplifying the reduction constraint $\langle C_1, C_2 \rangle$.

A first simple example of such a simplification, is obtained by using a well-founded precedence on the set of symbols. Thus every literal $s \sqsupset_2 t$ or $s \sqsupseteq_2 t$ can be removed if the top symbol of s is strictly greater than the top symbol of t in the precedence. Moreover, the remaining literals $s \sqsupset_2 t$ or $s \sqsupseteq_2 t$ in C_2 have to fulfil that the top symbol of s is greater than or equal to the top symbol of t in the precedence.

Being precise, if $\succeq_{\mathcal{F}}$ is a precedence, such that $top(s) \succeq_{\mathcal{F}} top(t)$ for every $s \sqsupset_2 t$ or $s \sqsupseteq_2 t$ in C_2, then we can simplify the constraint C_2 by using the following rules:

Precedence simplification rules

$$s \sqsupset_2 t \Longrightarrow \top \text{ if } top(s) \succ_{\mathcal{F}} top(t)$$
$$s \sqsupseteq_2 t \Longrightarrow \top \text{ if } top(s) \succ_{\mathcal{F}} top(t)$$

By building appropriate reduction triplets, we can easily show the correctness of this transformation.

Transformation 1 *Let $\langle C_1, C_2 \rangle$ be a reduction constraint and let $\langle C_1, C_2' \rangle$ be a reduction constraint obtained by applying the precedence simplification rules wrt. some precedence $\succeq_{\mathcal{F}}$ such that $top(s) \succeq_{\mathcal{F}} top(t)$ for every $s \sqsupset_2 t$ or $s \sqsupseteq_2 t$ in C_2. Then $\langle C_1, C_2 \rangle$ is satisfiable if and only if $\langle C_1, C_2' \rangle$ is satisfiable.*

Moreover, if we choose adequately the precedence we can apply an optimal simplification with respect to this precedence-based transformation.

Now we present a transformation technique, based on renamings of top function symbols (as already done in the dependency pair method). This transformation is not a simplification in the sense that no literal is removed, but it allows us to apply, in the future, a different treatment to the symbols when they occur on top of a term in C_2 (which will ease the proof of satisfiability of the final reduction constraint obtained after all the simplification process).

Transformation 2 *(Renaming) Let $\langle C_1, C_2\rangle$ be a reduction constraint and let C_2' the result of renaming all function symbols f heading a term in C_2 by a new symbol $f^{\#}$. Then $\langle C_1, C_2\rangle$ is satisfiable iff $\langle C_1, C_2'\rangle$ is satisfiable.*

5.4 Reduction constraints and the DP method

In this section we show that the constraints obtained in our method have the same semantics as the ones produced by the dependency pair method. In particular, we show that one of the constraints obtained in the disjunction when using the MSPO constraint method coincides, after applying the precedence and the renaming transformations, with the one given by the DP method.

The following results states that the constraint obtained by the dependency pair method is a reduction constraint.

Theorem 5. *Let R be a TRS and let C_1 be the constraint containing $l_i \sqsupseteq_1 r_i$ for every rule $l_i \to r_i$ in R and let C_2 be the constraint containing $s \sqsupset_2 t$ for every dependency pair $\langle s, t\rangle$ in $DP(R)$. Then R is terminating iff the reduction constraint $\langle C_1, C_2\rangle$ is satisfiable.*

Now, using always case 2 of the definition of SPO, except when the term on the right is a variable (where we apply case 1), and applying precedence transformation (with defined symbols greater than constructors) and finally the renaming transformation we get the constraint given by the DP method. Moreover, it is not difficult to prove that there is a reduction triplet $\langle \succeq_I, \succeq_Q, \succ_q\rangle$ if and only if there is a DP-reduction pair $\langle \succeq_I, \succ_q\rangle$ solving the resulting reduction constraint.

Theorem 6. *The dependency pair method is included in the MSPO constraint method.*

The following example shows a case in which the constraint obtained by DP-method is, in principle, not the easiest one to be solved among all constraints generated by MSPO.

Example 1. The following system is an automatic translation of a prolog program that computes the Ackermann function.

$$\begin{aligned}
ack\_in(0, x) &\to ack\_out(s(x))\\
ack\_in(s(y), 0) &\to u11(ack\_in(y, s(0)))\\
u11(ack\_out(x)) &\to ack\_out(x)\\
ack\_in(s(y), s(x)) &\to u21(ack\_in(s(y), x), y)\\
u21(ack\_out(x), y) &\to u22(ack\_in(y, x))\\
u22(ack\_out(x)) &\to ack\_out(x)
\end{aligned}$$

This system has been proved included in the MSPO with a constraint which for the rule

$$u21(ack\_out(x), y) \rightarrow u22(ack\_in(y, x))$$

contains the following literals which correspond to the application of case 2 first and then case 3 of the SPO:

$$\begin{array}{l} u21(ack\_out(x), y) \succ_q u22(ack\_in(y, x)) \\ u21(ack\_out(x), y) \succeq_Q ack\_in(y, x) \end{array}$$

Note that with the DP-method both, at least initially, would be strict.

Similarly, the constraint framework defined in [GTSK04] is basically the same as the one described in this paper, except on the fact that, since we extract our constraints from the definition of MSPO, our constraint C_2 may include non-strict inequalities from the beginning.

Due to this equivalence between both frameworks all sound transformation techniques described in [GTSK04] (called there *processors*), including , for instance, the DP graph or many other ideas developed in [AG00,HM05,GTSKF06] can be used in our framework and vice versa.

6 Innermost rewriting: constrained rules

In innermost rewriting, a subterm is a redex only if all arguments are in normal form. Therefore, we can impose this condition on the left-hand sides of the rules. In this section we show a way to keep this condition aside of the rules and then use this information to prove innermost termination with the reduction constraint framework.

Definition 8. *Let R be a TRS. Then $s \xrightarrow{i}_{\mathcal{R}} t$ is an innermost rewriting step if $s = u[l\sigma]$, $t = u[r\sigma]$, $l \rightarrow r \in R$, and all the proper subterms of $l\sigma$ are in normal form, that is, $l\sigma$ is irreducible in non-top positions.*

As a first consequence of the above definition, when using innermost rewriting, all the rules in a TRS which has a proper subterm $l|_p$ such that for any normal substitution σ, $l|_p\sigma$ is a redex, can be eliminated.

Note that we can still have rules which have a proper subterm $l|_p$ such that $l|_p\sigma$ is a redex for some normal substitution σ, but not in general.

To prove innermost termination of a TRS, we have to show that any innermost reduction is finite, which can be proved by showing that any innermost reduction is included in a well-founded ordering.

Theorem 7. *A rewrite system $\mathcal{R}$ over a set of terms $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is innermost terminating if and only if there is a well-founded, monotonic ordering over $\mathcal{T}(\mathcal{F}, \mathcal{X})$ such that for every $l \rightarrow r \in R$ and for all substitutions σ such that $l\sigma$ is irreducible in non-top positions, $l\sigma \succ r\sigma$.*

Note that if a term t is irreducible then any subterm of t is also irreducible. Thus, the condition "$l\sigma$ is irreducible in non-top positions" in the above theorem can also be written as "$l_i\sigma$ is irreducible for all l_i argument of l".

This condition of irreducibility can be expressed by adding a constraint to the rules. For instance, given the rules

$$\begin{array}{r}f(x, g(x)) \to f(1, g(x)) \\ g(1) \to g(0)\end{array}$$

the innermost condition can be expressed by the constrained rules

$$\begin{array}{rl}f(x, g(x)) \to f(1, g(x)) & | \ \{irred(x), irred(g(x))\} \\ g(1) \to g(0) & | \ \{irred(1)\}\end{array}$$

Let us formalize the notion of constrained rule and its use in innermost rewriting.

Definition 9. *Given a rule $l \to r$ and a conjunction K of literals build on a given set of predicates, we say that $l \to r \mid K$ is a constrained rule.*

A rewrite step using a constrained rule $l \to r \mid K$ is a rewrite step using $l \to r$ which is only applicable for those substitutions σ satisfying K, i.e., $s \to t$ using the constrained rule $l \to r \mid K$ iff $s = u[l\sigma] \to u[r\sigma] = t$ for some context u and substitution σ solution of K.

Now we show that by using constrained rewriting with *irreducibility constraints* we can characterize the innermost rewriting strategy.

Definition 10. *Let R be a set of rules. The set R^i of constrained rules for innermost rewriting contains for each rule $l \to r \in R$ a constrained rule $l \to r \mid K^i(l)$ where $K^i(f(t_1, \ldots, t_n)) = \{irred_R(t_i) \mid \forall i \in \{1, \ldots, n\}\}$ and $irred_R(t)$ means that t is irreducible with respect to R.*

Irreducibility constraints are used in the CARIBOO system [FGK02], by means of *abstraction constraints.* However, our aim when using these constraints is to be able to apply the same constraint solving techniques as in the innermost version of the DP method, like, for instance, when building the approximated *innermost DP graph.*

Lemma 2. *Let R be a set of rules. Then, $s \xrightarrow{i} t$ using $l \to r \in R$ if and only if $s \to t$ using the constrained rewrite rule $l \to r \mid K^i(l) \in R^i$*

This notion of constrained rewriting can be generalized other relations like orderings.

Definition 11. *Given two terms s and t and a set of literals K, $s \succeq t \mid K$ denotes that $s\sigma \succeq t\sigma$ for all solutions σ of K; and $s \succ t \mid K$ denotes that $s\sigma \succ t\sigma$ for all solutions σ of K.*

The following theorem for innermost termination follows from Theorem 7 using Lemma 2 and Definition 11.

Theorem 8. *Let $\succ$ be a reduction ordering. A TRS $\mathcal{R}$ is innermost terminating iff $l \succ r \mid K^i(l)$ holds for every rule $l \to r \in R$.*

Corollary 1. *A TRS $\mathcal{R}$ is innermost terminating iff there exists a reduction triplet $\langle \succeq_I, \succeq_Q, \succ_q \rangle$ s.t. $l \succ_{mspo} r \mid K^i(l)$, for every $l \to r \in R$.*

In order to be able to use constrained rules in our method, first we have to adapt the notion of reduction constraints.

6.1 Decorated reduction constraints

We will now increase the expressive power of our reduction constraints $\langle C_1, C_2 \rangle$ by attaching conditions to the pairs in C_1 and C_2. These conditions will contain all or part of the information coming from the constraints of the rules.

Definition 12. *A decorated literal (or pair) $l \mid D$ is a literal (pair) l with an attached condition D.*

A decorated reduction constraint *is a tuple $\langle C_1, C_2 \rangle$, where C_1 is a conjunction of decorated positive literals over the relation $\sqsupseteq_1$ and C_2 is a conjunction of decorated positive literals over $\sqsupset_2$ and $\sqsupseteq_2$.*

Definition 13. *A decorated reduction constraint $\langle C_1, C_2 \rangle$ is satisfiable iff there exists a reduction triplet $\langle \geq_1, \geq_2, >_2 \rangle$ such that $\geq_1$ satisfies C_1 and $\langle \geq_2, >_2 \rangle$ satisfies C_2, i.e., $\geq_2$ satisfies all decorated literals $s \sqsupseteq_2 t \mid D$ in C_2 and $>_2$ satisfies all decorated literals $s \sqsupset_2 t \mid D$ in C_2.*

From this, all definitions and results given for reduction constraints can be extended in a natural way to decorated reduction constraint (see [Bor03] for details).

6.2 Proving termination of constrained rules by MSPO using decorated reduction constraints

We can translate the termination problem of a set of constrained rules into an *ordering constraint solving problem* using decorated reduction constraints. This translation is simply based on applying the definition of MSPO, and SPO, to a set of constrained rules.

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

$$l_1 \succ_{mspo} r_1 \mid K_1 \wedge \ldots \wedge l_n \succ_{mspo} r_n \mid K_n$$

This decorated ordering constraint is transformed by applying the definition of MSPO into the conjunction of two decorated ordering constraints, C_I and C_{SPO}

$$\begin{array}{ll} C_I : & l_1 \succeq_I r_1 \mid K_1 \wedge \ldots \wedge l_n \succeq_I r_n \mid K_n \\ C_{SPO} : & l_1 \succ_{spo} r_1 \mid K_1 \wedge \ldots \wedge l_n \succ_{spo} r_n \mid K_n \end{array}$$

Now the definition of SPO given in Section 4 is applied to the second part of the constraint. This is formalized by means of correct constraint transformation rules:

$$
\begin{array}{ll}
s \succeq_{spo} t \mid K \Longrightarrow \bot & \text{if } K \text{ is false} \\
s \succeq_{spo} t \mid K \Longrightarrow \top & \text{if } s \equiv t \\
s \succeq_{spo} t \mid K \Longrightarrow s \succ_{spo} t \mid K & \text{if } s \not\equiv t \\
s \succ_{spo} t \mid K \Longrightarrow \bot & \text{if } K \text{ is false} \\
x \succ_{spo} t \mid K \Longrightarrow \bot & \\
s \succ_{spo} x \mid K \Longrightarrow \top & \text{if } s \not\equiv x \in Vars(s) \\
s = f(s_1, \ldots, s_m) \succ_{spo} g(t_1, \ldots, t_n) = t \mid K \Longrightarrow & \\
\quad s_1 \succeq_{spo} t \mid K \vee \ldots \vee s_m \succeq_{spo} t \mid K \vee & \\
\quad (s \succ_q t \mid K \wedge s \succ_{spo} t_1 \mid K \wedge \ldots \wedge s \succ_{spo} t_n \mid K) \vee & \\
\quad (s \succeq_Q t \mid K \wedge \{s_1, \ldots, s_m\} \succ\!\!\succ_{spo} \{t_1, \ldots, t_n\} \mid K) &
\end{array}
$$

where $\{s_1, \ldots, s_m\} \succ\!\!\succ_{spo} \{t_1, \ldots, t_n\} \mid K$ is translated into a decorated ordering constraint over $\succ_{spo}$ and $\succeq_{spo}$.

As for the case of (non-constrained) rewriting, it is easy to see that these transformation rules are correct, terminating and confluent. Moreover, the resulting normal form is a decorated ordering constraint over $\succ_q$ and $\succeq_Q$. Then after computing the disjunctive normal form, the initial constraint C_{SPO} has been translated into a disjunction of constraints over $\succ_q$ and $\succeq_Q$ each one of the form

$$C_Q : \quad s_1 \succ_q t_1 \mid K'_1 \wedge \ldots \wedge s_p \succ_q t_p \mid K'_p \wedge s'_1 \succeq_Q t'_1 \mid K'_{p+1} \wedge \ldots \wedge s'_q \succeq_Q t'_q \mid K'_q$$

where none of the terms are variables.

Note that all K'_j coincides with some of the K_i coming from the rules R (i.e., no new constraints are generated).

Now, as for the non-decorated case, we have to find a reduction triplet satisfying C_I and one of these decorated constraints C_Q.

Theorem 9. *Let R be a set of constrained rules and let $\mathcal{C}_I$ be the constraint containing $l_i \succeq_I r_i \mid K_i$ for every constrained rule $l_i \to r_i \mid K_i \in R$, and let $\mathcal{C}^1_Q \cup \ldots \cup \mathcal{C}^k_Q$ be the disjunction of decorated ordering constraints over $\succ_q$ and $\succeq_Q$ obtained by applying the MSPO constraint method for constrained termination. Then the set of constrained rules R is terminating iff some decorated reduction constraint $\langle \mathcal{C}_I, \mathcal{C}^i_Q \rangle$ is satisfiable.*

In particular, for innermost termination, we have the following corollary.

Corollary 2. *Let R be a TRS and let R^i be the set of constrained rules $l \succ r \mid K^i(l)$ for every $l \to r \in R$. Then $\mathcal{R}$ is innermost terminating iff $\langle \mathcal{C}_I, \mathcal{C}^i_Q \rangle$ is satisfiable for some decorated reduction constraint $\langle \mathcal{C}_I, \mathcal{C}^i_Q \rangle$ obtained by applying the MSPO method to R^i.*

Now, as seen for (non-constrained) rewriting, in order to prove termination we have to show that some decorated reduction constraint $\langle \mathcal{C}_I, \mathcal{C}^i_Q \rangle$ is satisfiable. All transformations known for non-decorated constraints can also be used for transforming decorated reduction constraints, although, in some cases, the transformation can be improved by using the additional information of the decorated pairs (see [Bor03] for details).

7 Higher-Order rewriting

In this section we extend the results on first-order rewriting to the higher-order case. The aim of this section is to show the generality of our approach and how simple is to extend it to other kind of rewriting. Due to the lack of room we will present a very simple form of the monotonic higher-order semantic path ordering. Moreover, we will consider a very simple type system, which is enough to present our ideas. See [JR07,BR01] for more powerful versions.

7.1 Types, Signatures and Terms

We consider terms of a simply typed lambda-calculus generated by a signature of higher-order function symbols.

The set of types $\mathbb{T}$ is generated from the set $V_{\mathbb{T}}$ of *type variables* (considered as sorts) by the constructor $\rightarrow$ for *functional types* in the usual way. As usual, $\rightarrow$ associates to the right. In the following, we use α, β for type variables and $\sigma, \tau, \rho, \theta$ for arbitrary types.

A signature $\mathcal{F}$ is a set of function symbols which are meant to be algebraic operators, equipped with a fixed number n of arguments (called the *arity*) of respective types $\sigma_1 \in \mathbb{T}, \ldots, \sigma_n \in \mathbb{T}$, and an output type $\sigma \in \mathbb{T}$. A *type declaration* for a function symbol f will be written as $f : \sigma_1 \times \ldots \times \sigma_n \rightarrow \sigma$. Type declarations are not types, although they are used for typing purposes.

The set $\mathcal{T}(\mathcal{F}, \mathcal{X})$ of *raw algebraic λ-terms* is generated from the signature $\mathcal{F}$ and a denumerable set $\mathcal{X}$ of variables according to the grammar rules

$$\mathcal{T} := \mathcal{X} \mid (\lambda \mathcal{X} : \mathbb{T}.\mathcal{T}) \mid @(\mathcal{T}, \mathcal{T}) \mid \mathcal{F}(\mathcal{T}, \ldots, \mathcal{T}).$$

Terms of the form $\lambda x : \sigma.u$ are called *abstractions*, while the other terms are said to be *neutral*. For sake of brevity, we will often omit types. $@(u, v)$ denotes the application of u to v. The application operator is allowed to have a variable arity. As a matter of convenience, we may write $@(u, v_1, \ldots, v_n)$ for $@(@(\ldots @(u, v_1) \ldots), v_n)$, assuming $n \geq 1$.

7.2 Typing Rules

Typing rules restrict the set of terms by constraining them to follow a precise discipline. Environments are sets of pairs written $x : \sigma$, where x is a variable and σ is a type. Our typing judgments are written as $\Gamma \vdash M : \sigma$ if the term M can be proved to have the type σ in the environment Γ with the following type system

$$\textbf{Variables:} \quad \frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma} \qquad \textbf{Functions:} \quad \frac{\begin{array}{c} f : \sigma_1 \times \ldots \times \sigma_n \rightarrow \sigma \in \mathcal{F} \\ \Gamma \vdash t_1 : \sigma_1 \; \ldots \; \Gamma \vdash t_n : \sigma_n \end{array}}{\Gamma \vdash f(t_1, \ldots, t_n) : \sigma}$$

Abstraction:
$$\frac{\Gamma \cup \{x:\sigma\} \vdash t:\tau}{\Gamma \vdash (\lambda x:\sigma.t):\sigma \to \tau}$$

Application:
$$\frac{\Gamma \vdash s:\sigma \to \tau \quad \Gamma \vdash t:\sigma}{\Gamma \vdash @(s,t):\tau}$$

7.3 Higher-order rewriting and termination

The rewrite relation considered in this paper is the union of the one induced by a set of higher-order rewrite rules and the β-reduction rule $@(\lambda x.v, u) \longrightarrow_\beta v\{x \mapsto u\}$, both working modulo α-conversion. For simplicity reasons, in this work we do not consider η-reduction, although all results can be extended to include it.

A higher-order *term rewrite system* is a set of rewrite rules $R = \{\Gamma \vdash l_i \to r_i\}_i$, where l_i and r_i are higher-order terms such that l_i and r_i have the same type σ_i in the environment Γ.

Given a term rewriting system R, a term s rewrites to a term t at position p with the rule $\Gamma \vdash l \to r$ and the substitution γ, written $s \xrightarrow[l\to r]{p} t$, or simply $s \to_R t$, if $s|_p = l\gamma$ and $t = s[r\gamma]_p$ (modulo α-conversion).

Higher-order reduction orderings are basically reduction orderings (monotonic, stable under substitutions and well-founded) operating on typed higher-order terms and including β-reduction. A higher-order rewrite system R is terminating if there is a higher-order reduction ordering $\succ$ such that $l \succ r$ for all rules $l \to r$ in R.

7.4 The Higher-Order Semantic Path Ordering (HOSPO)

From now on, to ease the reading, we will omit the environments in judgments.

Definition 14. *Given a compatible ordering pair on higher-order typed terms $\langle \succeq_Q, \succ_q \rangle$ (where none of $\succeq_Q$ and $\succ_q$ needs to include β-reduction), the HOSPO, denoted as $\succ_{hspo}$, is defined as* $s:\tau \succ_{hspo} t:\tau$ *iff*

1. *$f \in \mathcal{F}$, $s = f(s_1,\ldots,s_m)$ and $s_i \succeq_{hspo} t$, for some $i = 1,\ldots,m$, or*
2. *$f,g \in \mathcal{F}$,$s = f(s_1,\ldots,s_m) \succ_q t = g(t_1,\ldots,t_n)$ and for all $i = 1,\ldots,n$, either $s \succ_{hspo} t_i$ or $s_j \succeq_{hspo} t$ for some $j = 1,\ldots,m$, or*
3. *$f,g \in \mathcal{F}$, $s = f(s_1,\ldots,s_m) \succeq_Q t = g(t_1,\ldots,t_n)$ and $\{s_1,\ldots,s_m\} \succ\!\succ_{hspo} \{t_1,\ldots,t_n\}$, or*
4. *$f \in \mathcal{F}$, $s = f(s_1,\ldots,s_m)$, $t = @(t_1,\ldots,t_n)$, and for all $i = 1,\ldots,n$, either $s \succ_{hspo} t_i$ or $s_j \succeq_{hspo} t$ for some $j = 1,\ldots,m$, or*
5. *$s = @(s_1,s_2)$, $t = @(t_1,t_2)$, $\{s_1,s_2\} \succ\!\succ_{hspo} \{t_1,t_2\}$, or*
6. *$s = \lambda x.u$, $t = \lambda x.v$, $u \succ_{hspo} v$, or*
7. *$s = @(\lambda x.u, v)$ and $u\{x \mapsto v\} \succeq_{hspo} t$*

where $\succeq_{hspo}$ is defined as $\succ_{hspo} \cup$ α-conversion.

Definition 15. *Let $\langle \succeq_I, \succeq_Q, \succ_q \rangle$ be a reduction triplet on higher-order typed terms. The monotonic higher-order semantic path ordering (MHOSPO), denoted by $\succ_{mhspo}$, is defined as*

$$s \succ_{mhspo} t \quad \textit{if and only if} \quad s \succeq_I t \quad \textit{and} \quad s \succ_{hspo} t$$

Theorem 10. *$\succ_{mhspo}$ is a higher-order reduction ordering.*

Finally we show how to translate a MHOSPO termination proof into a (higher-order) reduction constraint problem $\langle C_1, C_2\rangle$. The constraint C_1 is obtained as before from $\succeq_I$. To obtain C_2 we use the definition of HOSPO.

This is formalized by means of correct constraint transformation rules like (we only outline the ones that are different from those in 5.2):

$$\begin{array}{l}
s = f(s_1,\dots,s_m) \succ_{hspo} g(t_1,\dots,t_n) = t \Longrightarrow \\
\quad \bigvee_{s_i:\mathcal{T}} s_i \succeq_{hspo} t \vee \\
\quad (s \succ_q t \bigwedge_{t_i:\mathcal{T}'} (s \succ_{hspo} t_i \vee \bigvee_{s_j:\mathcal{T}} s_j \succeq_{hspo} t_i)) \vee \\
\quad (s \succeq_Q t \wedge \{s_1,\dots,s_m\} \succ\!\!\succ_{hspo} \{t_1,\dots,t_n\}) \\
s = f(s_1,\dots,s_m) \succ_{hspo} @(t_1,\dots,t_n) = t \Longrightarrow \\
\quad \bigwedge_{t_i:\mathcal{T}'} (s \succ_{hspo} t_i \vee \bigvee_{s_j:\mathcal{T}} s_j \succeq_{hspo} t_i) \\
\vdots
\end{array}$$

But, note that, due to the type conditions, the transformation rule

$$s \succ_{hspo} x \Longrightarrow \top \text{ if } s \not\equiv x \in Vars(s)$$

does not hold in general, and hence there might be terms on the right hand side of literals in C_2 being a variable.

Example 2. Filter. To make it simpler we only consider a type variable α. Let $V_{\mathbb{T}} = \{\alpha\}$, $\mathcal{X} = \{\ x, xs : \alpha,\ P : \alpha \to \alpha\ \}$ and $\mathcal{F} = \{[] : \alpha,\ cons : \alpha \times \alpha \to \alpha,\ True, False : \alpha,\ filter : (\alpha \to \alpha) \times \alpha \to \alpha,$ $iffil : \alpha \times (\alpha \to \alpha) \times \alpha \times \alpha \to \alpha\}$

$$\begin{array}{l}
filter(P, []) \to [] \\
filter(P, cons(x, xs)) \to iffil(@(P, x), P, x, xs) \\
iffil(True, P, x, xs) \to cons(x, filter(P, xs)) \\
iffil(False, P, x, xs) \to filter(P, xs)
\end{array}$$

The constraint C_1 is obtained as before:
$filter(P, []) \succeq_I []\ \wedge\ filter(P, cons(x, xs)) \succeq_I iffil(@(P, x), P, x, xs)\ \wedge$
$iffil(True, P, x, xs) \succeq_I cons(x, filter(P, xs))\ \wedge$
$iffil(False, P, x, xs) \succeq_I filter(P, xs)$
and as one of the constraints for C_2 we have:
$filter(P, cons(x, xs)) \succ_q iffil(@(P, x), P, x, xs)\ \wedge$
$iffil(True, P, x, xs) \succ_q cons(x, filter(P, xs))\ \wedge$
$iffil(True, P, x, xs) \succeq_Q filter(P, xs)\ \wedge\ iffil(False, P, x, xs) \succeq_Q filter(P, xs)$

Now using the precedence transformation and then the renaming transformation we obtain
$Filter(P, cons(x, xs)) \succ_q Iffil(@(P, x), P, x, xs) \wedge$
$Iffil(True, P, x, xs) \succeq_Q Filter(P, xs) \wedge Iffil(False, P, x, xs) \succeq_Q Filter(P, xs)$

Finally, the reduction constraint is solved by taking $\succeq_I = \succeq_Q$ and $\succ_q$ as the strict part of $\succeq_Q$, which is defined by a simple polynomial interpretation like
$|nil| = 0; |cons(x_1, x_2)| = x_2 + 1; |filter(x_1, x_2, x_3)| = x_3;$
$|iffil(x_1, x_2, x_3, x_4)| = x_4 + 1; |Filter(x_1, x_2, x_3)| = x_3; |Iffil(x_1, x_2, x_3, x_4)| = x_4.$

8 Conclusions and future work

Termptation (available at `http://www.lsi.upc.es/~albert`) is a fully automated system for proving termination of first-order term rewrite systems which follows the termination proof techniques described in this paper.

The current implementation of the system has to be improved in several ways. On the one hand, by incorporating the state of the art techniques for solving reduction constraints. On the other hand, by extending the system to handle other reduction strategies, as well as AC-rewriting, CS-rewriting and HO-rewriting.

We are especially interested in the HO-case, since we consider that it is a difficult case were our working program of developing theory at the level of orderings and practice at the level of constraints may be the right one to build a powerful termination tool.

References

[AG00] T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1-2):133–178, 2000.

[BFR00] C. Borralleras, M. Ferreira, and A. Rubio. Complete monotonic semantic path orderings. In *Proc. 17th Int. Conf. on Automated Deduction*, *LNAI* 1831, pages 346–364. Springer, 2000.

[Bor03] C. Borralleras. *Ordering-based methods for proving termination automatically.* PhD thesis, Dpto. LSI, Universitat Politècnica de Catalunya, 2003.

[BR01] C. Borralleras and A. Rubio. A monotonic higher-order semantic path ordering. In *Proc. 8th Int. Conf. on Logic for Programming, Artificial Intelligence and Reasoning*, *LNAI* 2250, pages 531–547. Springer, 2001.

[BR03] C. Borralleras and A. Rubio. Monotonic AC-compatible semantic path orderings. In *Proc. 14th Int. Conf. on Rewriting Techniques and Applications*, *LNCS* 2706, pages 279–295. Springer, 2003.

[FGK02] O. Fissore, I. Gnaedig, and H. Kirchner. CARIBOO : An Induction Based Proof Tool for Termination with Strategies. In *Proc. 4th Int. Conf. on Principles and Practice of Declarative Programming*, pages 62–73. ACM Press, 2002.

[GTSK04] J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In *Proc. 8th Int. Conf. on Logic for Programming, Artificial Intelligence and Reasoning*, *LNAI* 3452, pages 301–331. Springer, 2004.

[GTSKF06] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *J. Autom. Reasoning*, 37(3):155–203, 2006.

[HM05] N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *Information and Computation*, 199(1-2):172–199, 2005.

[JR07] J.-P. Jouannaud and A. Rubio. Polymorphic higher-order recursive path orderings. *Journal of the ACM*, 54(1), 2007.

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

[vdP98] J. van de Pol. Operational semantics of rewriting with priorities. *Theoretical Computer Science*, 200(1-2):289–312, 1998.