Proving Termination Using Recursive Path Orders and SAT Solving[⋆]

Peter Schneider-Kamp[1], René Thiemann[1], Elena Annov[2], Michael Codish[2], and Jürgen Giesl[1]

[1] LuFG Informatik 2, RWTH Aachen, Germany
{psk,thiemann,giesl}@informatik.rwth-aachen.de
[2] Department of Computer Science, Ben-Gurion University, Israel
{annov,mcodish}@cs.bgu.ac.il

Abstract. We introduce a propositional encoding of the recursive path order with status (RPO). RPO is a combination of a multiset path order and a lexicographic path order which considers permutations of the arguments in the lexicographic comparison. Our encoding allows us to apply SAT solvers in order to determine whether a given term rewrite system is RPO-terminating. Furthermore, to apply RPO within the dependency pair framework, we combined our novel encoding for RPO with an existing encoding for argument filters. We implemented our contributions in the termination prover AProVE. Our experiments show that due to our encoding, combining termination provers with SAT solvers improves the performance of RPO-implementations by orders of magnitude.

1 Introduction

Since the past year, several papers have illustrated the huge potential in applying SAT solvers for various types of termination problems for term rewrite systems (TRSs). The key idea is classic: the specific termination problem for a TRS $\mathcal{R}$ is encoded to a propositional formula φ which is satisfiable if and only if $\mathcal{R}$ has the desired termination property. Satisfiability of φ is tested using a state-of-the-art SAT solver and the termination proof for $\mathcal{R}$ is reconstructed from a satisfying assignment of φ. However, in order to obtain significant speedups, it is crucial to base the approach on polynomial encodings which are also small in practice.

The first such attempt addresses LPO-termination [16]. This work is based on BDDs and does not yield competitive results. A significant improvement is described in [3] and further extended in [4,23] for argument filters as used in the popular dependency pair framework [1,10] for termination of TRSs. Both [3] and [4,23] describe extremely fast SAT-based implementations. Successful SAT encodings of other termination techniques are presented in [8,9,14,24]. A common theme in all of these works is to represent (finite domain) integer variables as binary numbers in bit representation and to encode arithmetic constraints as Boolean functions on these representations.

[⋆] Supported by the Deutsche Forschungsgemeinschaft DFG under grant GI 274/5-1.

This paper introduces the first SAT-based encoding for recursive path orders. The main new and interesting contributions are (1) the encoding for the lexicographic comparison w.r.t. permutations, (2) the encoding for the multiset extension of the base order, and (3) the combination of the encoding for precedences and argument filters in order to use RPO with dependency pairs.

Our encoding of RPO is implemented in the termination prover AProVE [11]. The combination of a termination prover with a SAT solver yields a surprisingly fast implementation of RPO. All 865 TRSs in the *Termination Problem Data Base (TPDB)* [21] are analyzed in about 100 seconds for the case of strict precedences. Allowing non-strict precedences takes about 3 times longer. Moreover, power increases considerably compared to the implementation of LPO described in [4]: 27 additional termination proofs are obtained. The TPDB is the collection of examples used in the annual *International Termination Competition* [19].

After the necessary preliminaries on RPO in Sect. 2, Sect. 3 shows how to encode both multiset comparisons and lexicographic comparisons w.r.t. permutations, and how to combine them into a single class of orders. Sect. 4 combines these encodings with the concept of argument filters. In Sect. 5 we describe the implementation of our results and provide extensive experimental evidence indicating speedups in orders of magnitude. We conclude in Sect. 6.

2 Preliminaries

The classical approach to prove termination of a TRS $\mathcal{R}$ is to find a *reduction order* $\succ$ which orients all rules $\ell \to r$ in $\mathcal{R}$ (i.e., $\ell \succ r$). A reduction order is an order which is well founded, monotonic, and stable (closed under contexts and substitutions). In practice, most reduction orders amenable to automation are *simplification orders* [6]. We refer to [2] for further details on term rewriting.

Three of the most prominent simplification orders are the lexicographic path order (LPO) [15], the multiset path order (MPO) [6], and the recursive path order (RPO) [18] which combines the lexicographic and multiset path order allowing also permutations in the lexicographic comparison. This section introduces their definitions using a formulation that is suitable for the subsequent SAT encoding.

We assume an algebra of terms constructed over sets of function symbols $\mathcal{F}$ and variables $\mathcal{V}$. For a quasi-order $\succsim$ (i.e., a transitive and reflexive relation), we define $s \succ t$ iff $s \succsim t$ and $t \not\succsim s$ and we define $s \sim t$ iff both $s \succsim t$ and $t \succsim s$. Path orders are defined in terms of lexicographic and multiset extensions of a base order (on terms). We often denote tuples of terms as $\bar{s} = \langle s_1, \dots s_n \rangle$, etc.

Definition 1 (lexicographic extension). *Let $\succsim$ be a quasi-order. The lexicographic extensions of $\succ$, $\sim$, and $\succsim$ are defined on sequences of terms:*

- $\langle s_1, \dots, s_n \rangle \sim^{lex} \langle t_1, \dots, t_m \rangle$ *if and only if* $n = m$ *and* $s_i \sim t_i$ *for all* $1 \le i \le n$
- $\langle s_1, \dots, s_n \rangle \succ^{lex} \langle t_1, \dots, t_m \rangle$ *if and only if (a)* $m = 0$ *and* $n > 0$*; or (b)* $s_1 \succ t_1$*; or (c)* $s_1 \sim t_1$ *and* $\langle s_2, \dots, s_n \rangle \succ^{lex} \langle t_2, \dots, t_m \rangle$.
- $\succsim^{lex} \; = \; \sim^{lex} \cup \succ^{lex}$

So for tuples of numbers $\bar{s} = \langle 3, 3, 4, 0\rangle$ and $\bar{t} = \langle 3, 2, 5, 6\rangle$, we have $\bar{s} >^{lex} \bar{t}$ as $s_1 = t_1$ and $s_2 > t_2$ (where $>$ is the usual order on numbers).

The multiset extension of an order $\succ$ is defined as follows: $\bar{s} \succ^{mul} \bar{t}$ holds if $\bar{t}$ is obtained by replacing at least one element of $\bar{s}$ by a finite number of (strictly) smaller elements. However, the order of the elements in $\bar{s}$ and $\bar{t}$ is irrelevant. For example, let $\bar{s} = \langle 3, 3, 4, 0\rangle$ and $\bar{t} = \langle 4, 3, 2, 1, 1\rangle$. We have $\bar{s} >^{mul} \bar{t}$ because $s_1 = 3$ is replaced by the smaller elements $t_3 = 2$, $t_4 = 1$, $t_5 = 1$ and $s_4 = 0$ is replaced by zero smaller elements. So each element in $\bar{t}$ is "covered" by some element in $\bar{s}$. Such a cover is either by a larger s_i (then s_i may cover several t_j) or by an equal s_i (then one s_i covers one t_j). In this paper we formalize the multiset extension by a *multiset cover* which is a pair of mappings (γ, ε). Intuitively, γ expresses which elements of $\bar{s}$ cover which elements in $\bar{t}$ and ε expresses for which s_i this cover is by means of equal terms and for which by means of greater terms. This formalization facilitates encodings to propositional logic.

So in the example above, we have $\gamma(1) = 3$, $\gamma(2) = 2$ (since t_1 is covered by s_3 and t_2 is covered by s_2), and $\gamma(3) = \gamma(4) = \gamma(5) = 1$ (since t_3, t_4, and t_5 are all covered by s_1). Moreover, $\varepsilon(2) = \varepsilon(3) = \mathit{true}$ (since s_2 and s_3 are replaced by equal components), whereas $\varepsilon(1) = \varepsilon(4) = \mathit{false}$ (since s_1 and s_4 are replaced by (possibly zero) smaller components). Of course, in general multiset covers are not unique. For example, t_2 could also be covered by s_1 instead of s_2.

Definition 2 (multiset cover). *Let $\bar{s} = \langle s_1, \dots s_n\rangle$ and $\bar{t} = \langle t_1, \dots t_m\rangle$ be tuples of terms. A multiset cover (γ, ε) is a pair of mappings $\gamma : \{1, \dots, m\} \to \{1, \dots, n\}$ and $\varepsilon : \{1, \dots, n\} \to \{\mathit{false}, \mathit{true}\}$ such that for each $1 \leq i \leq n$, if $\varepsilon(i)$ (indicating equality) then $\{j \mid \gamma(j) = i\}$ is a singleton set.*

For $\bar{s} = \langle s_1, \dots s_n\rangle$ and $\bar{t} = \langle t_1, \dots t_m\rangle$ we define that $\bar{s} \succsim^{mul} \bar{t}$ if there exists a multiset cover (γ, ε) such that $\gamma(j) = i$ implies that either: $\varepsilon(i) = \mathit{true}$ and $s_i \sim t_j$, or $\varepsilon(i) = \mathit{false}$ and $s_i \succ t_j$.

Definition 3 (multiset extension). *Let $\succsim$ be a quasi-order on terms. The multiset extensions of $\succsim$, $\succ$, and $\sim$ are defined on tuples of terms:*

- *$\langle s_1, \dots, s_n\rangle \succsim^{mul} \langle t_1, \dots, t_m\rangle$ if and only if there exists a multiset cover (γ, ε) such that for all i, j, $\gamma(j) = i \;\Rightarrow\;$ (`if` $\varepsilon(i)$ `then` $s_i \sim t_j$ `else` $s_i \succ t_j$).*
- *$\langle s_1, \dots, s_n\rangle \succ^{mul} \langle t_1, \dots, t_m\rangle$ if and only if $\langle s_1, \dots, s_n\rangle \succsim^{mul} \langle t_1, \dots, t_m\rangle$ and for some i, $\neg\varepsilon(i)$, i.e., some s_i is not used for equality but rather replaced by zero or more smaller arguments t_j.*
- *$\langle s_1, \dots, s_n\rangle \sim^{mul} \langle t_1, \dots, t_m\rangle$ if and only if $\langle s_1, \dots, s_n\rangle \succsim^{mul} \langle t_1, \dots, t_m\rangle$, $n = m$, and for all i, $\varepsilon(i)$, i.e., all s_i are used to cover some t_j by equality.*

Let $\geq_{\mathcal{F}}$ denote a quasi-order (a so-called *precedence*) on the set of function symbols $\mathcal{F}$ and let $>_{\mathcal{F}} = (\geq_{\mathcal{F}} \setminus \leq_{\mathcal{F}})$ and $\approx_{\mathcal{F}} = (\geq_{\mathcal{F}} \cap \leq_{\mathcal{F}})$. Then $\geq_{\mathcal{F}}$ induces corresponding lexicographic and multiset path orders on terms.

Definition 4 (lexicographic and multiset path orders). *For a precedence $\geq_{\mathcal{F}}$ and $\rho \in \{\mathit{lpo}, \mathit{mpo}\}$ we define the relations $\succ_\rho$ and $\sim_\rho$ on terms. We use the notation $\bar{s} = \langle s_1, \dots s_n\rangle$ and $\bar{t} = \langle t_1, \dots t_m\rangle$.*

- *$s \succ_\rho t$ iff $s = f(\bar{s})$ and one of the following holds:*
  (1) $s_i \succ_\rho t$ or $s_i \sim_\rho t$ for some $1 \leq i \leq n$; or
  (2) $t = g(\bar{t})$ and $s \succ_\rho t_j$ for all $1 \leq j \leq m$ and either:
  (i) $f >_\mathcal{F} g$ or (ii) $f \approx_\mathcal{F} g$ and $\bar{s} \succ_\rho^{ext} \bar{t}$;
- *$s \sim_\rho t$ iff (a) $s = t$; or (b) $s = f(\bar{s})$, $t = g(\bar{t})$, $f \approx_\mathcal{F} g$, and $\bar{s} \sim_\rho^{ext} \bar{t}$;*

where $\succ_\rho^{ext}$ and $\sim_\rho^{ext}$ are the lexicographic or multiset extensions of $\succ_\rho$ and $\sim_\rho$ for the respective cases when $\rho = lpo$ and $\rho = mpo$.

Example 1. Consider the following three TRSs for adding numbers:

$$\begin{array}{ll}
\text{(a)} & \left\{ \ \mathsf{add}(\mathsf{0}, y) \to y \ , \quad \mathsf{add}(\mathsf{s}(x), y) \to \mathsf{add}(x, \mathsf{s}(y)) \ \right\} \\
\text{(b)} & \left\{ \ \mathsf{add}(x, \mathsf{0}) \to x \ , \quad \mathsf{add}(x, \mathsf{s}(y)) \to \mathsf{s}(\mathsf{add}(y, x)) \ \right\} \\
\text{(c)} & \left\{ \ \mathsf{add}(x, \mathsf{0}) \to x \ , \quad \mathsf{add}(x, \mathsf{s}(y)) \to \mathsf{add}(\mathsf{s}(x), y) \ \right\}
\end{array}$$

Example (a) is LPO-terminating for the precedence $\mathsf{add} >_\mathcal{F} \mathsf{s}$, but not MPO-terminating for any precedence. Example (b) is MPO-terminating for $\mathsf{add} >_\mathcal{F} \mathsf{s}$, but not LPO-terminating for any precedence as the second rule swaps x and y. Example (c) is neither LPO- nor MPO-terminating. However, termination could be proved using a path order where lexicographic comparison proceeds from right to left instead of left to right. The following definitions extend this observation to arbitrary permutations of the order in which we compare arguments.

As remarked before, the RPO combines such an extension of the LPO with MPO. This combination is facilitated by a *status function* which indicates for each function symbol if its arguments are to be compared based on a multiset extension or based on a lexicographic extension using some permutation μ.

Definition 5 (status function). *A status function σ maps each symbol $f \in \mathcal{F}$ of arity n either to the symbol $\mathtt{mul}$ or to a permutation μ_f on $\{1, \ldots, n\}$.*

Definition 6 (recursive path order with status). *For a precedence $\geq_\mathcal{F}$ and status function σ we define the relations $\succ_{rpo}$ and $\sim_{rpo}$ on terms. We use the notation $\bar{s} = \langle s_1, \ldots s_n \rangle$ and $\bar{t} = \langle t_1, \ldots t_m \rangle$.*

- *$s \succ_{rpo} t$ iff $s = f(\bar{s})$ and one of the following holds:*
  (1) $s_i \succ_{rpo} t$ or $s_i \sim_{rpo} t$ for some $1 \leq i \leq n$; or
  (2) $t = g(\bar{t})$ and $s \succ_{rpo} t_j$ for all $1 \leq j \leq m$ and either:
  (i) $f >_\mathcal{F} g$ or (ii) $f \approx_\mathcal{F} g$ and $\bar{s} \succ_{rpo}^{f,g} \bar{t}$;
- *$s \sim_{rpo} t$ iff (a) $s = t$; or (b) $s = f(\bar{s})$, $t = g(\bar{t})$, $f \approx_\mathcal{F} g$, and $\bar{s} \sim_{rpo}^{f,g} \bar{t}$;*

where $\succ_{rpo}^{f,g}$ and $\sim_{rpo}^{f,g}$ are the tuple extensions of $\succ_{rpo}$ and $\sim_{rpo}$ defined by:

- *$\langle s_1, \ldots s_n \rangle \succ_{rpo}^{f,g} \langle t_1, \ldots t_m \rangle$ iff one of the following holds:*
  (1) σ maps f and g to permutations μ_f and μ_g; and
  $\mu_f\langle s_1, \ldots, s_n \rangle \succ_{rpo}^{lex} \mu_g\langle t_1, \ldots, t_m \rangle$;
  (2) σ maps f and g to $\mathtt{mul}$; and $\langle s_1, \ldots s_n \rangle \succ_{rpo}^{mul} \langle t_1, \ldots t_m \rangle$.
- *$\langle s_1, \ldots s_n \rangle \sim_{rpo}^{f,g} \langle t_1, \ldots t_m \rangle$ iff one of the following holds:*

(1) σ maps f and g to μ_f and μ_g; and $\mu_f\langle s_1,\ldots,s_n\rangle \sim_{rpo}^{lex} \mu_g\langle t_1,\ldots,t_m\rangle$;
(2) σ maps f and g to mul; and $\langle s_1,\ldots s_n\rangle \sim_{rpo}^{mul} \langle t_1,\ldots t_m\rangle$.

Def. 6 can be specialized to capture the previous path orders by taking specific forms of status functions: LPO when σ maps all symbols to the identity permutation; lexicographic path order w.r.t. permutation (LPOS) when σ maps all symbols to some permutation; MPO when σ maps all symbols to mul.

The RPO termination problem is to determine for a given TRS if there exists a precedence and a status function such that the system is RPO-terminating. There are two variants of the problem: "strict-" and "quasi-RPO termination" depending on if the precedence $\geq_{\mathcal{F}}$ is strict or not (i.e., on whether $f \approx_{\mathcal{F}} g$ can hold for $f \neq g$). The corresponding decision problems, strict- and quasi-RPO termination, are decidable and NP complete [5]. In this paper we address the implementation of decision procedures for RPO termination problems by encoding them into corresponding SAT problems.

3 Encoding RPO Problems

We introduce an encoding τ which maps constraints of the form $s \succ_{rpo} t$ to propositional statements about the status and the precedence of the symbols in the terms s and t. A satisfying assignment for the encoding of such a constraint indicates a precedence and a status function such that the constraint holds.

The first part of the encoding is straightforward and similar to the one in [3,4]. All "missing" cases (e.g., $\tau(x \succ_{rpo} t)$ for variables x) are defined to be *false*.

$$\tau(f(\bar{s}) \succ_{rpo} t) = \bigvee_{i=1}^{n} (\tau(s_i \succ_{rpo} t) \vee \tau(s_i \sim_{rpo} t)) \quad \vee \quad \tau_2(f(\bar{s}) \succ_{rpo} t) \tag{1}$$

$$\tau_2(f(\bar{s}) \succ_{rpo} g(\bar{t})) = \bigwedge_{j=1}^{m} \tau(f(\bar{s}) \succ_{rpo} t_j) \wedge \begin{pmatrix} (f >_{\mathcal{F}} g) \vee \\ ((f \approx_{\mathcal{F}} g) \wedge \tau(\bar{s} \succ_{rpo}^{f,g} \bar{t})) \end{pmatrix} \tag{2}$$

$$\tau(s \sim_{rpo} s) = \mathit{true} \tag{3}$$

$$\tau(f(\bar{s}) \sim_{rpo} g(\bar{t})) = (f \approx_{\mathcal{F}} g) \wedge \tau(\bar{s} \sim_{rpo}^{f,g} \bar{t}) \tag{4}$$

The propositional encoding of the "partial order constraints" of the form $f >_{\mathcal{F}} g$ and $f \approx_{\mathcal{F}} g$ is performed following the approach applied in [3,4]. The basic idea is to interpret the symbols in $\mathcal{F}$ as indices in a partial order taking finite domain values from the set $\{0,\ldots,m-1\}$ where m is the number of symbols in $\mathcal{F}$. Each symbol $f \in \mathcal{F}$ is then represented using $\lceil \log_2 m \rceil$ propositional variables and the constraints are encoded as integer constraints on the binary representations.

In Sect. 3.1 and 3.2. we encode lexicographic comparisons w.r.t. permutations and multiset comparison. Then in Sect. 3.3 we combine them into $\succ_{rpo}^{f,g}$ and $\sim_{rpo}^{f,g}$.

3.1 Encoding Lexicographic Comparisons w.r.t. Permutation

For lexicographic comparisons with permutations, we associate with each symbol $f \in \mathcal{F}$ (of arity n) a permutation μ_f encoded through n^2 propositional variables

$f_{i,k}$ with $i, k \in \{1, \ldots, n\}$. Here, $f_{i,k}$ is *true* iff $\mu_f(i) = k$ (i.e., the i-th argument of $f(s_1, \ldots, s_n)$ is considered at k-th position when comparing lexicographically). To ease presentation, we define that $f_{i,k}$ is *false* for $k > n$.

For the encoding to be correct, we impose constraints on the variables $f_{i,k}$ to ensure that they indeed correspond to a permutation on $\{1, \ldots, n\}$. So for each $i \in \{1, \ldots, n\}$ there must be exactly one $k \in \{1, \ldots, n\}$ and for each $k \in \{1, \ldots, n\}$ there must be exactly one $i \in \{1, \ldots, n\}$ such that $f_{i,k}$ is *true*.

We denote by $one(b_1, \ldots, b_n)$ the constraint expressing that exactly one of the bits $b_1, \ldots, b_n$ is *true*. Then our encoding includes a formula of the form

$$\bigwedge_{f/n \in \mathcal{F}} \left(\bigwedge_{i=1}^{n} one(f_{i,1}, \ldots, f_{i,n}) \quad \wedge \quad \bigwedge_{k=1}^{n} one(f_{1,k}, \ldots, f_{n,k}) \right) \tag{5}$$

We apply a linear encoding to propositional logic for constraints of the form $one(b_1, \ldots, b_n)$ which introduces $\approx 2n$ fresh Boolean variables which we denote here as $one(b_i, \ldots, b_n)$ (expressing that one of the variables $b_i, \ldots, b_n$ is *true*) and $zero(b_i, \ldots, b_n)$ (expressing that all of the variables $b_i, \ldots, b_n$ are *false*) for $1 < i \leq n$. The encoding applies a ternary propositional connective $x \to y\,;z$ (denoting `if-then-else`) equivalent to $(x \to y) \wedge (\neg x \to z)$:

$$\bigwedge_{1 \leq i \leq n} \left(\begin{array}{c} \left(one(b_i, \ldots, b_n) \leftrightarrow \left(\begin{array}{r} (b_i \;\to\; zero(b_{i+1}, \ldots, b_n) \\ ;\;\; one(b_{i+1}, \ldots, b_n) \end{array} \right) \right) \\ \wedge\, (zero(b_i, \ldots, b_n) \leftrightarrow \neg b_i \wedge zero(b_{i+1}, \ldots, b_n)) \end{array} \right)$$

where $one(b_{n+1}, \ldots, b_n) = \mathit{false}$ and $zero(b_{n+1}, \ldots, b_n) = \mathit{true}$. This encoding introduces $\approx 2n$ conjuncts each involving a formula with at most 4 Boolean variables. So the encoding is more concise than the more straightforward one which introduces a quadratic number of conjuncts $\neg b_i \vee \neg b_j$ for all $1 \leq i < j \leq n$.

Now consider the encoding of $\bar{s} \sim_{rpo}^{f,g} \bar{t}$ and $\bar{s} \succ_{rpo}^{f,g} \bar{t}$ for the case where the arguments of f and g are compared lexicographically (thus, we use the notation $\sim_{lex}^{f,g}$ and $\succ_{lex}^{f,g}$). Like in Def. 6, let $\bar{s} = \langle s_1, \ldots, s_n \rangle$ and $\bar{t} = \langle t_1, \ldots, t_m \rangle$. Now equality constraints of the form $\bar{s} \sim_{lex}^{f,g} \bar{t}$ are encoded by stating that for all k, the arguments s_i and t_j used at the k-th position in the comparison (as denoted by $f_{i,k}$ and and $g_{j,k}$) must be equal. This implies that $\bar{s} \sim_{lex}^{f,g} \bar{t}$ only holds if $n = m$.

$$\tau(\bar{s} \sim_{lex}^{f,g} \bar{t}) = (n = m) \wedge \left(\bigwedge_{k=1}^{n} \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{m} f_{i,k} \wedge g_{j,k} \to \tau(s_i \sim_{rpo} t_j) \right) \tag{6}$$

To encode $\bar{s} \succ_{lex}^{f,g} \bar{t}$, we define auxiliary relations $\succ_{lex}^{f,g,k}$, where $k \in \mathbb{N}$ denotes that the k-th component of $\bar{s}$ and $\bar{t}$ is currently being compared. Thus $\succ_{lex}^{f,g} = \succ_{lex}^{f,g,1}$, since the comparison starts with the first component. For any k, there are three cases to consider when encoding $\bar{s} \succ_{lex}^{f,g,k} \bar{t}$. If there is no s_i that can be used for the k-th comparison (i.e., $k > n$), then we encode to *false*. If there is such an s_i but no such t_j (i.e., $m < k \leq n$), then we encode to *true*. If there are both an s_i and a t_j used for the k-th comparison (i.e., $f_{i,k}$ and $g_{j,k}$ hold), then we encode to a disjunction that either s_i is greater than t_j, or s_i is equal to t_j, and we continue the encoding at position $k + 1$. Since exactly one $f_{i,k}$ is *true*, the disjunction and conjunction over all $f_{i,k}$ (with $i \in \{1, ..., n\}$) coincide (similar for $g_{j,k}$). Here, we use a disjunction of conjunctions, as this will be more convenient in Sect. 4.

$$\tau(\bar{s} \succ_{lex}^{f,g,k} \bar{t}) = \begin{cases} \mathit{false} & \text{if } k > n \\ \mathit{true} & \text{if } m < k \leq n \\ \bigvee_{i=1}^{n} \Big(f_{i,k} \wedge \Big(\bigwedge_{j=1}^{m} g_{j,k} \rightarrow & \text{otherwise} \\ \quad (\tau(s_i \succ_{rpo} t_j) \vee (\tau(s_i \sim_{rpo} t_j) \wedge \tau(\bar{s} \succ_{lex}^{f,g,k+1} \bar{t})))\Big)\Big) & \end{cases} \tag{7}$$

Example 2. Consider again the TRS of Ex. 1(c):

$$\{ \ \mathsf{add}(x, \mathsf{0}) \to x \ , \ \ \mathsf{add}(x, \mathsf{s}(y)) \to \mathsf{add}(\mathsf{s}(x), y) \ \ \}$$

In the encoding of the constraints for the second rule, we have to encode the comparison $\langle x, \mathsf{s}(y)\rangle \succ_{lex}^{\mathsf{add},\mathsf{add},1} \langle \mathsf{s}(x), y\rangle$, which yields:

$$\begin{aligned} &\Big(\mathsf{add}_{1,1} \wedge \Big(\mathsf{add}_{1,1} \to \Big(\tau(x \succ_{rpo} \mathsf{s}(x)) \vee (\tau(x \sim_{rpo} \mathsf{s}(x)) \wedge \tau(\langle x, \mathsf{s}(y)\rangle \succ_{lex}^{\mathsf{add},\mathsf{add},2} \langle \mathsf{s}(x), y\rangle))\Big)\Big) \\ &\qquad \wedge \Big(\mathsf{add}_{2,1} \to \Big(\tau(x \succ_{rpo} y) \vee (\tau(x \sim_{rpo} y) \wedge \tau(\langle x, \mathsf{s}(y)\rangle \succ_{lex}^{\mathsf{add},\mathsf{add},2} \langle \mathsf{s}(x), y\rangle))\Big)\Big) \Big) \\ \vee &\Big(\mathsf{add}_{2,1} \wedge \Big(\mathsf{add}_{1,1} \to \Big(\tau(\mathsf{s}(y) \succ_{rpo} \mathsf{s}(x)) \vee (\tau(\mathsf{s}(y) \sim_{rpo} \mathsf{s}(x)) \wedge \tau(\langle x, \mathsf{s}(y)\rangle \succ_{lex}^{\mathsf{add},\mathsf{add},2} \langle \mathsf{s}(x), y\rangle))\Big)\Big) \\ &\qquad \wedge \Big(\mathsf{add}_{2,1} \to \Big(\tau(\mathsf{s}(y) \succ_{rpo} y) \vee (\tau(\mathsf{s}(y) \sim_{rpo} y) \wedge \tau(\langle x, \mathsf{s}(y)\rangle \succ_{lex}^{\mathsf{add},\mathsf{add},2} \langle \mathsf{s}(x), y\rangle))\Big)\Big) \Big) \end{aligned}$$

Seeing that $\tau(x \succ_{rpo} \mathsf{s}(x)) = \tau(x \sim_{rpo} \mathsf{s}(x)) = \tau(x \succ_{rpo} y) = \tau(x \sim_{rpo} y) = \tau(\mathsf{s}(y) \succ_{rpo} \mathsf{s}(x)) = \tau(\mathsf{s}(y) \sim_{rpo} \mathsf{s}(x)) = \mathit{false}$ and $\tau(\mathsf{s}(y) \succ_{rpo} y) = \mathit{true}$, the above formula can be simplified to $\mathsf{add}_{2,1} \wedge \neg \mathsf{add}_{1,1}$. Together with the constraint (5) which ensures that the variables $\mathsf{add}_{i,k}$ specify a valid permutation μ_{add}, this implies that $\mathsf{add}_{1,2}$ and $\neg \mathsf{add}_{2,2}$ must be *true*. And indeed, for the permutation $\mu_{\mathsf{add}} = \langle 2, 1\rangle$ the tuple $\mu_{\mathsf{add}}(\langle x, \mathsf{s}(y)\rangle) = \langle \mathsf{s}(y), x\rangle$ is greater than the tuple $\mu_{\mathsf{add}}(\langle \mathsf{s}(x), y\rangle) = \langle y, \mathsf{s}(x)\rangle$.

3.2 Encoding Multiset Comparisons

For multiset comparisons, we associate $\bar{s}$ and $\bar{t}$ with a multiset cover (γ, ε) encoded by $n * m$ propositional variables $\gamma_{i,j}$ and n variables ε_i. Here, $\gamma_{i,j}$ is *true* iff $\gamma(j) = i$ (s_i covers t_j) and ε_i is *true* iff $\varepsilon(i) = \mathit{true}$ (s_i is used for equality).

For the encoding to be correct, we again have to impose constraints on these variables to ensure that (γ, ε) indeed forms a multiset cover. So for each $j \in \{1, \ldots, m\}$ there must be exactly one $i \in \{1, \ldots, n\}$ such that $\gamma_{i,j}$ is *true*, and for each $i \in \{1, \ldots, n\}$, if ε_i is *true* then there must be exactly one $j \in \{1, \ldots, m\}$ such that $\gamma_{i,j}$ is *true*. Thus, our encoding includes the following formula:

$$\bigwedge_{j=1}^{m} \mathit{one}(\gamma_{1,j}, \ldots, \gamma_{n,j}) \quad \wedge \quad \bigwedge_{i=1}^{n} (\varepsilon_i \to \mathit{one}(\gamma_{i,1}, \ldots, \gamma_{i,m})) \tag{8}$$

Now we encode $\bar{s} \succ_{rpo}^{f,g} \bar{t}$ for the case where f and g have multiset status. To have an analogous notation to the case of lexicographic comparisons, we use the notation $\succ_{mul}^{f,g}$ instead of $\succ_{rpo}^{mul}$. This will also be convenient later in Sect. 4. The encoding of $\succsim_{mul}^{f,g}$, $\succ_{mul}^{f,g}$, and $\sim_{mul}^{f,g}$ is similar to Def. 3. To encode $\bar{s} \succsim_{mul}^{f,g} \bar{t}$, one has to require that if $\gamma_{i,j}$ and ε_i are *true*, $s_i \sim_{rpo} t_j$ holds, and else, if $\gamma_{i,j}$ is *true* and ε_i is not, $s_i \succ_{rpo} t_j$ holds. For $\succ^{f,g}_{mul}$, we must have at least one s_i that is not used for equality, and for $\sim^{f,g}_{mul}$, all s_i must be used for equality.

$$\tau(\bar{s} \succsim^{f,g}_{mul} \bar{t}) = \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{m} (\gamma_{i,j} \to ((\varepsilon_i \to \tau(s_i \sim_{rpo} t_j)) \wedge (\neg\varepsilon_i \to \tau(s_i \succ_{rpo} t_j)))) \tag{9}$$

$$\tau(\bar{s} \succ^{f,g}_{mul} \bar{t}) = \tau(\bar{s} \succsim^{f,g}_{mul} \bar{t}) \wedge \neg \bigwedge_{i=1}^{n} \varepsilon_i \tag{10}$$

$$\tau(\bar{s} \sim^{f,g}_{mul} \bar{t}) = \tau(\bar{s} \succsim^{f,g}_{mul} \bar{t}) \wedge \bigwedge_{i=1}^{n} \varepsilon_i \tag{11}$$

Example 3. Consider again the rules for the TRS from Ex. 1(b):

$$\Big\{ \ \mathsf{add}(x, \mathsf{0}) \to x \ , \ \ \mathsf{add}(x, \mathsf{s}(y)) \to \mathsf{s}(\mathsf{add}(y, x)) \ \ \Big\}.$$

In the encoding of the constraints for the second rule, we have to encode the comparison $\langle x, \mathsf{s}(y)\rangle \succ^{f,g}_{mul} \langle y, x\rangle$, which yields:

$$\begin{aligned}
&\Big(\gamma_{1,1} \to \Big((\varepsilon_1 \to \tau(x \sim_{rpo} y)) \wedge (\neg\varepsilon_1 \to \tau(x \succ_{rpo} y))\Big)\Big)\\
\wedge\ &\Big(\gamma_{1,2} \to \Big((\varepsilon_1 \to \tau(x \sim_{rpo} x)) \wedge (\neg\varepsilon_1 \to \tau(x \succ_{rpo} x))\Big)\Big)\\
\\
\wedge\ &\Big(\gamma_{2,1} \to \Big((\varepsilon_2 \to \tau(\mathsf{s}(y) \sim_{rpo} y)) \wedge (\neg\varepsilon_2 \to \tau(\mathsf{s}(y) \succ_{rpo} y))\Big)\Big)\\
\wedge\ &\Big(\gamma_{2,2} \to \Big((\varepsilon_2 \to \tau(\mathsf{s}(y) \sim_{rpo} x)) \wedge (\neg\varepsilon_2 \to \tau(\mathsf{s}(y) \succ_{rpo} x))\Big)\Big)
\end{aligned}$$

Seeing that $\tau(x \sim_{rpo} y) = \tau(x \succ_{rpo} y) = \tau(x \succ_{rpo} x) = \tau(\mathsf{s}(y) \sim_{rpo} y) = \tau(\mathsf{s}(y) \sim_{rpo} x) = \tau(\mathsf{s}(y) \succ_{rpo} x) =$ *false* and $\tau(x \sim_{rpo} x) = \tau(\mathsf{s}(y) \succ_{rpo} y) =$ *true*, the above formula can be simplified to $\neg\gamma_{1,1} \wedge (\neg\gamma_{1,2} \vee \varepsilon_1) \wedge (\neg\gamma_{2,1} \vee \neg\varepsilon_2) \wedge \neg\gamma_{2,2}$. Together with the constraint (8) which ensures that the variables $\gamma_{i,j}$ and ε_i specify a valid multiset cover (γ, ε), this implies that $\gamma_{2,1}$, $\neg\varepsilon_2$, $\gamma_{1,2}$, and ε_1 must hold. And indeed, for the multiset cover (γ, ε) with $\gamma(1) = 2$, $\gamma(2) = 1$, $\varepsilon(1) =$ *true*, and $\varepsilon(2) =$ *false*, the tuple $\langle x, \mathsf{s}(y)\rangle$ is greater than $\langle y, x\rangle$.

3.3 Combining Lexicographic and Multiset Comparisons

We have shown how to encode lexicographic and multiset comparisons. In order to combine $\succ^{f,g}_{lex}$ and $\succ^{f,g}_{mul}$ into $\succ^{f,g}_{rpo}$ as well as $\sim^{f,g}_{lex}$ and $\sim^{f,g}_{mul}$ into $\sim^{f,g}_{rpo}$, we introduce for each symbol $f \in \mathcal{F}$ a variable m_f, which is *true* iff the arguments of f are to be compared as multisets (i.e., the status function maps f to ***mul***).

$$\tau(\bar{s} \succ^{f,g}_{rpo} \bar{t}) = \Big(m_f \wedge m_g \wedge \tau(\bar{s} \succ^{f,g}_{mul} \bar{t})\Big) \vee \Big(\neg m_f \wedge \neg m_g \wedge \tau(\bar{s} \succ^{f,g,1}_{lex} \bar{t})\Big) \tag{12}$$

$$\tau(\bar{s} \sim^{f,g}_{rpo} \bar{t}) = \Big(m_f \wedge m_g \wedge \tau(\bar{s} \sim^{f,g}_{mul} \bar{t})\Big) \vee \Big(\neg m_f \wedge \neg m_g \wedge \tau(\bar{s} \sim^{f,g}_{lex} \bar{t})\Big) \tag{13}$$

Similar to Def. 6, the above encoding function τ can be specialized to other standard path orderings: lexicographic path order w.r.t. permutation (LPOS) when m_f is set to *false* for all $f \in \mathcal{F}$; LPO when additionally $f_{i,k}$ is set to *true* iff $i = k$; MPO when m_f is set to *true* for all $f \in \mathcal{F}$.

We conclude this section with an approximation of the size of the propositional formula obtained when encoding $s \succ_{rpo} t$ where $s = f(s_1, \ldots, s_n)$ and

$t = g(t_1, \ldots, t_m)$ with the total size of terms s and t being k. A single step of unfolding Def. 6 results in a formula containing at least n copies of t (with all its subterms) and m copies of s (with all its subterms) occurring in constraints of the form $s' \succ_{rpo} t'$. Hence, without memoing, the final encoding is clearly exponential in k. To obtain a polynomial encoding, we introduce sharing of common subformulas in the propositional formula. The approach is similar to that proposed by Tseitin to obtain a linear CNF transformation of Boolean formulas [22]. If $\tau(s' \succ_{rpo} t')$ occurs in the encoding $\tau(s \succ_{rpo} t)$, then we do not immediately perform the encoding of $s' \succ_{rpo} t'$ as well. Instead, we introduce a fresh Boolean variable of the form $X_{s' \succ_{rpo} t'}$, and encode also the meaning of such fresh variables. The encoding of $X_{s' \succ_{rpo} t'}$ is of the form $X_{s' \succ_{rpo} t'} \leftrightarrow \tau(s' \succ_{rpo} t')$. Again, when constructing $\tau(s' \succ_{rpo} t')$, all subformulas $\tau(s'' \succ_{rpo} t'')$ encountered are replaced by Boolean variables $X_{s'' \succ_{rpo} t''}$. In total, there are at most $\mathcal{O}(k^2)$ fresh Boolean variables to encode. As the encodings of multiset comparisons and lexicographic comparisons are both of size $\mathcal{O}(k^3)$, the size of the overall encoding is in $\mathcal{O}(k^5)$. Thus, the size of the encoding is indeed polynomial.[3]

4 RPO and Dependency Pairs

One of the most powerful and popular techniques for proving termination of term rewriting is the *dependency pair* (DP) method [1,10]. This method has proven highly successful for systems which are not simply terminating, i.e., where termination cannot be shown directly using simplification orders such as LPO, MPO, and RPO. Instead it is the combination of the simplification order, in our case RPO, with the DP method which significantly increases termination proving power. A main advantage of the DP method is that it permits the use of orders that are not monotonic and thus allows the application of *argument filters*.

An argument filter is a function which specifies for every function symbol f, which parts of a term $f(\ldots)$ may be eliminated before comparing terms with the underlying simplification order. More formally (we adopt notation of [17]):

Definition 7 (argument filter). *An* argument filter π *maps every n-ary function symbol to an argument position $i \in \{1, \ldots, n\}$ or to a (possibly empty) list $[i_1, \ldots, i_p]$ with $1 \le i_1 < \cdots < i_p \le n$. An argument filter π induces a mapping from terms to terms:*

$$\pi(t) = \begin{cases} t & \text{if } t \text{ is a variable} \\ \pi(t_i) & \text{if } t = f(t_1, \ldots, t_n) \text{ and } \pi(f) = i \\ f(\pi(t_{i_1}), \ldots, \pi(t_{i_p})) & \text{if } t = f(t_1, \ldots, t_n) \text{ and } \pi(f) = [i_1, \ldots, i_p] \end{cases}$$

For a relation $\succ$ on terms, let $\succ^\pi$ be the relation where $s \succ^\pi t$ holds if and only if $\pi(s) \succ \pi(t)$. An argument filter with $\pi(f) = i$ is called collapsing on f.

[3] A finer analysis shows that not all multiset and lexicographic comparisons are large. For example, for $s = \mathsf{f}(\mathsf{a}_1, \ldots, \mathsf{a}_n)$ and $t = \mathsf{g}(\mathsf{b}_1, \ldots, \mathsf{b}_n)$ with constants a_i and b_j, there is one comparison of two n-tuples with encoding size $\mathcal{O}(n^3)$, but the other $n^2 + 2n$ comparisons only need size $\mathcal{O}(n)$ each. In fact, one can show that the size of the overall encoding is in $\mathcal{O}(k^3)$.

Example 4. Consider the following rewrite rule (which will later turn out to be a dependency pair when proving the termination of Ex. 5):

$$\mathsf{ADD}'(x, \mathsf{s}(y), z) \to \mathsf{ADD}'(y, x, \mathsf{s}(z))$$

The rule cannot be oriented by RPO. Lexicographic comparison fails as the first two arguments are swapped. Multiset comparison is prevented by the third argument ($\mathsf{s}(z)$ cannot be covered). But an argument filter $\pi(\mathsf{ADD}') = \{1, 2\}$, which eliminates the third argument of ADD', enables the multiset comparison.

While very powerful in the context of the DP method, argument filters also present a severe bottleneck for automation, as the search space for argument filters is enormous (exponential in the arities of the function symbols). A SAT encoding for the LPO with argument filters is presented in [4] where the combined constraints on the argument filter and on the precedence of the LPO (which influence each other) are encoded as a single SAT problem.

This paper applies a similar strategy to combine RPO with argument filters. The combined search for an argument filter π, a precedence $>_{\mathcal{F}}$, and a status function σ are encoded into a single propositional formula. This formula is satisfiable iff there is an argument filter and an RPO which orient a set of inequalities. Each model of the encoding indicates such an argument filter and RPO.

4.1 Dependency Pairs and Argument Filters

We provide here a simplified presentation of the DP method and refer to [1,10] for further details. For a TRS $\mathcal{R}$ over the symbols $\mathcal{F}$, the *defined* symbols $\mathcal{D}_{\mathcal{R}} \subseteq \mathcal{F}$ consist of all root symbols of left-hand sides of $\mathcal{R}$. The signature $\mathcal{F}$ is extended by a fresh *tuple symbol* F for each defined symbol $f \in \mathcal{D}_{\mathcal{R}}$. Then for each rule $f(s_1, \ldots, s_n) \to r$ in $\mathcal{R}$ and each subterm $g(t_1, \ldots, t_m)$ of r with $g \in \mathcal{D}_{\mathcal{R}}$, $F(s_1, \ldots, s_n) \to G(t_1, \ldots, t_m)$ is a *dependency pair*, intuitively indicating that a function call to f may lead to a function call to g. The set of dependency pairs of $\mathcal{R}$ is denoted $DP(\mathcal{R})$. So in Ex. 1(b), add is the only defined symbol and there is one dependency pair: $\mathsf{ADD}(x, s(y)) \to \mathsf{ADD}(y, x)$.

The main result of the DP method states that a TRS $\mathcal{R}$ is terminating iff there is no infinite $\mathcal{R}$-chain of its dependency pairs $\mathcal{P} = DP(\mathcal{R})$. This means that there is no infinite sequence of dependency pairs $s_1 \to t_1, s_2 \to t_2, \ldots$ from $\mathcal{P}$ such that for all i there is a substitution σ_i where $t_i\sigma_i$ is terminating w.r.t. $\mathcal{R}$ and $t_i\sigma_i \to_{\mathcal{R}}^* s_{i+1}\sigma_{i+1}$. Termination proofs in the DP method are stated in terms of *DP problems* which are pairs of the form $(\mathcal{P}, \mathcal{R})$ where $\mathcal{P}$ is a set of dependency pairs and $\mathcal{R}$ a TRS. Such a pair is read as posing the question: "Is there an infinite $\mathcal{R}$-chain of dependency pairs from $\mathcal{P}$?" Hence, termination of $\mathcal{R}$ is stated as the initial DP problem $(\mathcal{P}, \mathcal{R})$ with $\mathcal{P} = DP(\mathcal{R})$.

Starting from the initial DP problem, termination proofs repeatedly restrict $(\mathcal{P}, \mathcal{R})$ to obtain a smaller DP problem $(\mathcal{P}', \mathcal{R})$ with $\mathcal{P}' \subset \mathcal{P}$ while maintaining a soundness property to guarantee that there is an infinite $\mathcal{R}$-chain of pairs from $\mathcal{P}'$ whenever there is an infinite $\mathcal{R}$-chain of pairs from $\mathcal{P}$. Thus, if one reaches the DP problem $(\varnothing, \mathcal{R})$, then termination is proved.

One of the main techniques to restrict DP problems involves the notion of a *reduction pair* $(\succsim, \succ)$ where $\succsim$ is reflexive, transitive, monotonic, and stable and $\succ$ is a stable well-founded order compatible with $\succsim$ (i.e., $\succsim \circ \succ \subseteq \succ$ or $\succ \circ \succsim \subseteq \succ$). But $\succ$ need not be monotonic. Given a DP problem $(\mathcal{P}, \mathcal{R})$ and a reduction pair $(\succsim, \succ)$, the technique requires that (a) the dependency pairs in $\mathcal{P}$ are weakly or strictly decreasing and, (b) all rules in $\mathcal{R}$ are weakly decreasing. Then it is sound to obtain $\mathcal{P}'$ by removing all strictly decreasing dependency pairs from $\mathcal{P}$. It is possible to further strengthen the approach [12,13] by introducing a notion of *usable rules* and considering these in (b) instead of all rules. Arts and Giesl show in [1] that if $(\succsim, \succ)$ is a reduction pair and π is an argument filter, then $(\succsim^\pi, \succ^\pi)$ is also a reduction pair. In particular, we focus on reduction pairs of this form to prove termination of TRS where the direct application of RPO fails.

Example 5. Building on the rule from Ex. 4, consider the TRS (on the left) for addition using an accumulator and its three dependency pairs (on the right):

$$\begin{array}{rlcrl}
\mathsf{add}(x,y) & \to \mathsf{add}'(x,y,\mathsf{0}) & \qquad & \mathsf{ADD}(x,y) & \to \mathsf{ADD}'(x,y,\mathsf{0}) \\
\mathsf{add}'(\mathsf{0},\mathsf{0},z) & \to z & & & \\
\mathsf{add}'(\mathsf{s}(x),y,z) & \to \mathsf{add}'(x,y,\mathsf{s}(z)) & & \mathsf{ADD}'(\mathsf{s}(x),y,z) & \to \mathsf{ADD}'(x,y,\mathsf{s}(z)) \\
\mathsf{add}'(x,\mathsf{s}(y),z) & \to \mathsf{add}'(y,x,\mathsf{s}(z)) & & \mathsf{ADD}'(x,\mathsf{s}(y),z) & \to \mathsf{ADD}'(y,x,\mathsf{s}(z))
\end{array}$$

To orient the DPs we use $(\succsim^\pi_{rpo}, \succ^\pi_{rpo})$ where $\pi(\mathsf{ADD})=\pi(\mathsf{ADD}')=[1,2]$, $\pi(\mathsf{s})=[1]$, and where $\succsim_{rpo}$ and $\succ_{rpo}$ are induced by the precedence $\mathsf{ADD} >_{\mathcal{F}} \mathsf{ADD}'$ and the status function σ that maps ADD and ADD' to mul. Since the problematic third accumulator argument (in the last two DPs) is filtered away, all three DPs are strictly decreasing and can be removed, as there are no usable rules. This results in the DP problem $(\varnothing, \mathcal{R})$ which proves termination for the TRS.

4.2 Encoding RPO with Argument Filters

To encode argument filters, each n-ary function symbol $f \in \mathcal{F}$ is associated with n propositional variables $f_1, \ldots, f_n$, and another variable list_f. Here, f_i is *true* iff $i \in \pi(f)$ or $i = \pi(f)$, and list_f is *true* iff π is not collapsing on f. To ensure that these $n+1$ propositional variables indeed correspond to an argument filter, we impose the following constraints which express that if π collapses f then it is replaced by exactly one of its subterms:

$$\neg \mathit{list}_f \to \mathit{one}(f_1, \ldots, f_n).$$

To encode the combination of RPO with argument filters, consider again the equations (1) - (4). Each reference to a subterm must now be "wrapped" by the question: "has this subterm been filtered by π?" In the following, similar to the encoding of Sect. 3, all "missing" cases are defined to be *false*. Equations (1′) - (3′) enhance Equations (1) - (3). Equations (4a′) and (4b′) enhance Equation (4) in the cases when one of the terms is a variable x and (4c′) considers the other case and examines whether the filter collapses the root symbols or not.

$$\tau(f(\bar{s}) \succ_{rpo}^{\pi} t) = \bigvee_{i=1}^{n} \left(f_i \wedge \left(\begin{array}{l} \tau(s_i \succ_{rpo}^{\pi} t) \quad \vee \\ (list_f \wedge \tau(s_i \sim_{rpo}^{\pi} t)) \end{array} \right) \right) \;\vee\; \tau_2(f(\bar{s}) \succ_{rpo}^{\pi} t) \tag{1′}$$

$$\begin{aligned} \tau_2(f(\bar{s}) \succ_{rpo}^{\pi} g(\bar{t})) = {} & \bigwedge_{j=1}^{m} \left(g_j \to \tau(f(\bar{s}) \succ_{rpo}^{\pi} t_j) \right) \quad \wedge \\ & \left(list_g \to \left(list_f \wedge \left((f >_{\mathcal{F}} g) \vee ((f \approx_{\mathcal{F}} g) \wedge \tau(\bar{s} \succ_{rpo}^{f,g,\pi} \bar{t})) \right) \right) \right) \end{aligned} \tag{2′}$$

$$\tau(s \sim_{rpo}^{\pi} s) = true \tag{3′}$$

$$\tau(f(\bar{s}) \sim_{rpo}^{\pi} x) = (\neg list_f \wedge \bigwedge_{i=1}^{n} \left(f_i \to \tau(s_i \sim_{rpo}^{\pi} x) \right) \quad \text{for variables } x \tag{4a′}$$

$$\tau(x \sim_{rpo}^{\pi} g(\bar{t})) = (\neg list_g \wedge \bigwedge_{j=1}^{m} \left(g_j \to \tau(x \sim_{rpo}^{\pi} t_j) \right) \quad \text{for variables } x \tag{4b′}$$

$$\begin{aligned} \tau(f(\bar{s}) \sim_{rpo}^{\pi} g(\bar{t})) = {} & \left(\neg list_f \to \bigwedge_{i=1}^{n} \left(f_i \to \tau(s_i \sim_{rpo}^{\pi} g(\bar{t})) \right) \right) \quad \wedge \\ & \left((list_f \wedge \neg list_g) \to \bigwedge_{j=1}^{m} \left(g_j \to \tau(f(\bar{s}) \sim_{rpo}^{\pi} t_j) \right) \right) \quad \wedge \\ & \left((list_f \wedge list_g) \to \left((f \approx_{\mathcal{F}} g) \wedge \tau(\bar{s} \sim_{rpo}^{f,g,\pi} \bar{t}) \right) \right) \end{aligned} \tag{4c′}$$

For the lexicographic comparison with permutations, we enhance Formula (5) to specify the relation between filters and permutations. Only non-filtered arguments are permuted. Moreover, for an n-ary symbol f with $\ell < n$ non-filtered arguments, the permutation should map all ℓ non-filtered arguments to positions from $\{1, \ldots, \ell\}$. Formula (5a′) states that if some argument of f is considered at the k-th position (i.e., some $f_{i,k}$ is *true*), then there is exactly one such argument. Formula (5b′) specifies that filtered arguments may not be used in the permutation. So if the i-th argument of f is filtered (i.e., f_i is *false*), then the permutation variables $f_{i,k}$ (for $1 \leq k \leq n$) are also *false*. Formula (5c′) states that if the i-th argument of f is not filtered (i.e., f_i is *true*), then the i-th argument of f is considered at exactly one position in the permutation. Finally, Formula (5d′) expresses that all ℓ non-filtered arguments are permuted "to the left", i.e., to positions from $\{1, \ldots, \ell\}$. Hence, if an argument is mapped to position k, then some argument is also mapped to position $k-1$.

$$\bigwedge_{k=1}^{n} \left(\bigvee_{i=1}^{n} f_{i,k} \to one(f_{1,k}, \ldots, f_{n,k}) \right) \tag{5a′}$$

$$\bigwedge_{i=1}^{n} \left(\neg f_i \to \bigwedge_{k=1}^{n} \neg f_{i,k} \right) \tag{5b′}$$

$$\bigwedge_{i=1}^{n} (f_i \to one(f_{i,1}, \ldots, f_{i,n})) \tag{5c′}$$

$$\bigwedge_{k=2}^{n} \left(\bigvee_{i=1}^{n} f_{i,k} \to \bigvee_{i=1}^{n} f_{i,k-1} \right) \tag{5d′}$$

For the encoding of $f(\bar{s}) \sim_{rpo}^{f,g,\pi} g(\bar{t})$ and $f(\bar{s}) \succ_{rpo}^{f,g,\pi} g(\bar{t})$ when the arguments of f and g are compared lexicographically, we use the notation $\sim_{lex}^{f,g,\pi}$ and $\succ_{lex}^{f,g,\pi}$. For an equality constraint of the form $\bar{s} \sim_{lex}^{f,g,\pi} \bar{t}$ we enhance Equation (6). There must be a one-to-one correspondence between the non-filtered arguments of $\bar{s}$ and of $\bar{t}$ via the permutations for f and for g. To express this, we use a constraint of the form *eq_arity(f,g)* in Equation (6′) which states that the number of non-filtered arguments of f and of g are the same. It corresponds to the constraint $(n = m)$ in Equation (6) and is encoded as

$$eq\_arity(f,g) = \bigwedge_{k=1}^{\max(n,m)} \left(\bigvee_{i=1}^{n} f_{i,k} \leftrightarrow \bigvee_{j=1}^{m} g_{j,k} \right)$$

$$\tau(\bar{s} \sim_{lex}^{f,g,\pi} \bar{t}) = eq\_arity(f,g) \wedge \bigwedge_{k=1}^{n} \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{m} \Big(f_{i,k} \wedge g_{j,k} \rightarrow \tau(s_i \sim_{rpo}^{\pi} t_j) \Big) \tag{6′}$$

Next we enhance Equation (7) to define $\succ_{rpo}^{f,g,\pi} = \succ_{rpo}^{f,g,1,\pi}$. For $m < k \leq n$ we now require that f considers an argument at the k-th position. The remaining cases are structurally identical to the corresponding cases of Equation (7).

$$\tau(\bar{s} \succ_{lex}^{f,g,k,\pi} \bar{t}) = \begin{cases} false & \text{if } k > n \\ \bigvee_{i=1}^{n} f_{i,k} & \text{if } m < k \leq n \\ \bigvee_{i=1}^{n} \Big(f_{i,k} \wedge \Big(\bigwedge_{j=1}^{m} g_{j,k} \rightarrow & \text{otherwise} \\ \quad (\tau(s_i \succ_{rpo}^{\pi} t_j) \vee (\tau(s_i \sim_{rpo}^{\pi} t_j) \wedge \tau(\bar{s} \succ_{lex}^{f,g,k+1,\pi} \bar{t}))) \Big) \Big) & \end{cases} \tag{7′}$$

For the multiset comparison, we enhance Formula (8) such that the multiset cover only considers non-filtered arguments of $f(\bar{s})$ and $g(\bar{t})$. Formula (8a′) states that if the j-th argument of g is not filtered (i.e., g_j is *true*), then there must be exactly one argument of f that covers it. Formula (8b′) states that if the i-th argument of f is filtered (i.e., f_i is *false*), then it cannot cover any arguments of g. Formula (8c′) specifies that if the j-th argument of g is filtered (i.e., g_j is *false*), then there is no argument of f that covers it. Finally, Formula (8d′) is taken straight from the original Formula (8).

$$\bigwedge_{j=1}^{m} (g_j \rightarrow one(\gamma_{1,j}, \ldots, \gamma_{n,j})) \tag{8a′}$$

$$\bigwedge_{i=1}^{n} \left(\neg f_i \rightarrow \neg \bigvee_{j=1}^{m} \gamma_{i,j} \right) \tag{8b′}$$

$$\bigwedge_{j=1}^{m} \left(\neg g_j \rightarrow \neg \bigvee_{i=1}^{n} \gamma_{i,j} \right) \tag{8c′}$$

$$\bigwedge_{i=1}^{n} \left(\varepsilon_i \rightarrow one(\gamma_{i,1}, \ldots, \gamma_{i,m}) \right) \tag{8d′}$$

Now we define $\tau(\bar{s} \succsim_{mul}^{f,g,\pi} \bar{t}) = \tau(\bar{s} \succsim_{mul}^{f,g} \bar{t})$. For the encoding of $\succ_{mul}^{f,g,\pi}$ and $\sim_{mul}^{f,g,\pi}$, we restrict Equations (10) and (11) to arguments that are not filtered:

$$\tau(\bar{s} \succ_{mul}^{f,g,\pi} \bar{t}) = \tau(\bar{s} \succsim_{mul}^{f,g} \bar{t}) \wedge \neg \bigwedge_{i=1}^{n} (f_i \rightarrow \varepsilon_i) \tag{10′}$$

$$\tau(\bar{s} \sim_{mul}^{f,g,\pi} \bar{t}) = \tau(\bar{s} \succsim_{mul}^{f,g} \bar{t}) \wedge \bigwedge_{i=1}^{n} (f_i \rightarrow \varepsilon_i) \tag{11′}$$

Finally, for the combination of lexicographic and multiset comparisons, we simply change the equations (12) and (13) to use $\succ_{mul}^{f,g,\pi}$ instead of $\succ_{mul}^{f,g}$ etc.:

$$\tau(\bar{s} \succ_{rpo}^{f,g,\pi} \bar{t}) = \Big(m_f \wedge m_g \wedge \tau(\bar{s} \succ_{mul}^{f,g,\pi} \bar{t}) \Big) \vee \Big(\neg m_f \wedge \neg m_g \wedge \tau(\bar{s} \succ_{lex}^{f,g,1,\pi} \bar{t}) \Big) \tag{12′}$$

$$\tau(\bar{s} \sim_{rpo}^{f,g,\pi} \bar{t}) = \Big(m_f \wedge m_g \wedge \tau(\bar{s} \sim_{mul}^{f,g,\pi} \bar{t}) \Big) \vee \Big(\neg m_f \wedge \neg m_g \wedge \tau(\bar{s} \sim_{lex}^{f,g,\pi} \bar{t}) \Big) \tag{13′}$$

Example 6. We solved the inequality $\mathsf{ADD}'(x, \mathsf{s}(y), z) \mathrel{\underset{(\sim)}{\succ}} \mathsf{ADD}'(y, x, \mathsf{s}(z))$ in Ex. 5 by the argument filter $\pi(\mathsf{ADD}') = [1, 2]$ and RPO. To find such argument filters and the status and precedence of the RPO, such inequalities are now encoded into propositional formulas. Indeed, the formula resulting from our inequality is satisfiable by the corresponding setting of the propositional variables (i.e., $m_{\mathsf{ADD}'} = list_{\mathsf{ADD}'} = \mathsf{ADD}'_1 = \mathsf{ADD}'_2 = true$ and $\mathsf{ADD}'_3 = false$). So we use a multiset comparison for the filtered tuples $\langle x, \mathsf{s}(y)\rangle$ and $\langle y, x\rangle$. Hence, as in Ex. 3 we set $\gamma_{1,2} = \varepsilon_1 = \gamma_{2,1} = true$ and $\varepsilon_2 = false$.

In recent refinements of the DP method [12], the choice of the argument filter π also influences the set of usable rules which contribute to the inequalities that have to be oriented. We showed in [4] how to extend the encoding of LPO and argument filters in order to take this refinement into account as well. In a similar way, this refinement can also be integrated into our encoding of RPO and argument filters. Finally, similar to Sect. 3.3 one can easily show that the size of our encoding is again polynomial.

5 Implementation and Experiments

We implemented the encoding of RPO (also in combination with argument filters) in the termination analyzer AProVE [11], using the SAT4J solver [20]. (We also tried other SAT solvers like MiniSAT [7] and obtained similar results.) The encoding can also be restricted to instances of RPO like LPO or MPO.

We tested the implementation on all 865 TRSs from the TPDB [21]. The experiments were run on a 2.2 GHz AMD Athlon 64 with a time-out of 60 seconds (as in the *International Termination Competition* [19]). For each encoding we give the number of TRSs which could be proved terminating (with the number of time-outs in brackets) and the analysis time (in seconds) for the full collection.

The first two rows compare our new SAT-based approach for direct application of path orders to the previous dedicated solvers for path orders in AProVE 1.2 which did not use SAT solving. The last two rows give a similar comparison for path orders within the DP framework. The columns contain the data for LPO with strict and non-strict precedence (denoted *lpo*/*qlpo*), for LPO with status (*lpos*/*qlpos*), for MPO (*mpo*/*qmpo*), and for RPO with status (*rpo*/*qrpo*).

	Solver	*lpo*	*qlpo*	*lpos*	*qlpos*	*mpo*	*qmpo*	*rpo*	*qrpo*
1	SAT-based	**123** (0)	**127** (0)	**141** (0)	**155** (0)	**92** (0)	**98** (0)	**146** (0)	**162** (0)
	(direct)	**31.0**	**44.7**	**26.1**	**40.6**	**49.4**	**74.2**	**50.0**	**85.3**
2	dedicated	**123** (5)	**127**(16)	**141** (6)	**154**(45)	**92** (7)	**98**(31)	**145**(10)	**158** (65)
	(direct)	**334.4**	**1426.3**	**460.4**	**3291.7**	**653.2**	**2669.1**	**908.6**	**4708.2**
3	SAT-based	**357** (0)	**389** (0)	**362** (0)	**395** (2)	**369** (0)	**408** (1)	**375** (0)	**416** (2)
	(arg. filt.)	**79.3**	**199.6**	**69.0**	**261.1**	**110.9**	**267.8**	**108.8**	**331.4**
4	dedicated	**350**(55)	**374**(79)	**355**(57)	**380**(92)	**359**(69)	**391**(82)	**364**(74)	**394**(102)
	(arg. filt.)	**4039.6**	**5469.4**	**4522.8**	**6476.5**	**5169.7**	**5839.5**	**5536.6**	**7186.1**

The table shows that with our new SAT encoding, performance improves by orders of magnitude over existing solvers both for direct analysis with path orders and for the combination of path orders and argument filters in the DP framework. Note that without a time-out, this effect would be aggravated. By using SAT, the number of time-outs reduces dramatically from up to 102 to at most 2. The two remaining SAT examples with time-out have function symbols of high arity and can only be shown terminating by further sophisticated termination techniques in addition to RPO. Apart from these two, there are only 15 examples that take longer than two seconds and only 3 of these take longer than 10 seconds. The table also shows that the use of RPO instead of LPO increases power substantially, while in the SAT-based setting, runtimes increase only mildly.

6 Conclusion

In [4] we demonstrated the power of propositional encoding and application of SAT solving to LPO termination analysis. This paper extends this approach to the more powerful class of recursive path orders. The main new challenges were the encoding of multiset comparisons and of lexicographic comparisons w.r.t. permutations as well as the combination with argument filters.

We solved this problem by a novel SAT encoding which combines all of the constraints originating from these notions into a single search process. Through implementation and experimentation we showed that our encoding leads to speedups in orders of magnitude over existing termination tools as well as increased termination proving power. To experiment with our SAT-based implementation and for further details on our experiments please visit our evaluation web site at `http://aprove.informatik.rwth-aachen.de/eval/SATRPO/`.

References

1. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. Theoretical Computer Science 236, 133–178 (2000)
2. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1998)
3. Codish, M., Lagoon, V., Stuckey, P.J.: Solving partial order constraints for LPO termination. In: Pfenning, F. (ed.) RTA 2006. LNCS, vol. 4098, pp. 4–18. Springer, Heidelberg (2006)
4. Codish, M., Schneider-Kamp, P., Lagoon, V., Thiemann, R., Giesl, J.: SAT solving for argument filterings. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS (LNAI), vol. 4246, pp. 30–44. Springer, Heidelberg (2006)
5. Comon, H., Treinen, R.: Ordering constraints on trees. In: Tison, S. (ed.) CAAP 1994. LNCS, vol. 787, pp. 1–14. Springer, Heidelberg (1994)
6. Dershowitz, N.: Orderings for term-rewriting systems. Theoretical Computer Science 17, 279–301 (1982)
7. Eén, N., Sörensson, N.: An Extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
8. Endrullis, J., Waldmann, J., Zantema, H.: Matrix interpretations for proving termination of term rewriting. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 574–588. Springer, Heidelberg (2006)

9. Fuhs, C., Giesl, J., Middeldorp, A., Schneider-Kamp, P., Thiemann, R., Zankl, H.: SAT solving for termination analysis with polynomial interpretations. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 340–354. Springer, Heidelberg (2007)
10. Giesl, J., Thiemann, R., Schneider-Kamp, P.: The dependency pair framework: Combining techniques for automated termination proofs. In: Baader, F., Voronkov, A. (eds.) LPAR 2004. LNCS (LNAI), vol. 3452, pp. 301–331. Springer, Heidelberg (2005)
11. Giesl, J., Schneider-Kamp, P., Thiemann, R.: AProVE 1.2: Automatic termination proofs in the dependency pair framework. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 281–286. Springer, Heidelberg (2006)
12. Giesl, J., Thiemann, R., Schneider-Kamp, P., Falke, S.: Mechanizing and improving dependency pairs. Journal of Automated Reasoning 37(3), 155–203 (2006)
13. Hirokawa, N., Middeldorp, A.: Tyrolean termination tool: Techniques and features. Information and Computation 205(4), 474–511 (2007)
14. Hofbauer, D., Waldmann, J.: Termination of string rewriting with matrix interpretations. In: Pfenning, F. (ed.) RTA 2006. LNCS, vol. 4098, pp. 328–342. Springer, Heidelberg (2006)
15. Kamin, S., Lévy, J.J.: Two generalizations of the recursive path ordering. Unpublished Manuscript, University of Illinois, IL, USA (1980)
16. Kurihara, M., Kondo, H.: Efficient BDD encodings for partial order constraints with application to expert systems in software verification. In: Orchard, B., Yang, C., Ali, M. (eds.) IEA/AIE 2004. LNCS (LNAI), vol. 3029, pp. 827–837. Springer, Heidelberg (2004)
17. Kusakari, K., Nakamura, M., Toyama, Y.: Argument filtering transformation. In: Nadathur, G. (ed.) PPDP 1999. LNCS, vol. 1702, pp. 47–61. Springer, Heidelberg (1999)
18. Lescanne, P.: Computer experiments with the REVE term rewriting system generator. In: Demers, A., Teitelbaum, T. (eds.) POPL 1983, pp. 99–108 (1983)
19. Marché, C., Zantema, H.: The termination competition. In: Baader, F. (ed.) Proc. RTA 2007. LNCS, vol. 4533, pp. 303–313. Springer, Heidelberg (2007)
20. SAT4J satisfiability library for Java, `http://www.sat4j.org`
21. The termination problem data base, `http://www.lri.fr/~marche/tpdb/`
22. Tseitin, G.: On the complexity of derivation in propositional calculus. In: Studies in Constructive Mathematics and Mathematical Logic, pp. 115–125 (1968)
23. Zankl, H., Hirokawa, N., Middeldorp, A.: Constraints for argument filterings. In: van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plášil, F. (eds.) SOFSEM 2007. LNCS, vol. 4362, pp. 579–590. Springer, Heidelberg (2007)
24. Zankl, H., Middeldorp, A.: Satisfying KBO constraints. In: Baader, F. (ed.) RTA 2007. LNCS, vol. 4533, pp. 389–403. Springer, Heidelberg (2007)