Noname manuscript No.
(will be inserted by the editor)

SAT Solving for Termination Proofs with Recursive Path Orders and Dependency Pairs

Michael Codish · Jürgen Giesl · Peter Schneider-Kamp · René Thiemann

the date of receipt and acceptance should be inserted later

Abstract This paper introduces a propositional encoding for recursive path orders (RPO), in connection with dependency pairs. Hence, we capture in a uniform setting all common instances of RPO, i.e., lexicographic path orders (LPO), multiset path orders (MPO), and lexicographic path orders with status (LPOS). This facilitates the application of SAT solvers for termination analysis of term rewrite systems (TRSs).

We address four main inter-related issues and show how to encode them as satisfiability problems of propositional formulas that can be efficiently handled by SAT solving: (A) the lexicographic comparison w.r.t. a *permutation* of the arguments; (B) the *multiset extension* of a base order; (C) the combined search for a path order together with an *argument filter* to orient a set of inequalities; and (D) how the choice of the argument filter influences the set of inequalities that have to be oriented (so-called *usable rules*).

We have implemented our contributions in the termination prover AProVE. Extensive experiments show that by our encoding and the application of SAT solvers one obtains speedups in orders of magnitude as well as increased termination proving power.

Keywords Termination · SAT Solving · Term Rewriting · Recursive Path Order · Dependency Pairs

Supported by the G.I.F. under grant 966-116.6 and by the DFG under grant GI 274/5-3.

M. Codish
Department of Computer Science, Ben-Gurion University of the Negev, PoB 653, Beer-Sheva, Israel 84105, E-mail: mcodish@cs.bgu.ac.il

J. Giesl
LuFG Informatik 2, RWTH Aachen University, Ahornstr. 55, 52074 Aachen, Germany, E-mail: giesl@informatik.rwth-aachen.de

P. Schneider-Kamp
Department of Mathematics & Computer Science, University of Southern Denmark, Campusvej 55, DK-5230 Odense M, Denmark, E-mail: petersk@imada.sdu.dk

R. Thiemann
Institute of Computer Science, University of Innsbruck, Techniker Str. 21a, A-6020 Innsbruck, Austria, E-mail: rene.thiemann@uibk.ac.at

1 Introduction

One of the the most fundamental decision problems in computer science is the halting problem, i.e., given a program and an input, decide whether the program terminates after finitely many steps. While Turing showed this problem to be undecidable in general, developing analysis techniques that can automatically prove termination for many programs is of great practical interest.

In this paper, we focus on methods to prove termination of *term rewrite systems*. The main theme when proving termination of TRSs is the use of well-founded orders (i.e., orders $\succ$ where there is no infinite decreasing sequence $t_0 \succ t_1 \succ \ldots$). Roughly speaking, if all rules of a TRS $\mathcal{R}$ are decreasing w.r.t. some well-founded order $\succ$, then termination is proven. This directly yields a general strategy to prove termination: set up a constraint $\ell \succ r$ for each rule $\ell \to r$ and try to find a suitable well-founded order $\succ$ satisfying all constraints. Hence, proving termination becomes a search problem.

Recent techniques for termination proving involve iterative methods. Here, one generates constraints which allow to identify some rules which do not cause non-termination and removes them. The termination proof then continues iteratively until all rules have been removed. Usually, all rules have to be at least weakly decreasing (i.e., all rules $\ell \to r$ of the TRS $\mathcal{R}$ have to satisfy $\ell \succsim r$ for a quasi-order $\succsim$ that is "compatible" with $\succ$). Then one can remove those rules that are strictly decreasing (i.e., those rules $\ell \to r$ satisfying $\ell \succ r$). If the constraints can be solved, one can continue to prove termination of the simplified system, searching for other orders until no rules remain. In this setting, the front-end of the termination prover produces the following Constraint (1), where the disjunction enforces that at least one rule can be removed. Then the back-end of the termination prover has to search for a well-founded order which satisfies the constraint. We note that here, $\succ$ has to be closed under contexts. We call this constraint the *rule removal constraint*.

$$\boxed{\bigwedge_{\ell \to r \in \mathcal{R}} \ell \succsim r \quad \wedge \quad \bigvee_{\ell \to r \in \mathcal{R}} \ell \succ r} \tag{1}$$

An improvement of this approach is the so-called *dependency pair* (DP) method [1, 22, 24, 26] used in most TRS termination tools. Here, one regards both the rules of the TRS $\mathcal{R}$ and a set $\mathcal{P}$ of additional rules called *dependency pairs* (which result from recursive function calls). In this approach all rules $\mathcal{R}$ and all DPs $\mathcal{P}$ have to be weakly decreasing and one can remove those DPs that are strictly decreasing. In this approach, the front-end of the termination prover generates the following constraint where $\succ$ is no longer required to be closed under contexts. We call this constraint the *dependency pair constraint*.

$$\boxed{\bigwedge_{\ell \to r \in \mathcal{P} \cup \mathcal{R}} \ell \succsim r \quad \wedge \quad \bigvee_{\ell \to r \in \mathcal{P}} \ell \succ r} \tag{2}$$

This can be improved further by observing that under certain conditions, one does not have to require $\ell \succsim r$ for all rules $\ell \to r$ of $\mathcal{R}$. Instead, it suffices to require $\ell \succsim r$ only for a certain subset of $\mathcal{R}$, viz. the so-called *usable rules* $\mathcal{U}(\mathcal{P}, \mathcal{R}, \pi)$ [1, 24, 27]. This leads to the following constraint which we call the *usable rule constraint*.

$$\boxed{\bigwedge_{\ell \to r \in \mathcal{P} \cup \mathcal{U}(\mathcal{P},\mathcal{R},\pi)} \ell \succsim r \quad \wedge \quad \bigvee_{\ell \to r \in \mathcal{P}} \ell \succ r} \tag{3}$$

Note that here, the constraint itself depends on the order that is searched for (since the order $\succ$ determines which rules are considered to be "usable", i.e., the set $\mathcal{U}(\mathcal{P}, \mathcal{R}, \pi)$ depends on $\succ$).

So a main problem for termination analysis of TRSs is to solve constraints like (1)–(3). In other words, given a constraint of one of these forms, one has to search (automatically) for a well-founded order satisfying the constraint.

A class of orders which are often used to find solutions for such constraints are recursive path orders (RPO) [12,29,36]. These orders are based on a precedence relation $>_\Sigma$ on the function symbols occurring in terms. A recursive path order between a pair of terms $s = f(s_1, \ldots, s_n)$ and $t = g(t_1, \ldots, t_m)$ is then defined as a statement on the precedence relation (e.g., $f >_\Sigma g$) between the root symbols of s and t, together with an inductive statement on the relation between the arguments $\langle s_1, \ldots, s_n \rangle$ and $\langle t_1, \ldots, t_m \rangle$ of the two terms. In an RPO definition, every function symbol f is associated with a status which indicates if the arguments of terms rooted with f should be compared lexicographically w.r.t. some permutation of the arguments, or as multisets. Allowing symbols to have either lexicographic or multiset status increases the number of possible RPOs exponentially, and allowing the status of a symbol to be defined in terms of a permutation of arguments leads to an additional exponential increase. The search for an RPO becomes even more complex when introducing *argument filters* [1]. This can be done when the desired well-founded order is not required to be closed under contexts, as in the case of the dependency pair constraint (2) and the usable rule constraint (3). With argument filters one may specify for each function symbol f a set of argument positions that should be ignored when comparing terms rooted with f. Combining RPOs with argument filters increases power substantially. As stated in [26]: "the dependency pair method derives much of its power from the ability to use argument filterings to simplify constraints". While the addition of argument filters to RPO is often needed to solve the constraints, it increases the search space by another exponential factor.

This enormous search space is the main reason why dedicated solvers often use restricted forms of RPO. Examples are the lexicographic path order (LPO [29]) where arguments are always compared lexicographically from left to right, or the multiset path order (MPO [12]) where arguments are always compared as multisets. Moreover, even when using restricted classes of RPO, additional incomplete heuristics are often applied to prune the search space when argument filters are integrated. Typically, these choices limit the amount of argument filters to a number that is linear in the arity of the function symbols. For example, for every function symbol f one only allows to either filter away all arguments but one or not to filter away any of f's arguments, cf. [26].

The decision problem for RPO is the following: given a constraint of the forms (1)–(3), does there exist an RPO which satisfies the constraint? This decision problem is NP-complete [31] and it remains NP-complete when restricted to LPO, or to MPO, as well as when regarding RPO in combination with argument filters. In this paper we take a SAT-based approach to solve the RPO decision problem: the constraint is encoded into a propositional formula φ which is satisfiable iff the constraint is satisfied by some RPO. Satisfiability of φ is tested using any SAT solver and the order satisfying the constraint can be reconstructed from a satisfying assignment of φ. However, it is not straightforward to find a "good" SAT encoding, i.e., an encoding which really leads to significant speedups. In particular, one has to develop polynomially sized encodings which are also small in practice.

In [32] the authors address the class of lexicographic path orders and propose an encoding to propositional logic. Based on the use of BDDs, this pioneering work already outperformed dedicated solvers. A significant improvement is described in [10] which presents an efficient SAT-based implementation of LPO which outperforms dedicated solvers by orders of magnitude. This SAT-based approach provides a starting point for the work presented in the current paper.

While RPO is a powerful and fundamental class of orders used for termination proofs of TRSs, there also exist other important classes of orders based on interpretations. The power of these orders is "orthogonal" to RPO, i.e., there exist TRSs where termination can be proved by RPO but not by any of these orders based on interpretations, and vice versa. In related work by other researchers and ourselves, it has been shown that SAT solving is also useful for orders based on interpretations like polynomial orders (possibly with "maximum") [8,17–19], increasing interpretations [45], matrix orders [15], Knuth-Bendix orders [44], and also for variants of semantic labelling [30] and of the size-change principle [4].

The current paper introduces the first SAT-based encoding for full RPO. Our goal is to determine suitable "parameters" (i.e., a suitable precedence, status of the function symbols, and argument filter), such that the resulting order is compatible with the given rewrite system or rather with the given constraint of type (1)–(3). The first two contributions of this paper are

(A) an encoding for the lexicographic comparison w.r.t. permutations and
(B) an encoding for the multiset extension of the base order.

The extension for permutations is of course not obtained by a mere (correct but naive) enumeration of permutations combined with the encoding for the fixed (left to right) order of arguments that is used in LPO. Instead, the idea is to encode the search for a permutation as part of the search for the other parameters of the RPO. The propositional formula resulting from the encoding is satisfiable iff there exists an RPO satisfying the original constraint.

The third contribution of this paper concerns the combined search for an RPO and an argument filter to solve dependency pair constraints (2) and usable rule constraints (3), i.e., we present

(C) an encoding of the combination of RPO with argument filters.

This extension of the RPO encoding is non-trivial as the choice of the argument filter influences the structure of the terms that have to be oriented by the RPO. Again, the encoding is of course not obtained by a mere (correct but naive) enumeration which considers each possible argument filter and builds the SAT encoding for all (exponentially many) corresponding RPO decision problems. Instead, the idea is to encode the search for an argument filter together with the search for the RPO.

The fourth contribution is concerned with the encoding of usable rules which occur in usable rule constraints of the form (3), i.e., we give

(D) an encoding of the set of usable rules.

Here, the challenge stems from the mutual dependencies between the various parts of the decision problem: A particular choice of an argument filter modifies the constraints on the desired RPO (less subterms are compared by the RPO). But at the same time, the choice of the argument filter may render certain rules non-usable, which relaxes the constraints on the order (less rules need to be oriented). With less rules to orient, it

is often possible to solve a decision problem which would not have been solvable when using a weaker argument filter (which filters less subterms), etc. So to summarize, we encode the following aspects of an RPO decision problem to SAT:

- the search for the precedence on function symbols.
- the search for the status of the function symbols that decides whether arguments are compared as multisets or lexicographically. In the latter case, the status also determines the permutation of arguments that is used for the lexicographic comparison.
- the search for the argument filter, where the argument filter influences both the resulting RPO and the set of rules that have to be oriented.

To obtain an efficient SAT-based search procedure, the overall SAT encoding must encode all these aspects *at once*. This overall encoding captures all-in-one the synergy between precedences, statuses, and argument filters. In this way, the task of finding a solution for all of the search problems simultaneously is delegated to the underlying search strategy of the SAT solver. To define the encoding, we first translate the definitions of the underlying termination techniques into "inductive" form. Then these inductive definitions are transformed into recursive generators for corresponding propositional formulas. To this end, we use the well-known idea of *reification*, i.e., of introducing fresh Boolean variables which represent different parts of the definition. The challenge is to design an encoding of all the different aspects of the overall search problem which leads to "small" SAT problems that can be solved efficiently in practice. Often this means keeping track of which Boolean variables correspond to which parts of a definition and reusing these variables instead of re-encoding recurring parts of the definition.

Reification is widely applied in many different types of encoding (or modeling) problems. For some recent examples see: Jefferson in [28] to model sophisticated propagators for constraint programming problems, Feydy *et al.* in [16] to model difference constraints and to design finite domain propagators, Lescuyer and Conchon in [37] to provide proofs by reflection in the Coq theorem prover, Gotlieb in [25] to model a verification problem (where it is illustrated that constraint programming can compete with other techniques based on SAT and SMT solvers), Bofill *et al.* in [7] to model Max-SAT problems and to encode them as pseudo-Boolean constraints, and there are many more.

We start with the necessary preliminaries on term rewriting in Section 2. In Section 3, we first give a definition of RPO specifically tailored towards the encoding. Afterwards we show how to encode rule removal constraints like (1), including both multiset comparisons and lexicographic comparisons w.r.t. permutations (Contributions (A) and (B)). Required notions about dependency pairs are recapitulated in Section 4. In this section, we also introduce and discuss our novel encoding for dependency pair constraints like (2), where the order is a combination of RPO with some argument filter (Contribution (C)). After recalling the concept of usable rules, in Section 5 we show how to extend our encoding to usable rule constraints like (3) where the set of constraints depends on the argument filter (Contribution (D)). For all Contributions (A)–(D), and for all three forms of Constraints (1)–(3), throughout Sections 3–5 we prove that our encoding introduces a propositional formula of polynomial size (more precisely, of size $\mathcal{O}(n^3)$, where n is the size of the constraint). In Section 6 we describe the implementation of our results in the termination prover AProVE [23]. It turns out that the combination of a termination prover with a SAT solver yields a surprisingly

fast implementation of RPO. We provide extensive experimental evidence indicating speedups in orders of magnitude as well as an impressive increase in the power of automated termination analysis. Finally, we conclude in Section 7.

This paper extends the preliminary work presented in [9] and [40] substantially.[1] It contains the details for the complete SAT encoding of RPO, including the use of RPO for constraints like (1) and (3), and a formal analysis of the size of the encoding. We also provide a more extensive experimental evaluation including experiments where RPO is integrated with other termination techniques in order to obtain an approach as powerful as possible.

2 Preliminaries

In this section we recapitulate basic notions of term rewriting. For further details on term rewriting we refer to [3], for example.

A signature Σ is a finite set of function symbols. A *term* over Σ is either a variable from the set $\mathcal{V} = \{x, y, \dots\}$, or it is a function application $f(t_1, \dots, t_n)$ where f is some n-ary symbol in Σ and $t_1, \dots, t_n$ are terms. The *root* of $t = f(t_1, \dots, t_n)$ is $root(t) = f$. The set of all terms over Σ and $\mathcal{V}$ is denoted by $\mathcal{T}(\Sigma, \mathcal{V})$. Finally, for any term $t \in \mathcal{T}(\Sigma, \mathcal{V})$, let $\mathcal{V}(t)$ be the set of all variables from $\mathcal{V}$ occurring in t, i.e., $\mathcal{V}(x) = \{x\}$ for $x \in \mathcal{V}$ and $\mathcal{V}(f(t_1, \dots, t_n)) = \bigcup_{1 \leqslant i \leqslant n} \mathcal{V}(t_i)$.

A *substitution* is a function $\delta : \mathcal{V} \to \mathcal{T}(\Sigma, \mathcal{V})$. Substitutions are homomorphically extended to mappings from terms to terms by applying them to all variables occurring in the input term. Instead of $\delta(t)$ we often write $t\delta$. A *context* is a term C with exactly one hole ($\Box$) in it. Then $C[t]$ is the term which is obtained by replacing $\Box$ in C by t.

Now we define TRSs and introduce the notion of the rewrite relation. A *term rewrite system* $\mathcal{R}$ is a finite set of rules $\ell \to r$ with $\ell, r \in \mathcal{T}(\Sigma, \mathcal{V})$, $\mathcal{V}(r) \subseteq \mathcal{V}(\ell)$, and $\ell \notin \mathcal{V}$. The *rewrite relation* for $\mathcal{R}$ is denoted $\to_{\mathcal{R}}$: for $s, t \in \mathcal{T}(\Sigma, \mathcal{V})$ we have $s \to_{\mathcal{R}} t$ iff $s = C[\ell\delta]$ and $t = C[r\delta]$ for some rule $\ell \to r \in \mathcal{R}$, some substitution δ, and come context C. A term t_0 is *terminating* for $\mathcal{R}$ iff there is no infinite sequence $t_0 \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} \dots$ A TRS $\mathcal{R}$ is *terminating* iff $\to_{\mathcal{R}}$ is terminating for all terms.

Most techniques to prove termination are based on well-founded orders. A *rewrite order* is a relation that is transitive, stable (closed under substitutions), and monotonic (closed under contexts). A *reduction order* is a well-founded rewrite order and a *reduction quasi-order* is a reflexive rewrite order. An *order pair* is a pair $(\succsim, \succ)$ where $\succ$ is well founded and $\succsim$ and $\succ$ are *compatible*, i.e., $\succsim \circ \succ \subseteq \succ$ or $\succ \circ \succsim \subseteq \succ$. A *reduction pair* is an order pair $(\succsim, \succ)$ where $\succ$ is stable and $\succsim$ is a reduction quasi-order. A reduction pair $(\succsim, \succ)$ is *monotonic* iff $\succ$ is a reduction order.

The most classic termination criterion for TRSs states that a TRS $\mathcal{R}$ is terminating iff there is a reduction order $\succ$ which orients all rules of $\mathcal{R}$ (i.e., $\ell \succ r$ for all $\ell \to r \in \mathcal{R}$ or, in set notation, $\mathcal{R} \subseteq \succ$) [38]. This can easily be refined to the following "rule removal" technique (which goes back to [5, 20, 34] and which can also be used for relative termination proofs). It relaxes this criterion by just requiring a weak decrease of all rules w.r.t. $\succsim$. Then all strictly decreasing rules can be removed. Formally, $\mathcal{R}$ is terminating if $\mathcal{R} \setminus \succ$ is terminating where $(\succsim, \succ)$ is a monotonic reduction pair satisfying $\mathcal{R} \subseteq \succsim$. Whereas the former condition (i.e., termination of $\mathcal{R} \setminus \succ$) is ensured

[1] In [9] we introduced an encoding for argument filters (as in Contribution (C)) in connection with LPO. Independently, a similar encoding was presented in [43, 44] where it is applied to Knuth-Bendix orders.

by recursively applying termination techniques on the smaller TRS $\mathcal{R} \setminus \succ$, the latter condition (i.e., $\mathcal{R} \subseteq \succsim$) can be seen as a set of constraints that have to be satisfied by some monotonic reduction pair. Since each application of the rule removal technique should at least delete one rule to really obtain a smaller TRS, this requirement is exactly the rule removal constraint (1) from the introduction.

Example 1 As an example, consider the TRS with the two rules $\mathsf{plus}(\mathsf{0}, y) \to y$ *and* $\mathsf{plus}(\mathsf{s}(x), y) \to \mathsf{plus}(x, \mathsf{s}(y))$ *for adding numbers. Here, the rule removal constraint (1) is the following formula:*

$$\begin{array}{lll} & \mathsf{plus}(\mathsf{0}, y) \succsim y \ \wedge \ \mathsf{plus}(\mathsf{s}(x), y) \succsim \mathsf{plus}(x, \mathsf{s}(y)) & \wedge \\ (& \mathsf{plus}(\mathsf{0}, y) \succ y \ \vee \ \mathsf{plus}(\mathsf{s}(x), y) \succ \mathsf{plus}(x, \mathsf{s}(y)) &) \end{array}$$

If one finds a monotonic reduction pair $(\succsim, \succ)$ *where the first rule is strictly decreasing and the second is just weakly decreasing, then one can remove the first rule and just has to prove termination of the remaining TRS with the second rule. On the other hand, if one finds a monotonic reduction pair where both rules are strictly decreasing, then one can remove both rules of the TRS, i.e., then one has already proved termination.*

3 Recursive Path Orders and their SAT Encodings

As outlined in the introduction, the fundamental problem for automated termination analysis of TRSs is the search for suitable reduction orders (and quasi-orders) satisfying constraints of the forms (1)–(3). Three prominent classes of reduction orders are the lexicographic path order (LPO [29]), the multiset path order (MPO [12]), and the recursive path order (RPO [36]), which combines the lexicographic and multiset path order allowing also permutations in the lexicographic comparison. In Section 3.1 we recapitulate their definitions using a formalization of "multiset extension" that is particularly suitable for the SAT encoding later on. The SAT encoding of RPO is then presented in Sections 3.2–3.9. We formally analyze the size of the encoding in Section 3.10. Finally, Section 3.11 briefly summarizes the contributions of Section 3.

3.1 The Recursive Path Order

When comparing two terms $f(s_1, \ldots, s_n)$ and $g(t_1, \ldots, t_m)$ by RPO, one possibility is to compare the tuples $\langle s_1, \ldots, s_n \rangle$ and $\langle t_1, \ldots, t_m \rangle$ of arguments. To this end, one has to extend an order over terms to an order over tuples of terms. RPO features two such extensions, the lexicographic and the multiset extension. We often denote tuples of terms as $\bar{s} = \langle s_1, \ldots, s_n \rangle$, etc.

Definition 2 (Lexicographic Extension) Let $(\succsim, \succ)$ be an order pair on terms and let the equivalence relation $\sim$ be defined as $\succsim \cap \precsim$, i.e., $s \sim t$ holds iff both $s \succsim t$ and $t \succsim s$. The *lexicographic extensions* of $\sim$, $\succ$, and $\succsim$ are defined on tuples of terms:

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

So for tuples of numbers $\bar{s} = \langle 3, 3, 4, 0\rangle$ and $\bar{t} = \langle 3, 2, 5, 6, 7\rangle$, 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 Definition 3 we formalize the multiset extension by a *multiset cover* which is a pair of mappings (γ, ε). Intuitively, γ expresses which elements in $\bar{s}$ 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. So, $\gamma(j) = i$ means that s_i covers t_j, and we have $\varepsilon(i) = \mathit{true}$ iff whatever s_i covers is equal to s_i. This formalization facilitates encodings to propositional logic afterwards.

Definition 3 (Multiset Cover) Let $\bar{s} = \langle s_1, \ldots, s_n\rangle$ and $\bar{t} = \langle t_1, \ldots, t_m\rangle$ be tuples of terms. A *multiset cover* (γ, ε) is a pair of mappings $\gamma : \{1, \ldots, m\} \to \{1, \ldots, n\}$ and $\varepsilon : \{1, \ldots, n\} \to \{\mathit{true}, \mathit{false}\}$ such that for each $1 \leqslant i \leqslant n$, if $\varepsilon(i) = \mathit{true}$ (indicating equality) then $\{j \mid \gamma(j) = i\}$ is a singleton set.

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.

Now we can define the multiset extension.

Definition 4 (Multiset Extension) Let $(\succsim, \succ)$ be an order pair on terms and let $\sim = \succsim \cap \precsim$. The *multiset extensions* of $\succsim$, $\succ$, and $\sim$ are defined on tuples of terms:

(ms$_1$) $\langle s_1, \ldots, s_n\rangle \succsim^{mul} \langle t_1, \ldots, t_m\rangle$ iff there exists a multiset cover (γ, ε) such that for all i, j: $\gamma(j) = i$ implies that either $\varepsilon(i) = \mathit{true}$ and $s_i \sim t_j$, or $\varepsilon(i) = \mathit{false}$ and $s_i \succ t_j$.

(ms$_2$) $\langle s_1, \ldots, s_n\rangle \succ^{mul} \langle t_1, \ldots, t_m\rangle$ iff $\langle s_1, \ldots, s_n\rangle \succsim^{mul} \langle t_1, \ldots, t_m\rangle$ and for some i, $\varepsilon(i) = \mathit{false}$, i.e., some s_i is not used for equality but rather replaced by zero or more smaller arguments t_j.

(ms$_3$) $\langle s_1, \ldots, s_n\rangle \sim^{mul} \langle t_1, \ldots, t_m\rangle$ iff $\langle s_1, \ldots, s_n\rangle \succsim^{mul} \langle t_1, \ldots, t_m\rangle$ and for all i, $\varepsilon(i) = \mathit{true}$, i.e., all s_i are used to cover some t_j by equality. Note that this implies $n = m$.

Before we can define RPO formally, we need to explain the two missing ingredients of RPO. First, there is a so-called *precedence* which is used to compare function symbols. Formally, a precedence is an order pair $(\geqslant_\Sigma, >_\Sigma)$ on the set of function symbols Σ where $\approx_\Sigma = \geqslant_\Sigma \cap \leqslant_\Sigma$ is the equivalence relation of symbols having the same precedence.

Second, each RPO has 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 w.r.t. some permutation μ. Here, a permutation is a bijection on the set $\{1, \ldots, n\}$ for some natural number n.

Definition 5 (Status Function) A *status function* σ maps each symbol $f \in \Sigma$ of arity n either to the symbol `mul` or to a permutation μ^f on $\{1, \ldots, n\}$.

Now we can define the RPO.

Definition 6 (Recursive Path Order) For a precedence $(\succcurlyeq_\Sigma, >_\Sigma)$ and status function σ we define the relations $\succ_{rpo}$ and $\sim_{rpo}$ on terms. Moreover, we define $\succsim_{rpo} = \succ_{rpo} \cup \sim_{rpo}$. We use the notation $\bar{s} = \langle s_1, \ldots, s_n \rangle$ and $\bar{t} = \langle t_1, \ldots, t_m \rangle$.

(gt) $s \succ_{rpo} t$ iff $s = f(\bar{s})$ and one of **($\mathbf{gt}_1$)** or **($\mathbf{gt}_2$)** holds:
- **($\mathbf{gt}_1$)** $s_i \succsim_{rpo} t$ or $s_i \sim_{rpo} t$ **for some** $1 \leqslant i \leqslant n$;
- **($\mathbf{gt}_2$)** $t = g(\bar{t})$ and $s \succ_{rpo} t_j$ **for all** $1 \leqslant j \leqslant m$ and either:
  (i) $f >_\Sigma g$ or (ii) $f \approx_\Sigma g$ and $\bar{s} \succ_{rpo}^{f,g} \bar{t}$;

(eq) $s \sim_{rpo} t$ iff one of **($\mathbf{eq}_1$)** or **($\mathbf{eq}_2$)** holds:
- **($\mathbf{eq}_1$)** $s = t$;
- **($\mathbf{eq}_2$)** $s = f(\bar{s})$, $t = g(\bar{t})$, $f \approx_\Sigma g$, and $\bar{s} \sim_{rpo}^{f,g} \bar{t}$.

Here, $\succ_{rpo}^{f,g}$ and $\sim_{rpo}^{f,g}$ are tuple extensions of $\succ_{rpo}$ and $\sim_{rpo}$. They are defined in terms of the lexicographic and multiset extensions of $\succ_{rpo}$ and $\sim_{rpo}$ as determined by the status of f and g:

($\mathbf{gt}^*$) $\langle s_1, \ldots, s_n \rangle \succ_{rpo}^{f,g} \langle t_1, \ldots, t_m \rangle$ iff one of **($\mathbf{gt}_1^*$)** or **($\mathbf{gt}_2^*$)** holds:
- **($\mathbf{gt}_1^*$)** σ maps f and g to permutations μ^f and μ^g; and $\mu^f\langle s_1, \ldots, s_n \rangle \succ_{rpo}^{lex} \mu^g\langle t_1, \ldots, t_m \rangle$;
- **($\mathbf{gt}_2^*$)** σ maps f and g to `mul`; and $\langle s_1, \ldots, s_n \rangle \succ_{rpo}^{mul} \langle t_1, \ldots, t_m \rangle$;

($\mathbf{eq}^*$) $\langle s_1, \ldots, s_n \rangle \sim_{rpo}^{f,g} \langle t_1, \ldots, t_m \rangle$ iff one of **($\mathbf{eq}_1^*$)** or **($\mathbf{eq}_2^*$)** holds:
- **($\mathbf{eq}_1^*$)** σ maps f and g to μ^f and μ^g; and $\mu^f\langle s_1, \ldots, s_n \rangle \sim_{rpo}^{lex} \mu^g\langle t_1, \ldots, t_m \rangle$;
- **($\mathbf{eq}_2^*$)** σ maps f and g to `mul`; and $\langle s_1, \ldots, s_n \rangle \sim_{rpo}^{mul} \langle t_1, \ldots, t_m \rangle$.

Definition 6 can be specialized to capture other path orders by taking specific forms of status functions: LPO, when σ maps all symbols to the identity permutation; MPO, when σ maps all symbols to `mul`.

Example 7 Consider the following three TRSs for adding numbers, where (a) is the TRS from Example 1.

$$(a)\ \left\{ \mathsf{plus}(\mathsf{0}, y) \to y\ ,\quad \mathsf{plus}(\mathsf{s}(x), y) \to \mathsf{plus}(x, \mathsf{s}(y)) \right\}$$
$$(b)\ \left\{ \mathsf{plus}(x, \mathsf{0}) \to x\ ,\quad \mathsf{plus}(x, \mathsf{s}(y)) \to \mathsf{s}(\mathsf{plus}(y, x)) \right\}$$
$$(c)\ \left\{ \mathsf{plus}(x, \mathsf{0}) \to x\ ,\quad \mathsf{plus}(x, \mathsf{s}(y)) \to \mathsf{plus}(\mathsf{s}(x), y) \right\}$$

The rule removal constraint (1) for TRS (a) is solvable by an LPO with the precedence $\mathsf{plus} >_\Sigma \mathsf{s}$*, but not by any MPO (regardless of the precedence). The rule removal constraint (1) for TRS (b) is solvable by an MPO taking the precedence* $\mathsf{plus} >_\Sigma \mathsf{s}$*, but not by any LPO since the second rule swaps x and y. The rule removal constraint (1) for TRS (c) cannot be solved by any LPO nor by any MPO. However, it is solvable by an RPO taking the precedence* $\mathsf{plus} >_\Sigma \mathsf{s}$ *and the permutation* $\sigma(\mathsf{plus}) = (2,\ 1)$ *so that lexicographic comparison proceeds from right to left instead of from left to right. For all three TRSs, the reduction order identified renders both of the rules in the TRS strictly decreasing and thus constitutes a proof of termination.*

As explained in the introduction, our goal is to solve RPO decision problems (i.e., to find out whether there exist a precedence and a status function such that the resulting RPO satisfies constraints like the rule removal constraint (1)). There are two variants of the problem: the "strict-" and the "quasi-RPO decision problem" depending on whether $f \approx_\Sigma g$ can hold for $f \neq g$. Both decision problems are decidable and NP-complete [31].

In the remainder of this section we focus on the implementation of a decision procedure for the RPO decision problem by encoding it into a corresponding SAT problem. This enables us to encode rule removal constraints (1) to SAT and to solve them by existing SAT solvers.

3.2 The Backbone of the Encoding

We now introduce an encoding τ which maps "atomic" constraints of the form $s \succ_{rpo} t$, $s \succsim_{rpo} t$, or $s \sim_{rpo} t$ to propositional statements about the precedence and the status of the symbols in the terms s and t. A satisfying assignment for the encoding of a constraint indicates a precedence and a status function such that the constraint holds. We say that an encoding τ is *correct* if for every atomic constraint c, the satisfying assignments of $\tau(c)$ correspond precisely to those precedences and statuses where the constraint c holds (i.e., to the solutions of c). The encoding is defined by a series of equations introduced in the following subsections. To encode more complex constraints like (1) that consist of conjunctions (or disjunctions) of atomic subconstraints, one first has to encode the atomic subconstraints and then take the conjunction (or disjunction) of the resulting propositional formulas. We assume standard binding order for Boolean connectives, i.e., $\wedge$ binds stronger than $\vee$ which in turn binds stronger than $\rightarrow$ and $\leftrightarrow$.

In this subsection, we encode the main structure of Definition 6, whereas Section 3.3 shows how to encode the constraints on the precedence $>_\Sigma$. To deal with the lexicographic extension of RPO, Section 3.4 shows how to encode permutations and then Section 3.5 explains how to encode lexicographic comparisons w.r.t. permutations. Section 3.7 is devoted to the encoding of multiset comparisons and afterwards, Section 3.8 shows how to encode the combination of both lexicographic and multiset comparisons. Finally, Section 3.9 puts all results of the previous subsections together and presents the overall encoding of the RPO decision problem for rule removal constraints (1).

Our encoding closely follows the formalization of RPO in Definition 6. In the following, the reader should distinguish between the *"definition"* (Definition 6) and its *"encoding"* which is being introduced here.

Equation (4) is the top-level encoding of the definition of $\succ_{rpo}$ in **(gt)**, which states that *"$s \succ_{rpo} t$ iff $s = f(\bar{s})$ and one of* $(\mathbf{gt}_1)$ *or* $(\mathbf{gt}_2)$ *holds"*. The equation expresses that the encoding τ of $f(\bar{s}) \succ_{rpo} t$ is a disjunction of the encodings τ_1 and τ_2 which correspond to Cases $(\mathbf{gt}_1)$ and $(\mathbf{gt}_2)$ in the definition.

$$\tau(f(\bar{s}) \succ_{rpo} t) = \tau_1(f(\bar{s}) \succ_{rpo} t) \vee \tau_2(f(\bar{s}) \succ_{rpo} t) \tag{4}$$

Equation (5) is the encoding of Case $(\mathbf{gt}_1)$ in the definition. It is expressed as a disjunction for the n components of $\bar{s}$, corresponding to the **"for some"** statement in the definition.

$$\tau_1(f(\bar{s}) \succ_{rpo} t) = \bigvee_{1 \leqslant i \leqslant n} (\tau(s_i \succ_{rpo} t) \vee \tau(s_i \sim_{rpo} t)) \tag{5}$$

Equation (6) is the encoding of Case $\mathbf{(gt_2)}$ in the definition. It is expressed as a conjunction with two parts. The first part is a conjunction for the m components of $\bar{t}$, corresponding to the **"for all"** statement in the definition. The second part is a disjunction which imposes precedence constraints $f >_\Sigma g$ resp. $f \approx_\Sigma g$ on the symbols f and g corresponding to the subcases (i) and (ii) of Case $\mathbf{(gt_2)}$ in the definition. The encodings of the precedence constraints will be described in Section 3.3 and the encoding of the extension $\succ_{rpo}^{f,g}$ will be described in Sections 3.5–3.8.

$$\tau_2(f(\bar{s}) \succ_{rpo} g(\bar{t})) = \underbrace{\bigwedge_{1 \leqslant j \leqslant m} \tau(f(\bar{s}) \succ_{rpo} t_j)}_{\text{for all arguments of } t} \wedge \underbrace{\left(\begin{array}{l}\tau(f >_\Sigma g) \vee \\ (\tau(f \approx_\Sigma g) \wedge \tau(\bar{s} \succ_{rpo}^{f,g} \bar{t}))\end{array}\right)}_{\text{subcase (i) or (ii)}} \tag{6}$$

Equations (7) and (8) encode the constraint $s \sim_{rpo} t$ as specified in Cases $\mathbf{(eq_1)}$ and $\mathbf{(eq_2)}$ of the definition. Equation (8) imposes the precedence constraint $f \approx_\Sigma g$ on the symbols f and g. Again, the encodings of the precedence constraint and of the extension $\sim_{rpo}^{f,g}$ will be described in Section 3.3 resp. in Sections 3.5-3.8.

$$\tau(s \sim_{rpo} s) = \mathit{true} \tag{7}$$

$$\tau(f(\bar{s}) \sim_{rpo} g(\bar{t})) = \tau(f \approx_\Sigma g) \wedge \tau(\bar{s} \sim_{rpo}^{f,g} \bar{t}) \tag{8}$$

All "missing" cases (e.g., $\tau(x \succ_{rpo} t)$ for variables x) are defined to be *false*.

The correctness of this backbone of the encoding can formally proved by a straightforward structural induction over the constraint being encoded where one assumes correctness of the encodings for precedence constraints and for lexicographic and multiset comparisons of tuples. These are subsequently introduced in the following subsections.

3.3 Encoding Precedence Constraints

Precedence constraints of the form $f >_\Sigma g$ and $f \approx_\Sigma g$ impose a partial order on the symbols. Their encoding is defined as in [10] where it is termed a "symbol-based encoding". Let $|\Sigma| = c$ and assume $c > 1$ (otherwise all precedence constraints are trivial). The symbols in Σ are interpreted as indices in a total order extending the imposed partial order taking values from the set $\{0, \ldots, c-1\}$. Each symbol $f \in \Sigma$ is viewed as a binary number $f = \langle f_k, \ldots, f_1 \rangle$ where f_k is the most significant bit and $k = \lceil \log_2 c \rceil$. The binary value of $\langle f_k, \ldots, f_1 \rangle$ represents the position of f in the partial order. Possibly, $\langle f_k, \ldots, f_1 \rangle = \langle g_k, \ldots, g_1 \rangle$ for $f \neq g$, if a partial order imposes no order between f and g, or if a (non-strict) partial order imposes $f \approx_\Sigma g$. Statements about precedences are interpreted as constraints on indices and they are encoded in k-bit arithmetic:

$$\tau(\langle f_k, \ldots, f_1 \rangle \approx_\Sigma \langle g_k, \ldots, g_1 \rangle) = \bigwedge_{1 \leqslant i \leqslant k} (f_i \leftrightarrow g_i) \tag{9}$$

$$\tau(\langle f_k, \ldots, f_1 \rangle >_\Sigma \langle g_k, \ldots, g_1 \rangle) = \tag{10}$$
$$\begin{cases} (f_k \wedge \neg g_k), & \text{if } k = 1 \\ (f_k \wedge \neg g_k) \vee ((f_k \leftrightarrow g_k) \wedge \tau(\langle f_{k-1}, \ldots, f_1 \rangle >_\Sigma \langle g_{k-1}, \ldots, g_1 \rangle)), & \text{if } k > 1 \end{cases}$$

In [10] the authors provide a formal justification for this symbol-based encoding. Alternative encodings for precedence constraints include the atom-based approach described in [32], a symbol-based encoding using unary representation for integers as applied

in [11], and the order encoding described in [41]. Experiments and fine-tuning indicate that the binary symbol-based encoding given in Equations (9) and (10) is best suited for our application.

3.4 Encoding Permutations

To encode lexicographic comparisons modulo permutations, we associate with each symbol $f \in \Sigma$ of arity n a permutation μ^f on $\{1, \ldots, n\}$. We represent μ^f as an $n \times n$ Boolean matrix where each element $\mu^f_{i,k} = \mathit{true}$ iff $\mu^f(i) = k$. To model a permutation, the matrix μ^f must contain exactly one *true* value in each row and column. This restriction is encoded as a conjunction of cardinality constraints stating that each row and column sums up to 1. Hence, our encoding includes the following formula $\tau(\mu^f)$, where "*true*" is identified with 1 and "*false*" is identified with 0.

$$\tau(\mu^f) \;=\; \bigwedge_{1 \leqslant i \leqslant n} \tau\left(\sum_{k=1}^{n} \mu^f_{i,k} = 1\right) \quad \wedge \quad \bigwedge_{1 \leqslant k \leqslant n} \tau\left(\sum_{i=1}^{n} \mu^f_{i,k} = 1\right) \tag{11}$$

There are a variety of alternatives described in the literature for encoding cardinality constraints to SAT. After experimenting with several of these encodings (including a straightforward quadratic approach), we decided to adopt the BDD-based encoding described in [14] which is linear in the number of variables when summing up to 1.

3.5 Encoding Lexicographic Extensions w.r.t. Permutations

Now we consider the encodings $\tau(\bar{s} \sim^{f,g}_{rpo} \bar{t})$ and $\tau(\bar{s} \succ^{f,g}_{rpo} \bar{t})$ for the tuple extensions. These are required to complete the definitions described in the Equations (6) and (8). At the moment, we restrict ourselves to the case where the arguments of f and g are compared lexicographically modulo the permutations μ^f and μ^g. To indicate this clearly, we write $\sim^{f,g}_{lex}$ and $\succ^{f,g}_{lex}$ instead of $\sim^{f,g}_{rpo}$ and $\succ^{f,g}_{rpo}$. The encoding corresponds to the second part of Definition 6, i.e., to Cases $\mathbf{(gt^*)}$ and $\mathbf{(eq^*)}$.

The encodings of $\bar{s} \sim^{f,g}_{lex} \bar{t}$ and $\bar{s} \succ^{f,g}_{lex} \bar{t}$ build on the following idea. If the two permutations μ^f and μ^g were given, then we could instead encode the constraint $\mu^f(\bar{s}) \sim^{lex}_{rpo} \mu^g(\bar{t})$ and $\mu^f(\bar{s}) \succ^{lex}_{rpo} \mu^g(\bar{t})$ for the permuted tuples $\mu^f(\bar{s})$ and $\mu^g(\bar{t})$. However, these permutations are not given. The objective is to find them (through the encoding) such that the constraints hold.

As in the definition, let $\bar{s} = \langle s_1, \ldots, s_n \rangle$ and $\bar{t} = \langle t_1, \ldots, t_m \rangle$. Equation (12) encodes $\bar{s} \sim^{f,g}_{lex} \bar{t}$ following Case $\mathbf{(eq^*_1)}$ of the definition where for $n = m$ we encode that for all k, the arguments s_i and t_j permuted to the k-th position by μ^f and μ^g are equivalent. Note that here, $(n = m)$ is a Boolean value which can be determined at encoding time.

$$\tau(\bar{s} \sim^{f,g}_{lex} \bar{t}) \;=\; (n = m) \;\wedge \bigwedge_{1 \leqslant i,j,k \leqslant \min(n,m)} \left(\mu^f_{i,k} \wedge \mu^g_{j,k} \rightarrow \tau(s_i \sim_{rpo} t_j)\right) \tag{12}$$

To formalize the encoding of $\bar{s} \succ^{f,g}_{lex} \bar{t}$, we consider the subterms s_i and t_j which are mapped by μ^f and μ^g, respectively, to each of the positions $1 \leqslant k \leqslant \min(n, m)$. To this end we introduce constraints of the form $\bar{s} \succ^{f,g,k}_{lex} \bar{t}$ which express lexicographic

comparisons starting from a position k. The encoding is initialized by $k = 1$, i.e., we start with the first position.

We then consider three cases for each position k: (1) $n < k$, i.e., there remain no positions in $\bar{s}$, and the encoding is *false*; (2) $n \geqslant k > m$, i.e, there remain positions in $\bar{s}$ but no positions in $\bar{t}$, and the encoding is *true*; and (3) both $n \geqslant k$ and $m \geqslant k$, and the encoding considers all $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$ to capture the case where $\mu^f(i) = k = \mu^g(j)$.

$$\tau(\bar{s} \succ_{lex}^{f,g} \bar{t}) \;=\; \tau(\bar{s} \succ_{lex}^{f,g,1} \bar{t})$$

$$\tau(\bar{s} \succ_{lex}^{f,g,k} \bar{t}) \;=\; \begin{cases} \mathit{false}, & \text{if } n < k \\ \mathit{true}, & \text{if } n \geqslant k > m \\ \tau'(\bar{s} \succ_{lex}^{f,g,k} \bar{t}) & \text{otherwise} \end{cases} \tag{13}$$

where

$$\tau'(\bar{s} \succ_{lex}^{f,g,k} \bar{t}) = \bigwedge_{\substack{1 \leqslant i \leqslant n, \\ 1 \leqslant j \leqslant m}} \left(\mu^f_{i,k} \wedge \mu^g_{j,k} \;\rightarrow\; \begin{array}{l} \tau(s_i \succ_{rpo} t_j) \vee \\ \Big(\tau(s_i \sim_{rpo} t_j) \wedge \tau(\bar{s} \succ_{lex}^{f,g,k+1} \bar{t})\Big) \end{array} \right)$$

Example 8 Consider again the TRS of Example 7(c):

$$\Big\{ \quad \mathsf{plus}(x, \mathsf{0}) \to x \;, \qquad \mathsf{plus}(x, \mathsf{s}(y)) \to \mathsf{plus}(\mathsf{s}(x), y) \qquad \Big\}$$

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

$$\begin{aligned}
&\tau(\langle x, \mathsf{s}(y)\rangle \succ_{lex}^{\mathsf{plus},\mathsf{plus}} \langle \mathsf{s}(x), y\rangle) = \tau(\langle x, \mathsf{s}(y)\rangle \succ_{lex}^{\mathsf{plus},\mathsf{plus},1} \langle \mathsf{s}(x), y\rangle) = \\
&\quad \Big(\mu^{\mathsf{plus}}_{1,1} \wedge \mu^{\mathsf{plus}}_{1,1} \rightarrow \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{plus},\mathsf{plus},2} \langle \mathsf{s}(x), y\rangle))\Big)\Big) \\
&\wedge \Big(\mu^{\mathsf{plus}}_{1,1} \wedge \mu^{\mathsf{plus}}_{2,1} \rightarrow \Big(\tau(x \succ_{rpo} y) \vee (\tau(x \sim_{rpo} y) \wedge \tau(\langle x, \mathsf{s}(y)\rangle \succ_{lex}^{\mathsf{plus},\mathsf{plus},2} \langle \mathsf{s}(x), y\rangle))\Big)\Big) \\
&\wedge \Big(\mu^{\mathsf{plus}}_{2,1} \wedge \mu^{\mathsf{plus}}_{1,1} \rightarrow \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{plus},\mathsf{plus},2} \langle \mathsf{s}(x), y\rangle))\Big)\Big) \\
&\wedge \Big(\mu^{\mathsf{plus}}_{2,1} \wedge \mu^{\mathsf{plus}}_{2,1} \rightarrow \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{plus},\mathsf{plus},2} \langle \mathsf{s}(x), y\rangle))\Big)\Big) \\
&\tau(\langle x, \mathsf{s}(y)\rangle \succ_{lex}^{\mathsf{plus},\mathsf{plus},2} \langle \mathsf{s}(x), y\rangle) = \\
&\quad \Big(\mu^{\mathsf{plus}}_{1,2} \wedge \mu^{\mathsf{plus}}_{1,2} \rightarrow \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{plus},\mathsf{plus},3} \langle \mathsf{s}(x), y\rangle))\Big)\Big) \\
&\wedge \Big(\mu^{\mathsf{plus}}_{1,2} \wedge \mu^{\mathsf{plus}}_{2,2} \rightarrow \Big(\tau(x \succ_{rpo} y) \vee (\tau(x \sim_{rpo} y) \wedge \tau(\langle x, \mathsf{s}(y)\rangle \succ_{lex}^{\mathsf{plus},\mathsf{plus},3} \langle \mathsf{s}(x), y\rangle))\Big)\Big) \\
&\wedge \Big(\mu^{\mathsf{plus}}_{2,2} \wedge \mu^{\mathsf{plus}}_{1,2} \rightarrow \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{plus},\mathsf{plus},3} \langle \mathsf{s}(x), y\rangle))\Big)\Big) \\
&\wedge \Big(\mu^{\mathsf{plus}}_{2,2} \wedge \mu^{\mathsf{plus}}_{2,2} \rightarrow \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{plus},\mathsf{plus},3} \langle \mathsf{s}(x), y\rangle))\Big)\Big) \\
&\tau(\langle x, \mathsf{s}(y)\rangle \succ_{lex}^{\mathsf{plus},\mathsf{plus},3} \langle \mathsf{s}(x), y\rangle) = \mathit{false}
\end{aligned}$$

Observe 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* $\tau(\mathsf{s}(y) \succ_{rpo} y) = \mathit{true}$*. Hence, the above simplifies to* $\tau(\langle x, \mathsf{s}(y)\rangle \succ_{lex}^{\mathsf{plus},\mathsf{plus}} \langle \mathsf{s}(x), y\rangle) = \neg\mu^{\mathsf{plus}}_{1,1}$*. Together with the constraint* $\tau(\mu^f)$ *from Equation (11) which ensures that the variables* $\mu^{\mathsf{plus}}_{i,k}$ *specify a valid permutation* μ^{plus}*, this implies that* $\mu^{\mathsf{plus}}_{1,2}$ *and* $\mu^{\mathsf{plus}}_{2,1}$ *must be true. And indeed, for the permutation* $\mu^{\mathsf{plus}} = (2, 1)$ *the tuple* $\mu^{\mathsf{plus}}\langle x, \mathsf{s}(y)\rangle = \langle \mathsf{s}(y), x\rangle$ *is* $\succ_{rpo}^{lex}$*-greater than the tuple* $\mu^{\mathsf{plus}}\langle \mathsf{s}(x), y\rangle = \langle y, \mathsf{s}(x)\rangle$*.*

That Equations (12) and (13) correctly encode the lexicographic extension follows from Definition 2 and straightforward structural induction.

3.6 Encoding Multiset Covers

To encode multiset comparisons, we associate with each pair of tuples $\bar{s} = \langle s_1, \ldots, s_n \rangle$ and $\bar{t} = \langle t_1, \ldots, t_m \rangle$ a multiset cover (γ, ε). Recall from Definition 3 that γ is a mapping $\gamma : \{1, \ldots, m\} \to \{1, \ldots, n\}$ and ε is a mapping $\varepsilon : \{1, \ldots, n\} \to \{\mathit{true}, \mathit{false}\}$. We represent γ as an $n \times m$ Boolean matrix where each element $\gamma_{i,j} = \mathit{true}$ iff $\gamma(j) = i$, i.e., if s_i covers t_j. We represent ε as a Boolean vector where $\varepsilon_i = \mathit{true}$ indicates that s_i is used for equality.

Following Definition 3, to model a multiset cover, 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 formula $\tau((\gamma, \varepsilon))$. Here, we again identify "*true*" with 1 and "*false*" with 0.

$$\tau((\gamma,\varepsilon)) \;=\; \bigwedge_{1 \leqslant j \leqslant m} \left(\sum_{i=1}^{n} \gamma_{i,j} = 1 \right) \quad \wedge \quad \bigwedge_{1 \leqslant i \leqslant n} \left(\varepsilon_i \to \left(\sum_{j=1}^{m} \gamma_{i,j} = 1 \right) \right) \tag{14}$$

The underlying encoding of cardinality constraints is subject to the same choices as described in Section 3.4.

3.7 Encoding Multiset Extensions

Now we again consider the encodings $\tau(\bar{s} \sim^{f,g}_{rpo} \bar{t})$ and $\tau(\bar{s} \succ^{f,g}_{rpo} \bar{t})$ for tuple extensions, but this time for the case where the arguments of f and g are compared using multiset extensions of the RPO (thus, we write $\sim^{f,g}_{mul}$ and $\succ^{f,g}_{mul}$).

To encode $\sim^{f,g}_{mul}$ and $\succ^{f,g}_{mul}$, we proceed according to Definition 4 where $\bar{s} = \langle s_1, \ldots, s_n \rangle$, $\bar{t} = \langle t_1, \ldots, t_m \rangle$, and where (γ, ε) is the multiset cover associated with $\bar{s}$ and $\bar{t}$. Case $\mathbf{(ms_1)}$ of the definition states that $\bar{s} \succsim^{f,g}_{mul} \bar{t}$ holds iff (γ, ε) satisfies the following: if $\gamma_{i,j}$ and ε_i are *true*, then $s_i \sim_{rpo} t_j$, and else, if $\gamma_{i,j}$ is *true* and ε_i is not, then $s_i \succ_{rpo} t_j$. This leads to Equation (15) in the encoding below. Case $\mathbf{(ms_2)}$ in the definition states that $\bar{s} \succ^{f,g}_{mul} \bar{t}$ holds iff $\bar{s} \succsim^{f,g}_{mul} \bar{t}$ and not all s_i are used for equality. This results in Equation (16) in the encoding. Finally, Case $\mathbf{(ms_3)}$ in the definition states that $\bar{s} \sim^{f,g}_{mul} \bar{t}$ holds iff $\bar{s} \succsim^{f,g}_{mul} \bar{t}$ and all s_i are used for equality. This leads to Equation (17).

$$\tau(\bar{s} \succsim^{f,g}_{mul} \bar{t}) = \tau((\gamma,\varepsilon)) \;\wedge \bigwedge_{\substack{1 \leqslant i \leqslant n,\\ 1 \leqslant j \leqslant m}} \left(\gamma_{i,j} \to \begin{pmatrix} (\;\; \varepsilon_i \to \tau(s_i \sim_{rpo} t_j)) \wedge \\ (\neg\varepsilon_i \to \tau(s_i \succ_{rpo} t_j)) \end{pmatrix} \right) \tag{15}$$

$$\tau(\bar{s} \succ^{f,g}_{mul} \bar{t}) = \tau(\bar{s} \succsim^{f,g}_{mul} \bar{t}) \;\wedge\; \neg \bigwedge_{1 \leqslant i \leqslant n} \varepsilon_i \tag{16}$$

$$\tau(\bar{s} \sim^{f,g}_{mul} \bar{t}) = \tau(\bar{s} \succsim^{f,g}_{mul} \bar{t}) \;\wedge \bigwedge_{1 \leqslant i \leqslant n} \varepsilon_i \tag{17}$$

Example 9 Consider again the TRS of Example 7(b):

$$\left\{ \quad \mathsf{plus}(x, \mathsf{0}) \to x \;, \qquad \mathsf{plus}(x, \mathsf{s}(y)) \to \mathsf{s}(\mathsf{plus}(y, x)) \qquad \right\}$$

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

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

Since $\tau(x \sim_{rpo} y) = \tau(x \succ_{rpo} y) = \tau(x \succ_{rpo} x) = \tau(\mathsf{s}(y) \sim_{rpo} y) = \tau(\mathsf{s}(y) \sim_{rpo} x) = \tau(\mathsf{s}(y) \succ_{rpo} x) = \mathit{false}$ *and* $\tau(x \sim_{rpo} x) = \tau(\mathsf{s}(y) \succ_{rpo} y) = \mathit{true}$*, we can simplify the above formula to* $\tau((\gamma,\varepsilon)) \wedge \neg\gamma_{1,1} \wedge (\neg\gamma_{1,2} \vee \varepsilon_1) \wedge (\neg\gamma_{2,1} \vee \neg\varepsilon_2) \wedge \neg\gamma_{2,2} \wedge (\neg\varepsilon_1 \vee \neg\varepsilon_2)$*. As* $\tau((\gamma,\varepsilon))$ *ensures that the variables* $\gamma_{i,j}$ *and* ε_i *specify a valid multiset cover* (γ,ε)*, this implies that* $\gamma_{2,1}$*,* $\gamma_{1,2}$*,* ε_1*, and* $\neg\varepsilon_2$ *must hold. And indeed, the multiset cover* (γ,ε) *with* $\gamma(1) = 2$*,* $\gamma(2) = 1$*,* $\varepsilon(1) = \mathit{true}$*, and* $\varepsilon(2) = \mathit{false}$ *shows that the tuple* $\langle x, \mathsf{s}(y)\rangle$ *is greater than the tuple* $\langle y, x\rangle$*. The reason is that* $t_1 = y$ *is covered by* $s_2 = \mathsf{s}(y)$ *and (as indicated by* $\varepsilon(2) = \mathit{false}$*) we have* $s_2 \succ_{rpo} t_1$*. Similarly,* $t_2 = x$ *is covered by* $s_1 = x$ *and (as indicated by* $\varepsilon(1) = \mathit{true}$*) we have* $s_1 \sim_{rpo} t_2$*.*

The encodings follow directly from Definitions 3 and 4 and their correctness can be proved by a straightforward structural induction.

3.8 Combining Lexicographic and Multiset Comparisons

We have shown how to encode lexicographic and multiset comparisons. In order to combine $\succ_{lex}^{f,g}$ and $\succ_{mul}^{f,g}$ into $\succ_{rpo}^{f,g}$ as well as $\sim_{lex}^{f,g}$ and $\sim_{mul}^{f,g}$ into $\sim_{rpo}^{f,g}$, each symbol $f \in \Sigma$ is associated with a Boolean flag m_f which indicates whether terms rooted with f are to be compared using multiset extensions ($m_f = \mathit{true}$) or using lexicographic extensions ($m_f = \mathit{false}$). In other words, the flag m_f indicates whether the status function σ used in the RPO definition maps f to `mul` or to μ^f.

$$\tau(\bar{s} \succ_{rpo}^{f,g} \bar{t}) = \Big(m_f \wedge m_g \wedge \tau(\bar{s} \succ_{mul}^{f,g} \bar{t})\Big) \vee \Big(\neg m_f \wedge \neg m_g \wedge \tau(\bar{s} \succ_{lex}^{f,g} \bar{t})\Big) \tag{18}$$

$$\tau(\bar{s} \sim_{rpo}^{f,g} \bar{t}) = \Big(m_f \wedge m_g \wedge \tau(\bar{s} \sim_{mul}^{f,g} \bar{t})\Big) \vee \Big(\neg m_f \wedge \neg m_g \wedge \tau(\bar{s} \sim_{lex}^{f,g} \bar{t})\Big) \tag{19}$$

Similar to Definition 6, the encoding function τ can be specialized to other standard path orders: lexicographic path order with status (LPOS) when m_f is set to *false* for all $f \in \Sigma$; LPO when additionally $\mu_{i,k}^f$ is set to *true* iff $i = k$; MPO when m_f is set to *true* for all $f \in \Sigma$.

3.9 Encoding RPO Constraints

At this point we have defined all necessary formulas to encode inequalities like $s \succ_{rpo} t$ and $s \succsim_{rpo} t$. Therefore, we can now encode the rule removal constraint (1) as

$$\boxed{\bigwedge_{f \in \Sigma} \tau(\mu^f) \wedge \bigwedge_{\ell \to r \in \mathcal{R}} \tau(\ell \succsim_{rpo} r) \wedge \bigvee_{\ell \to r \in \mathcal{R}} \tau(\ell \succ_{rpo} r)} \tag{1$'$}$$

where $\tau(\ell \succsim_{rpo} r)$ is just an abbreviation for $\tau(\ell \succ_{rpo} r) \vee \tau(\ell \sim_{rpo} r)$.

3.10 Size of the Encoding

We conclude this section with an analysis of the size of the propositional encoding. Note that for the sake of readability, we presented our encoding as resulting in arbitrary propositional formulas. As the result of Tseitin's transformation [42] to conjunctive normal form (CNF) has linear size, the bounds obtained in this section carry over to the size of the formulas in CNF.

To analyze the size of our encoding, we first consider the size of the encoding for a single inequality $s \succ_{rpo} t$. We focus on the case where s and t are ground terms. Observe in the definition of the encoding that replacing a non-variable subterm in s (or in t) by a variable results in an encoding which is at most as large as the encoding of $s \succ_{rpo} t$. Let $k = |s| + |t|$ denote the total number of occurrences of function symbols in s and t, let a be the maximal arity of a symbol in s and t, and let $c = |\Sigma|$ denote the cardinality of the underlying signature. In this subsection, we always assume that Σ only contains those function symbols that indeed occur in the constraint under consideration.

We recapitulate the (recursive part of the) backbone of the encoding presented in Equations (4)–(8). Here, $s = f(\bar{s})$ and $t = g(\bar{t})$ are terms with arguments $\bar{s} = \langle s_1, \ldots, s_i, \ldots, s_n \rangle$ and $\bar{t} = \langle t_1, \ldots, t_j, \ldots, t_m \rangle$, and to ease readability, we simplify the notation and write $\succ$ instead of $\succ_{rpo}$ or $\succ_{rpo}^{f,g}$, $\sim$ instead of $\sim_{rpo}$ or $\sim_{rpo}^{f,g}$, $>$ instead of $>_\Sigma$, and $\approx$ instead of $\approx_\Sigma$.

$$\tau(s \succ t) = \bigvee_i \begin{pmatrix} \tau(s_i \succ t) \vee \\ \tau(s_i \sim t) \end{pmatrix} \vee \bigwedge_j \tau(s \succ t_j) \wedge \begin{pmatrix} \tau(f > g) \vee \\ \tau(f \approx g) \wedge \tau(\bar{s} \succ \bar{t}) \end{pmatrix} \qquad (20)$$

$$\tau(s \sim t) = \tau(f \approx g) \ \wedge \ \tau(\bar{s} \sim \bar{t})$$

A naive unfolding of $\tau(s \succ t)$ according to this definition obviously leads to an exponentially sized encoding. To obtain a polynomially sized encoding instead, we use reification, i.e., we introduce new propositional variables to share common subformulas. The approach is similar to the one used in Tseitin's transformation to conjunctive normal form. The basic idea is to view each expression of the form $\tau(e)$ as a (fresh) propositional variable $\chi(e)$, and the equation defining $\tau(e)$ is viewed as a biimplication with $\chi(e)$ on the left-hand side.

For example, the part of Equation (20) that defines $\tau(s \succ t)$ is viewed as the following propositional statement.

$$\chi(s \succ t) \leftrightarrow \underbrace{\bigvee_i \begin{pmatrix} \chi(s_i \succ t) \vee \\ \chi(s_i \sim t) \end{pmatrix}}_{(i)} \vee \underbrace{\bigwedge_j \chi(s \succ t_j)}_{(ii)} \wedge \underbrace{\begin{pmatrix} \chi(f > g) \vee \\ \chi(f \approx g) \wedge \chi(\bar{s} \succ \bar{t}) \end{pmatrix}}_{(iii)} \qquad (21)$$

Each fresh variable $\chi(e)$ on the right-hand side of a biimplication is then defined itself by another biimplication in which it occurs on the left-hand side. Note that each $\chi(e)$ occurs exactly once on the left-hand side of a biimplication.

Consider first the biimplications for defining $\chi(\bar{s} \succ \bar{t})$ which occurs on the right-hand side of Equation (21). These biimplications take a total size of $\mathcal{O}(a + a \cdot n \cdot m)$:

- For the case of lexicographic comparison, Equation (13) contains $\mathcal{O}(n \cdot m)$ variables of the form $\chi(s_i \succ t_j)$ or $\chi(s_i \sim t_j)$. The size of the biimplication defining $\chi(\bar{s} \succ \bar{t})$ is in $\mathcal{O}(a \cdot n \cdot m)$.
- For the case of multiset comparison, we assume that the encoding of a cardinality constraint $\sum_{i=1}^{n} x_i = 1$ is at most quadratic. Then the right-hand side of Equation (14) is of the size $\mathcal{O}(m \cdot n^2 + n \cdot m^2) \subseteq \mathcal{O}(a \cdot n \cdot m)$. Hence, the size of the biimplication introduced by Equation (15) is also in $\mathcal{O}(a \cdot n \cdot m)$. The biimplications introduced by Equations (16) and (17) are both within the size $\mathcal{O}(n) \subseteq \mathcal{O}(a)$.
- Finally, Biimplication (18) for combining lexicographic and multiset comparison has a constant size.

Now we consider Biimplication (21) for $\chi(s \succ t)$. Its size is in $\mathcal{O}(a + \log(c))$. The reason is that part (i) and (ii) are in $\mathcal{O}(a)$, and for part (iii) we have to encode two precedence comparisons, each of size $\log(c)$, as can be seen in (9) and (10).

To summarize, encoding a comparison $s \succ t$ for $s = f(\bar{s})$ and $t = g(\bar{t})$ where f and g have the arities n and m, together with the corresponding lexicographic and multiset extensions for $\bar{s} \succ \bar{t}$ involves a collection of biimplications of a total size

$$\mathcal{O}(a + \log(c) + a \cdot n \cdot m) \tag{22}$$

Here and also later, it cannot be assumed that $\log(c)$ is a constant or that $\log(c)$ can be bounded by the sizes of s and t. The reason is that c is not the number of different symbols that occur in s and t, but it is the size of the signature of the whole rule removal constraint (1). Hence, it is possible that $\log(c)$ grows in the number of rules.

Inspecting the size of the encoding for $\tau(s \sim t)$ gives a similar result, within the same bound.

Now consider the overall encoding of $s \succ t$ including all of the biimplications introduced. In particular, this includes all biimplications of the form $\chi(s' \succ t')$ where s' is an arbitrary subterm of s and t' is an arbitrary subterm of t. With $k = |s| + |t|$, one obtains up to k^2 such biimplications. And since n and m are smaller or equal to a, (22) implies that the total size of the encoding is $\mathcal{O}(k^2 \cdot (\log(c) + a^3))$. If one now uses $a \leqslant k$ one obtains $\mathcal{O}(k^2 \cdot \log(c) + k^5)$. However, the approximation that every subterm has the maximal arity a (by using $a \cdot n \cdot m \leqslant a^3$) is too coarse: a more detailed comparison reveals that the size of the encoding is indeed within $\mathcal{O}(k^2 \cdot (\log(c) + a)) \subseteq \mathcal{O}(k^2 \cdot \log(c) + k^3)$.

To prove the cubic bound formally, let $s_1, \ldots, s_p$ and $t_1, \ldots, t_q$ be all subterms of the terms s and t respectively and let $n_1, \ldots, n_p$ and $m_1, \ldots, m_q$ be the corresponding arities of the root symbols of these subterms. Hence, the tree representation of the term s has $\sum_{1 \leqslant i \leqslant p} n_i$ edges and $|s|$ nodes. It follows that $\sum_{1 \leqslant i \leqslant p} n_i = |s| - 1$ and similarly that $\sum_{1 \leqslant j \leqslant q} m_j = |t| - 1$. The reason is that except for the root, every node in the tree representation has exactly one incoming edge. Now (22) implies that the overall size of the encoding (which also considers the biimplications arising from the encodings of all subterm comparisons) is bounded by:

$$\begin{aligned}
&\mathcal{O}\Big(\sum_{1 \leqslant i \leqslant p} \sum_{1 \leqslant j \leqslant q} \big(a + \log(c) + a \cdot n_i \cdot m_j\big)\Big) \\
&= \mathcal{O}(p \cdot q \cdot (a + \log(c)) + a \cdot \sum_{1 \leqslant i \leqslant p} n_i \cdot \sum_{1 \leqslant j \leqslant q} m_j) \\
&= \mathcal{O}(p \cdot q \cdot (a + \log(c)) + a \cdot (|s| - 1) \cdot (|t| - 1))
\end{aligned}$$

$$\subseteq \mathcal{O}(k^2 \cdot (\log(c) + a))$$
$$\subseteq \mathcal{O}(k^2 \cdot \log(c) + k^3)$$

Thus, encoding a single inequality constraint $s \succ t$ can be done in the size $\mathcal{O}(k^2 \cdot \log(c) + k^3)$ where $k = |s| + |t|$ and $c = |\Sigma|$.

We can now easily conclude that the size of our encoding is cubic, where the size of a constraint is defined to be the sum of all sizes of the terms occurring in the constraint. Here, we again assume that the encoding of a cardinality constraint $\sum_{i=1}^{n} x_i = 1$ is at most quadratic.

Theorem 10 *Let K be the size of the rule removal constraint (1). Then the size of Formula (1′) is in $\mathcal{O}(K^3)$.*

Proof By using the previous results, we can encode all constraints $\tau(\ell \succ_{rpo} r)$ and $\tau(\ell \sim_{rpo} r)$ in $\mathcal{O}(K^3)$ since $c = |\Sigma| \leqslant K$. It remains to analyze the sizes of each formula $\tau(\mu^f)$ from (11) for every $f \in \Sigma$. Since the cardinality constraint can be encoded in a formula of quadratic size, for each f the size of $\tau(\mu^f)$ is at most cubic in the arity of f. Hence, if $a_1, \ldots, a_c$ are the arities of all function symbols then we obtain formulas of a total size of at most $\mathcal{O}(\sum_{i=1}^{c} a_i^3) \subseteq \mathcal{O}((\sum_{i=1}^{c} a_i)^3) \subseteq \mathcal{O}(K^3)$. □

3.11 Summary

We have shown how to encode an RPO decision problem to SAT. For a given pair of terms s and t, the encoding $\tau(s \succ_{rpo} t)$ is a formula which is satisfiable iff there exists an RPO such that $s \succ_{rpo} t$. Given this encoding we can solve rule removal constraints of the following form (1), where we use monotonic reduction pairs based on RPO.

$$\boxed{\bigwedge_{\ell \to r \in \mathcal{R}} \ell \succsim_{rpo} r \quad \wedge \quad \bigvee_{\ell \to r \in \mathcal{R}} \ell \succ_{rpo} r}$$

In this approach all rules of a TRS must be weakly oriented (i.e., with $\succsim_{rpo}$) and those rules that are also strictly decreasing are removed. To identify these rules, one only has to inspect the satisfying variable assignment found by the SAT solver. A rule $\ell \succ r$ is identified as strictly decreasing (and removed) if the corresponding propositional variable $\chi(\ell \succ r)$ is *true* in the satisfying assignment.

With this rule removal approach, one can for example prove termination of all three TRSs in Example 7 automatically. Here, the resulting Constraint (1) is transformed into the Formula (1′) and satisfiability of this propositional formula is easily shown by existing SAT solvers.

4 Dependency Pairs, Argument Filters, and their SAT Encodings

A considerable improvement to the rule removal approach is obtained when considering so-called dependency pairs [1]. For instance, the following example is terminating, but termination cannot be shown using the rule removal approach with RPO. In contrast, its termination is easy to prove by DPs in combination with RPO. For this reason, virtually all current TRS termination provers use dependency pairs.

Example 11 *The following TRS from [1] computes division on natural numbers.*

$$\begin{aligned}
\mathsf{minus}(x, \mathsf{0}) &\to x \\
\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{minus}(x, y) \\
\mathsf{div}(\mathsf{0}, \mathsf{s}(y)) &\to \mathsf{0} \\
\mathsf{div}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{s}(\mathsf{div}(\mathsf{minus}(x, y), \mathsf{s}(y)))
\end{aligned}$$

Within the *DP framework* [1,22,24,26], a major termination technique is the *reduction pair processor* which is analogous to the technique of rule removal. This processor produces dependency pair constraints of the form (2). Although these constraints look very similar to the Constraints (1) of the rule removal technique, there is one major difference: the reduction pair for orienting the constraints does not have to be monotonic anymore. This allows us to use orders which are a combination of an RPO with a preprocessing on terms called *argument filter*. On the one hand, this combination increases the power of termination provers tremendously, but on the other hand it introduces a new challenge for the automation. In this section, we show how this problem can again be solved using a suitable SAT encoding.

In Section 4.1, we briefly describe the basics of the DP framework such as argument filters and the reduction pair processor. To ease readability, we just present a simplified variant of the DP framework which suffices for the contributions of the present paper. Then in Section 4.2 we define an order in the style of Definition 6 which directly describes the order resulting from the combination of RPO with argument filters. This enables us in Section 4.3 to extend our SAT encoding from the previous section to solve the resulting constraints directly (i.e., without an enumeration of argument filters). Moreover, in this way, the dependencies between the RPO and the argument filter can be detected and exploited by the SAT solver in order to prune the search space effectively. We prove that the size of our encoding for a dependency pair constraint (2) is again cubic in the size of the constraint. This may seem surprising as it implies that, asymptotically, there is no additional complexity over the encoding of the simpler rule removal constraint (1). The contributions of Section 4 are summarized in Section 4.4.

4.1 Dependency Pairs

The main idea of the DP method is to build a separate set of rules (the *dependency pairs*) that represent function calls. In addition, the existing rules are still used to evaluate the arguments. Formally, given a TRS $\mathcal{R}$, we first identify the *defined symbols* $\Sigma_D = \{root(\ell) \mid \ell \to r \in \mathcal{R}\}$. For each defined symbol $f \in \Sigma_D$, we extend the signature by a fresh *tuple symbol* $f^\sharp$, where we often write F instead of $f^\sharp$. For $t = f(t_1, \ldots, t_n)$ with $f \in \Sigma_D$, let $t^\sharp$ denote $f^\sharp(t_1, \ldots, t_n)$. Then the set of *dependency pairs* for a TRS $\mathcal{R}$ is

$$DP(\mathcal{R}) = \{\ell^\sharp \to t^\sharp \mid \ell \to r \in \mathcal{R},\ t \text{ is a subterm of } r,\ root(t) \in \Sigma_D\}.$$

Example 12 *Recall the TRS $\mathcal{R}$ from Example 11. The defined symbols of $\mathcal{R}$ are* minus *and* div*, and there are three dependency pairs:*

$$\begin{aligned}
\mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{MINUS}(x, y) \\
\mathsf{DIV}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{MINUS}(x, y) \\
\mathsf{DIV}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{DIV}(\mathsf{minus}(x, y), \mathsf{s}(y))
\end{aligned} \tag{23}$$

The main result underlying the dependency pair method states that a TRS $\mathcal{R}$ is terminating iff there is no infinite $\mathcal{R}$-*chain* of its dependency pairs $DP(\mathcal{R})$ [1], i.e., there is no infinite sequence of function calls in any possible evaluation. An $\mathcal{R}$-chain of dependency pairs $\mathcal{P}$ (also called $(\mathcal{P},\mathcal{R})$-chain) is a reduction of the form

$$s_1 \to_{\mathcal{P}} t_1 \to_{\mathcal{R}}^* s_2 \to_{\mathcal{P}} t_2 \to_{\mathcal{R}}^* s_3 \dots \tag{24}$$

Here, the $\mathcal{P}$-steps model the infinite sequence of function calls and may only be applied at the root. Moreover, all t_i must be terminating for $\mathcal{R}$.[2] The intermediate $\mathcal{R}$-steps are required to evaluate the arguments between two outer function calls. For example, after an application of the dependency pair (23) from Example 12, one first has to evaluate the minus-term before another application of (23) can take place. So for example, we have the following $(DP(\mathcal{R}),\mathcal{R})$-chain:

$$\begin{array}{l}\mathsf{DIV}(\mathsf{s}(\mathsf{s}(0)),\mathsf{s}(0)) \to_{DP(\mathcal{R})} \mathsf{DIV}(\mathsf{minus}(\mathsf{s}(0),0),\mathsf{s}(0)) \to_{\mathcal{R}}^* \\ \mathsf{DIV}(\mathsf{s}(0),\mathsf{s}(0)) \quad \to_{DP(\mathcal{R})} \mathsf{DIV}(\mathsf{minus}(0,0),\mathsf{s}(0)) \quad \to_{\mathcal{R}}^* \\ \mathsf{DIV}(0,\mathsf{s}(0))\end{array}$$

Now instead of trying to prove termination of a TRS $\mathcal{R}$, in the DP framework one considers so-called *DP problems* $(\mathcal{P},\mathcal{R})$ consisting of a set of dependency pairs $\mathcal{P}$ and a TRS $\mathcal{R}$ where the task is to prove absence of infinite $(\mathcal{P},\mathcal{R})$-chains. In analogy to the rule removal technique where one starts with the initial TRS $\mathcal{R}$ and simplifies it until there are no rules left, in the DP framework one starts with the initial DP problem $(DP(\mathcal{R}),\mathcal{R})$ and simplifies it using so-called *processors* that remove dependency pairs. These simplifications are repeated until there are no dependency pairs left.

The processor that corresponds to the rule removal technique is the so-called *reduction pair processor* [1,22,24,26]. For a DP problem $(\mathcal{P},\mathcal{R})$, it generates constraints which ensure that every root reduction with $\mathcal{P}$ is weakly or strictly decreasing and that every reduction with $\mathcal{R}$ is weakly decreasing for some reduction pair $(\succsim,\succ)$. Then according to the definition of chains in (24), one can remove all strictly decreasing DPs from $\mathcal{P}$, since they can only occur finitely often in $\mathcal{P}$-chains. Here, monotonicity of $\succ$ is not required, since in chains, $\mathcal{P}$-steps can only be applied at the root. To ensure that reductions with $\mathcal{R}$ are weakly decreasing one just has to demand that all rules of $\mathcal{R}$ are weakly decreasing, since $\succsim$ is a reduction quasi-order. Similarly, to guarantee that all root reductions with $\mathcal{P}$ are at least weakly decreasing, it suffices to require a weak decrease for all DPs in $\mathcal{P}$.

Theorem 13 (Reduction Pair Processor) *Let $(\succsim,\succ)$ be a reduction pair such that $\mathcal{P}\cup\mathcal{R}\subseteq\succsim$. If there is no infinite $(\mathcal{P}\setminus\succ,\mathcal{R})$-chain, then there is no infinite $(\mathcal{P},\mathcal{R})$-chain.*

As in the rule removal technique, at least one DP in $\mathcal{P}$ should be strictly decreasing to ensure that $\mathcal{P}\setminus\succ\,\subset\mathcal{P}$, i.e., to ensure that the reduction pair processor results in a simpler DP problem than the original one. Hence, to apply the reduction pair processor, one has to search for a reduction pair $(\succsim,\succ)$ satisfying the dependency pair constraint (2).

A typical choice for a reduction pair is to use an order like RPO in combination with an *argument filter* [1]. An argument filter is a mapping which specifies parts of terms which can be ignored when comparing terms (we adopt the notation of [33]).

[2] Such chains are usually called *minimal* [22]. Since we only consider minimal chains in this paper, we simply refer to them as "chains".

Definition 14 (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 $1 \leqslant i_1 < \cdots < i_p \leqslant n$. An argument filter with $\pi(f) = i$ is called *collapsing on* f. We write "$i \trianglelefteqslant \pi(f)$" to indicate that $\pi(f)$ is a list containing i or that $\pi(f) = i$. 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 iff $\pi(s) \succ \pi(t)$.

Arts and Giesl showed 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 the form $(\succsim^\pi_{rpo}, \succ^\pi_{rpo})$ to prove termination of TRSs like Example 11 where the direct application of orders like RPO fails.

Example 15 For the TRS of Example 11, according to Theorem 13 we search for a reduction pair solving the following inequality constraints where at least one of Constraints (25)–(27) should be oriented strictly. By Theorem 13, all dependency pairs corresponding to strictly decreasing inequalities can be removed.

$$\begin{aligned}
\mathsf{minus}(x, \mathsf{0}) &\succsim x \\
\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) &\succsim \mathsf{minus}(x, y) \\
\mathsf{div}(\mathsf{0}, \mathsf{s}(y)) &\succsim \mathsf{0} \\
\mathsf{div}(\mathsf{s}(x), \mathsf{s}(y)) &\succsim \mathsf{s}(\mathsf{div}(\mathsf{minus}(x, y), \mathsf{s}(y))) \\
\mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y)) &\ {}_{(}\!\succsim_{)}\ \mathsf{MINUS}(x, y) && (25) \\
\mathsf{DIV}(\mathsf{s}(x), \mathsf{s}(y)) &\ {}_{(}\!\succsim_{)}\ \mathsf{MINUS}(x, y) && (26) \\
\mathsf{DIV}(\mathsf{s}(x), \mathsf{s}(y)) &\ {}_{(}\!\succsim_{)}\ \mathsf{DIV}(\mathsf{minus}(x, y), \mathsf{s}(y)) && (27)
\end{aligned}$$

To solve the inequalities we may take $(\succsim^\pi_{rpo}, \succ^\pi_{rpo})$ *where* $\pi(\mathsf{minus}) = \pi(\mathsf{div}) = 1$, $\pi(\mathsf{s}) = \pi(\mathsf{MINUS}) = \pi(\mathsf{DIV}) = [1]$, $\pi(\mathsf{0}) = [\,]$, *and where* $\succsim_{rpo}$ *and* $\succ_{rpo}$ *are induced by the precedence* $\mathsf{DIV} >_\Sigma \mathsf{MINUS}$. *Thus, the constraints after applying* π *are the following:*

$$\begin{aligned}
x &\succsim x & \mathsf{MINUS}(\mathsf{s}(x)) &\ {}_{(}\!\succsim_{)}\ \mathsf{MINUS}(x) \\
\mathsf{s}(x) &\succsim x & \mathsf{DIV}(\mathsf{s}(x)) &\ {}_{(}\!\succsim_{)}\ \mathsf{MINUS}(x) \\
\mathsf{0} &\succsim \mathsf{0} & \mathsf{DIV}(\mathsf{s}(x)) &\ {}_{(}\!\succsim_{)}\ \mathsf{DIV}(x) \\
\mathsf{s}(x) &\succsim \mathsf{s}(x)
\end{aligned}$$

Thus, after filtering, the strict inequalities (25)–(27) *hold, i.e., all three dependency pairs are strictly decreasing with the chosen RPO. Hence, they are removed by the reduction pair processor. This results in the new DP problem* $(\varnothing, \mathcal{R})$ *which proves termination of the TRS.*

Note that while argument filters are very powerful in the context of the DP framework, they also present a severe bottleneck for the automation, as the search space for argument filters is enormous (exponential in the number of function symbols and in the arities of the function symbols).

4.2 Recursive Path Order Combined with Argument Filters

Consider first a naive brute force approach for solving the dependency pair constraint (2) using an RPO with argument filter. For any given argument filter π we could generate the formula

$$\boxed{\bigwedge_{s\to t\in\mathcal{P}\cup\mathcal{R}} \pi(s) \succsim_{rpo} \pi(t) \quad\wedge\quad \bigvee_{s\to t\in\mathcal{P}} \pi(s) \succ_{rpo} \pi(t)} \tag{2'}$$

If any of these formulas is satisfiable, then termination is proved.

The constraints "$\pi(s) \succsim_{rpo} \pi(t)$" and "$\pi(s) \succ_{rpo} \pi(t)$" can be encoded as described in Section 3. Then SAT solving can search for an RPO satisfying (2′) for the given argument filter π.

However, this approach is hopelessly inefficient, since it would call the SAT solver for each of the exponentially many argument filters. Even if one considers the less naive algorithms for enumerating argument filters from [24] and [26], for many examples the SAT solver would be called exponentially often.

We will show instead how to encode the argument filters into a propositional formula and to delegate the search for an argument filter to the SAT solver. In this way, the SAT solver is only called once with an encoding of Formula (2′) where π is not given, but left open. This is an advantage, since the filter, the precedence, and the status strongly influence each other. Now the SAT solver can search for an argument filter, a precedence, and a status at the same time.

So our goal is to encode constraints like "$s \succ^{\pi}_{rpo} t$" (or "$s \succsim^{\pi}_{rpo} t$") into a propositional formula such that every model of the formula corresponds to a concrete filter π, a precedence $(\geqslant_\Sigma, >_\Sigma)$, and a status σ which satisfy "$s \succ^{\pi}_{rpo} t$" (or "$s \succsim^{\pi}_{rpo} t$"). The definition of RPO with an argument filter π is straightforward: first apply the argument filter π to both terms s and t and then use Definition 6. Nevertheless, we provide an explicit definition from which the encoding presented in Section 4.3 is derived. The essential difference to Definition 6 is that all cases are refined to consider the effect of the argument filter π.

Definition 16 (RPO modulo π) For an argument filter π, a precedence $(\geqslant_\Sigma, >_\Sigma)$, and a status function σ, we define the relations $\succ^{\pi}_{rpo}$ and $\sim^{\pi}_{rpo}$ on terms. Moreover, we define $\succsim^{\pi}_{rpo} = \succ^{\pi}_{rpo} \cup \sim^{\pi}_{rpo}$. Again, we use the notation $\bar{s} = \langle s_1, \ldots, s_n\rangle$ and $\bar{t} = \langle t_1, \ldots, t_m\rangle$. Moreover, "$\pi(f) = [\ldots]$" means that π is not collapsing on f.

(gt) $s \succ^{\pi}_{rpo} t$ iff $s = f(\bar{s})$ and one of **(gt$_1$)** or **(gt$_2$)** holds:
- **(gt$_1$)** (a) $\pi(f) = i$ and $s_i \succ^{\pi}_{rpo} t$; or
  (b) $\pi(f) = [i_1, \ldots, i_p]$ and **for some** $i \in [i_1, \ldots, i_p]$, $s_i \succ^{\pi}_{rpo} t$ or $s_i \sim^{\pi}_{rpo} t$;
- **(gt$_2$)** $t = g(\bar{t})$ and
  (a) $\pi(g) = j$ and $s \succ^{\pi}_{rpo} t_j$; or
  (b) $\pi(f) = [\ldots]$, $\pi(g) = [j_1, \ldots, j_q]$, $s \succ^{\pi}_{rpo} t_j$ **for all** $j \in [j_1, \ldots, j_q]$, and either (i) $f >_\Sigma g$ or (ii) $f \approx_\Sigma g$ and $\bar{s} \succ^{f,g,\pi}_{rpo} \bar{t}$;

(eq) $s \sim^{\pi}_{rpo} t$ iff one of **(eq$_1$)** or **(eq$_2$)** holds:
- **(eq$_1$)** (a) $s = t$; or
  (b) $s = f(\bar{s})$ and $\pi(f) = i$ and $s_i \sim^{\pi}_{rpo} t$; or
  (c) $t = g(\bar{t})$ and $\pi(g) = j$ and $s \sim^{\pi}_{rpo} t_j$;
- **(eq$_2$)** $s = f(\bar{s})$, $t = g(\bar{t})$, $\pi(f) = [\ldots]$, $\pi(g) = [\ldots]$, $f \approx_\Sigma g$, and $\bar{s} \sim^{f,g,\pi}_{rpo} \bar{t}$.

Here, $\succ_{rpo}^{f,g,\pi}$ and $\sim_{rpo}^{f,g,\pi}$ are the tuple extensions of $\succ_{rpo}^{\pi}$ and $\sim_{rpo}^{\pi}$ defined as follows. Let $\pi(f) = [i_1, \ldots, i_p]$ and $\pi(g) = [j_1, \ldots, j_q]$.

(gt*) $\langle s_1, \ldots, s_n\rangle \succ_{rpo}^{f,g,\pi} \langle t_1, \ldots, t_m\rangle$ iff one of **(gt$_1^*$)** or **(gt$_2^*$)** holds:
- **(gt$_1^*$)** σ maps f and g to permutations μ^f and μ^g; and $\mu^f\langle s_{i_1}, \ldots, s_{i_p}\rangle \succ_{rpo}^{\pi\, lex} \mu^g\langle t_{j_1}, \ldots, t_{j_q}\rangle$;
- **(gt$_2^*$)** σ maps f and g to `mul`; and $\langle s_{i_1}, \ldots, s_{i_p}\rangle \succ_{rpo}^{\pi\, mul} \langle t_{j_1}, \ldots, t_{j_q}\rangle$;

(eq*) $\langle s_1, \ldots, s_n\rangle \sim_{rpo}^{f,g} \langle t_1, \ldots, t_m\rangle$ iff one of **(eq$_1^*$)** or **(eq$_2^*$)** holds:
- **(eq$_1^*$)** σ maps f and g to μ^f and μ^g; and $\mu^f\langle s_{i_1}, \ldots, s_{i_p}\rangle \sim_{rpo}^{\pi\, lex} \mu^g\langle t_{j_1}, \ldots, t_{j_q}\rangle$;
- **(eq$_2^*$)** σ maps f and g to `mul`; and $\langle s_{i_1}, \ldots, s_{i_p}\rangle \sim_{rpo}^{\pi\, mul} \langle t_{j_1}, \ldots, t_{j_q}\rangle$.

It follows directly from Definitions 6, 14, and 16 that for all terms s and t we have $s \succ_{rpo}^{\pi} t$ iff $\pi(s) \succ_{rpo} \pi(t)$ and $s \succsim_{rpo}^{\pi} t$ iff $\pi(s) \succsim_{rpo} \pi(t)$.

4.3 Encoding RPO modulo Argument Filter

Now we encode the RPO decision problem for the dependency pair constraint (2). Similar to Section 3, we start with encoding "atomic" constraints like $s \succ_{rpo}^{\pi} t$, $s \succsim_{rpo}^{\pi} t$, or $s \sim_{rpo}^{\pi} t$. So the question is whether there exist an argument filter π, a precedence $(\succcurlyeq_\Sigma, >_\Sigma)$, and a status σ such that $s \succ_{rpo}^{\pi} t$, $s \succsim_{rpo}^{\pi} t$, or $s \sim_{rpo}^{\pi} t$ holds. Our aim is to encode these decision problems as constraints on π, $(\succcurlyeq_\Sigma, >_\Sigma)$, and σ, similar to the encoding of $s \succ_{rpo} t$ in Section 3. The difference is that now we also have constraints on the argument filter π.

Encoding Argument Filters: Following Definition 14 we represent π by Boolean vectors π^f where each element π_i^f is *true* iff $i \in \pi(f)$ and by Boolean flags π_{list}^f indicating whether π maps f to a list, i.e., whether π is not collapsing on f. To model an argument filter π, the encoding introduces the following constraint indicating that for each symbol f, π maps f to a list of argument positions or to a single position. Here, let n be the arity of f.

$$\tau(\pi^f) \;=\; \pi_{list}^f \quad \vee \quad \left(\sum_{i=1}^{n} \pi_i^f = 1\right) \tag{28}$$

Encoding the Backbone: The encoding follows the formalization of RPO modulo π given in Definition 16. It is similar to the encoding of Equations (4)–(8) from Section 3. The difference is that now each reference to a subterm must be "wrapped" by the question: "has this subterm been filtered away by π?" Equations (4′)–(6′) adapt (4)–(6) and (7a')–(7c') adapt (7). Here, (7a') treats the case of terms that are already equal before applying the filter and (7b') and (7c') handle the case where one of the terms is a variable x. Finally, Equation (8′) adapts (8). Similar to the encoding of Section 3, all "missing" cases are defined to be *false*.

$$\tau(f(\bar{s}) \succ_{rpo}^{\pi} t) = \tau_1(f(\bar{s}) \succ_{rpo}^{\pi} t) \;\vee\; \tau_2(f(\bar{s}) \succ_{rpo}^{\pi} t) \tag{4′}$$

$$\tau_1(f(\bar{s}) \succ_{rpo}^{\pi} t) = \bigvee_{1 \leqslant i \leqslant n} \Big(\pi_i^f \wedge \Big(\tau(s_i \succ_{rpo}^{\pi} t) \vee (\pi_{list}^f \wedge \tau(s_i \sim_{rpo}^{\pi} t))\Big)\Big) \tag{5′}$$

$$\begin{aligned}\tau_2(f(\bar{s}) \succ_{rpo}^{\pi} g(\bar{t})) = \bigwedge_{1 \leqslant j \leqslant m} \Big(\pi_j^g \to \tau(f(\bar{s}) \succ_{rpo}^{\pi} t_j)\Big) \quad \wedge \\ \Big(\pi_{list}^g \to \Big(\pi_{list}^f \wedge \Big(\tau(f >_\Sigma g) \vee (\tau(f \approx_\Sigma g) \wedge \tau(\bar{s} \succ_{rpo}^{f,g,\pi} \bar{t}))\Big)\Big)\Big)\end{aligned} \tag{6′}$$

$$\tau(s \sim^{\pi}_{rpo} s) = \mathit{true} \tag{7a'}$$

$$\tau(f(\bar{s}) \sim^{\pi}_{rpo} x) = \neg\pi^{f}_{list} \wedge \bigwedge_{1 \leqslant i \leqslant n} \left(\pi^{f}_{i} \rightarrow \tau(s_i \sim^{\pi}_{rpo} x)\right) \quad \text{for variables } x \tag{7b'}$$

$$\tau(x \sim^{\pi}_{rpo} g(\bar{t})) = \neg\pi^{g}_{list} \wedge \bigwedge_{1 \leqslant j \leqslant m} \left(\pi^{g}_{j} \rightarrow \tau(x \sim^{\pi}_{rpo} t_j)\right) \quad \text{for variables } x \tag{7c'}$$

$$\begin{aligned}\tau(f(\bar{s}) \sim^{\pi}_{rpo} g(\bar{t})) = {} & \left(\neg\pi^{f}_{list} \rightarrow \bigwedge_{1 \leqslant i \leqslant n} \left(\pi^{f}_{i} \rightarrow \tau(s_i \sim^{\pi}_{rpo} g(\bar{t}))\right)\right) \quad \wedge \\ & \left((\pi^{f}_{list} \wedge \neg\pi^{g}_{list}) \rightarrow \bigwedge_{1 \leqslant j \leqslant m} \left(\pi^{g}_{j} \rightarrow \tau(f(\bar{s}) \sim^{\pi}_{rpo} t_j)\right)\right) \quad \wedge \\ & \left((\pi^{f}_{list} \wedge \pi^{g}_{list}) \rightarrow \left((f \approx_{\Sigma} g) \wedge \tau(\bar{s} \sim^{f,g,\pi}_{rpo} \bar{t})\right)\right)\end{aligned} \tag{8'}$$

Encoding Permutations modulo π: Let π be an argument filter, f be an n-ary symbol, and assume that $\pi(f) = [i_1, \ldots, i_p]$. We restrict permutations to consider only the non-filtered positions of f. Thus, a filtered permutation $\mu^{f,\pi}$ is a bijective mapping from $\pi(f)$ to $\{1, \ldots, p\}$. We represent a filtered permutation by an $n \times n$ Boolean matrix $\mu^{f,\pi}$ with elements $\mu^{f,\pi}_{i,k}$. To model a filtered permutation, the matrix $\mu^{f,\pi}$ must satisfy the constraints specified in Equation (11′) which generalizes Equation (11) from Section 3.

$$\tau(\mu^{f,\pi}) \;=\; \tau^{a}(\mu^{f,\pi}) \;\wedge\; \tau^{b}(\mu^{f,\pi}) \;\wedge\; \tau^{c}(\mu^{f,\pi}) \;\wedge\; \tau^{d}(\mu^{f,\pi}) \tag{11'}$$

where

$\tau^{a}(\mu^{f,\pi}) = \bigwedge_{1 \leqslant k \leqslant n} \left(\sum_{i=1}^{n} \mu^{f,\pi}_{i,k} \leqslant 1\right)$ encodes that $\mu^{f,\pi}$ is injective, i.e., at the k-th position, at most one of the elements $\mu^{f,\pi}_{i,k}$ may be *true*, for $1 \leqslant i \leqslant n$.

$\tau^{b}(\mu^{f,\pi}) = \bigwedge_{1 \leqslant i \leqslant n} \left(\neg\pi^{f}_{i} \rightarrow \bigwedge_{k=1}^{n} \neg\mu^{f,\pi}_{i,k}\right)$ encodes that if the i-th argument is filtered, then it is not in the domain of $\mu^{f,\pi}$, i.e, $\pi^{f}_{i} = \mathit{false}$ implies $\mu^{f,\pi}_{i,k} = \mathit{false}$, for $1 \leqslant k \leqslant n$.

$\tau^{c}(\mu^{f,\pi}) = \bigwedge_{1 \leqslant i \leqslant n} \left(\pi^{f}_{i} \rightarrow \left(\sum_{k=1}^{n} \mu^{f,\pi}_{i,k} = 1\right)\right)$ encodes that if the i-th argument is not filtered, then it is in the domain of $\mu^{f,\pi}$, i.e, exactly one $\mu^{f,\pi}_{i,k}$ is *true*, for $1 \leqslant k \leqslant n$.

$\tau^{d}(\mu^{f,\pi}) = \bigwedge_{2 \leqslant k \leqslant n} \left(\bigvee_{1 \leqslant i \leqslant n} \mu^{f,\pi}_{i,k} \rightarrow \bigvee_{1 \leqslant i \leqslant n} \mu^{f,\pi}_{i,k-1}\right)$ encodes that the range of $\mu^{f,\pi}$ is $\{1, \ldots, p\}$. If an argument is mapped to position k, then an argument is mapped to position $k - 1$.

Encoding Lexicographic Comparisons w.r.t. Permutations: Now we consider the encoding of $\bar{s} \sim^{f,g,\pi}_{rpo} \bar{t}$ and $\bar{s} \succ^{f,g,\pi}_{rpo} \bar{t}$ for the tuple extensions modulo π. We first consider the case where the arguments of f and g are compared lexicographically modulo the filtered permutations $\mu^{f,\pi}$ and $\mu^{g,\pi}$. We use the notation $\sim^{f,g,\pi}_{lex}$ and $\succ^{f,g,\pi}_{lex}$. The encoding corresponds to Cases **(gt*)** and **(eq*)** of Definition 16. For the equality constraint, this involves adapting Equation (12) of Section 3 to consider the argument filter π.

First consider the condition "$n = m$" in Equation (12). In the presence of an argument filter π, the symbols f and g may have different arities, yet have the same number of non-filtered argument positions. Part (a) of Equation (12′) encodes that f and g have the same number of non-filtered positions. Part (b) of the equation corresponds to the remainder of Equation (12) with π added.

$$\tau(\bar{s} \sim_{lex}^{f,g,\pi} \bar{t}) = \underbrace{\left(\sum_{i=1}^{n} \pi_i^f = \sum_{j=1}^{m} \pi_j^g\right)}_{\text{(a)}} \wedge \underbrace{\bigwedge_{\substack{1 \leqslant i \leqslant n,\, 1 \leqslant j \leqslant m,\\ 1 \leqslant k \leqslant \min(n,\, m)}} \left(\mu_{i,k}^{f,\pi} \wedge \mu_{j,k}^{g,\pi} \rightarrow \tau(s_i \sim_{rpo}^{\pi} t_j)\right)}_{\text{(b)}} \quad (12')$$

Next we adapt Equation (13) of Section 3 to define the encoding of $\succ_{lex}^{f,g,\pi}$. To this end, we consider the subterms s_i and t_j which are mapped by $\mu^{f,\pi}$ and $\mu^{g,\pi}$, respectively, to positions $1 \leqslant k \leqslant \min(n, m)$ and introduce constraints of the form $\bar{s} \succ_{lex}^{f,g,\pi,k} \bar{t}$ which express lexicographic comparisons starting from a position k. As before, the encoding is initialized by $k = 1$, i.e., we start with the first position.

We then consider three cases for each position k: (1) $n < k$, i.e., there remain (syntactically) no positions in $\bar{s}$, and the encoding is *false*; (2) $n \geqslant k > m$, i.e, there remain (syntactically) positions in $\bar{s}$ but no positions in $\bar{t}$, and we need to encode that some argument of f is actually considered at the k-th position; and (3) both $n \geqslant k$ and $m \geqslant k$, and the encoding considers all $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$ to capture the case where $\mu^{f,\pi}(i) = k = \mu^{g,\pi}(j)$. Note that in this case, when due to the argument filter no arguments are considered at the k-th position, the implications in $\tau'(\bar{s} \succ_{lex}^{f,g,\pi,k} \bar{t})$ are all trivially satisfied.

$$\begin{aligned}
&\tau(\bar{s} \succ_{lex}^{f,g,\pi} \bar{t}) \;=\; \tau(\bar{s} \succ_{lex}^{f,g,\pi,1} \bar{t})\\
&\tau(\bar{s} \succ_{lex}^{f,g,\pi,k} \bar{t}) \;=\; \begin{cases} \mathit{false}, & \text{if } n < k \\ \bigvee\limits_{1 \leqslant i \leqslant n} \mu_{i,k}^{f,\pi}, & \text{if } n \geqslant k > m \\ \tau'(\bar{s} \succ_{lex}^{f,g,\pi,k} \bar{t}) & \text{otherwise} \end{cases} \qquad (13')\\
&\textit{where}\\
&\tau'(\bar{s} \succ_{lex}^{f,g,\pi,k} \bar{t}) = \bigwedge_{\substack{1 \leqslant i \leqslant n,\\ 1 \leqslant j \leqslant m}} \left(\mu_{i,k}^{f,\pi} \wedge \mu_{j,k}^{g,\pi} \;\rightarrow\; \begin{array}{l} \tau(s_i \succ_{rpo}^{\pi} t_j) \vee \\ \left(\tau(s_i \sim_{rpo}^{\pi} t_j) \wedge \tau(\bar{s} \succ_{lex}^{f,g,\pi,k+1} \bar{t})\right) \end{array}\right)
\end{aligned}$$

Encoding Multiset Covers modulo π: To encode multiset comparisons, we associate with each pair of terms $f(\bar{s}) = f(s_1, \ldots, s_n)$ and $g(\bar{t}) = g(t_1, \ldots, t_m)$ a filtered multiset cover $(\gamma^{\pi}, \varepsilon^{\pi})$. Here, a filtered multiset cover is just like a multiset cover except that it may only refer to non-filtered positions. We represent γ^{π} by an $n \times m$ Boolean matrix where each element $\gamma_{i,j} = \mathit{true}$ iff i and j are not filtered according to $\pi(f)$ and $\pi(g)$, respectively, and $\gamma^{\pi}(j) = i$, i.e., if s_i covers t_j. We represent ε^{π} as a Boolean vector where $\varepsilon_i = \mathit{true}$ indicates that s_i is not filtered according to $\pi(f)$ and is used for equality. To model a filtered multiset cover, γ^{π} and ε^{π} must satisfy the constraints specified in Equation (14′) which generalizes Equation (14) from Section 3.

$$\tau((\gamma^{\pi}, \varepsilon^{\pi})) \;=\; \bigwedge_{f \,\in\, \Sigma} \left(\tau^a((\gamma^{\pi}, \varepsilon^{\pi})) \;\wedge\; \tau^b((\gamma^{\pi}, \varepsilon^{\pi})) \;\wedge\; \tau^c((\gamma^{\pi}, \varepsilon^{\pi})) \;\wedge\; \tau^d((\gamma^{\pi}, \varepsilon^{\pi}))\right) \quad (14')$$

where

$\tau^a((\gamma^\pi,\varepsilon^\pi)) = \bigwedge_{1\leqslant j\leqslant m}\left(\pi_j^g \to \left(\sum_{i=1}^{n}\gamma_{i,j}=1\right)\right)$ encodes that if the j-th argument of g is not filtered away (i.e., π_j^g is *true*), then there is exactly one argument of f that covers it.

$\tau^b((\gamma^\pi,\varepsilon^\pi)) = \bigwedge_{1\leqslant i\leqslant n}\left(\neg\pi_i^f \to \bigwedge_{1\leqslant j\leqslant m}\neg\gamma_{i,j}\right)$ encodes that if the i-th argument of f is filtered away (i.e., π_i^f is *false*), then it cannot cover any arguments of g.

$\tau^c((\gamma^\pi,\varepsilon^\pi)) = \bigwedge_{1\leqslant j\leqslant m}\left(\neg\pi_j^g \to \bigwedge_{1\leqslant i\leqslant n}\neg\gamma_{i,j}\right)$ encodes that if the j-th argument of g is filtered away (i.e., π_j^g is *false*), then there is no argument of f that covers it.

$\tau^d((\gamma^\pi,\varepsilon^\pi)) = \bigwedge_{1\leqslant i\leqslant n}\left(\varepsilon_i \to \left(\sum_{j=1}^{m}\gamma_{i,j}=1\right)\right)$ encodes that if the i-th argument is used for equality, then there is exactly one argument that is covered by it.

Encoding Multiset Comparisons: Now we define $\tau(\bar{s} \succsim^{f,g,\pi}_{mul} \bar{t})$ analogously to Equation (15) of Section 3. For the encoding of $\succ^{f,g,\pi}_{mul}$ and $\sim^{f,g,\pi}_{mul}$, we restrict Equations (16) and (17) of Section 3 to the arguments that are not filtered away.

$$\tau(\bar{s} \succsim^{f,g,\pi}_{mul} \bar{t}) = \tau((\gamma^\pi,\varepsilon^\pi)) \;\wedge \bigwedge_{\substack{1\leqslant i\leqslant n,\\ 1\leqslant j\leqslant m}} \left(\gamma_{i,j} \to \begin{pmatrix} \varepsilon_i \to \tau(s_i \sim^\pi_{rpo} t_j)) \wedge \\ (\neg\varepsilon_i \to \tau(s_i \succ^\pi_{rpo} t_j)) \end{pmatrix}\right) \tag{15'}$$

$$\tau(\bar{s} \succ^{f,g,\pi}_{mul} \bar{t}) = \tau(\bar{s} \succsim^{f,g,\pi}_{mul} \bar{t}) \wedge \neg \bigwedge_{1\leqslant i\leqslant n} (\pi_i^f \to \varepsilon_i) \tag{16'}$$

$$\tau(\bar{s} \sim^{f,g,\pi}_{mul} \bar{t}) = \tau(\bar{s} \succsim^{f,g,\pi}_{mul} \bar{t}) \wedge \bigwedge_{1\leqslant i\leqslant n} (\pi_i^f \to \varepsilon_i) \tag{17'}$$

Encoding RPO Constraints modulo π: Finally, for the combination of lexicographic and multiset comparisons, we simply change Equations (18) and (19) of Section 3 to use $\succ^{f,g,\pi}_{mul}$ instead of $\succ^{f,g}_{mul}$ etc.

$$\tau(\bar{s} \succ^{f,g,\pi}_{rpo} \bar{t}) = \left(m_f \wedge m_g \wedge \tau(\bar{s} \succ^{f,g,\pi}_{mul} \bar{t})\right) \vee \left(\neg m_f \wedge \neg m_g \wedge \tau(\bar{s} \succ^{f,g,\pi}_{lex} \bar{t})\right) \tag{18'}$$

$$\tau(\bar{s} \sim^{f,g,\pi}_{rpo} \bar{t}) = \left(m_f \wedge m_g \wedge \tau(\bar{s} \sim^{f,g,\pi}_{mul} \bar{t})\right) \vee \left(\neg m_f \wedge \neg m_g \wedge \tau(\bar{s} \sim^{f,g,\pi}_{lex} \bar{t})\right) \tag{19'}$$

Now we are ready to introduce a polynomially sized encoding of the dependency pair constraint (2).

$$\boxed{\bigwedge_{f\in\Sigma}\left(\tau(\mu^{f,\pi}) \wedge \tau(\pi^f)\right) \;\wedge \bigwedge_{\ell\to r\in\mathcal{P}\cup\mathcal{R}} \tau(\ell \succsim^\pi_{rpo} r) \;\wedge \bigvee_{\ell\to r\in\mathcal{P}} \tau(\ell \succ^\pi_{rpo} r)} \tag{2''}$$

where $\tau(\ell \succsim^\pi_{rpo} r)$ is just an abbreviation for $\tau(\ell \succ^\pi_{rpo} r) \vee \tau(\ell \sim^\pi_{rpo} r)$. Satisfiability of this formula indicates an argument filter, a precedence, and a status for each symbol which together prove that one can remove all strictly decreasing dependency pairs.

Example 17 Consider the first arguments $\mathsf{s}(x)$ *and* $\mathsf{minus}(x,y)$ *of* DIV *in the left- and right-hand sides of the dependency pair (23) in Example 12, i.e.,* $\mathsf{DIV}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{DIV}(\mathsf{minus}(x,y),\mathsf{s}(y))$*. Using the above encoding, after simplification of conjunctions, disjunctions, and implications with true and false, and using the side conditions for permutation, argument filter, and multiset cover we obtain:*

$$
\begin{aligned}
&\tau(\mathsf{s}(x) \succ^{\pi}_{rpo} \mathsf{minus}(x,y)) \\
&= \tau_1(\mathsf{s}(x) \succ^{\pi}_{rpo} \mathsf{minus}(x,y)) \vee \tau_2(\mathsf{s}(x) \succ^{\pi}_{rpo} \mathsf{minus}(x,y)) \\
&= \pi^{\mathsf{s}}_1 \wedge \Big(\underbrace{\tau(x \succ^{\pi}_{rpo} \mathsf{minus}(x,y))}_{false} \vee \Big(\pi^{\mathsf{s}}_{list} \wedge \underbrace{\tau(x \sim^{\pi}_{rpo} \mathsf{minus}(x,y))}_{\neg\pi^{\mathsf{minus}}_{list} \wedge \neg\pi^{\mathsf{minus}}_{2}} \Big) \Big) \vee \\
&\quad \Big(\pi^{\mathsf{minus}}_1 \to \underbrace{\tau(\mathsf{s}(x) \succ^{\pi}_{rpo} x)}_{\pi^{\mathsf{s}}_1 \wedge \pi^{\mathsf{s}}_{list}} \Big) \wedge \\
&\quad \Big(\pi^{\mathsf{minus}}_2 \to \underbrace{\tau(\mathsf{s}(x) \succ^{\pi}_{rpo} y)}_{false} \Big) \wedge \\
&\quad \Big(\pi^{\mathsf{minus}}_{list} \to \Big(\pi^{\mathsf{s}}_{list} \wedge \Big(\tau(\mathsf{s} >_{\Sigma} \mathsf{minus}) \vee (\tau(\mathsf{s} \approx_{\Sigma} \mathsf{minus}) \wedge \tau(\langle x \rangle \succ^{\mathsf{s},\mathsf{minus},\pi}_{rpo} \langle x,y \rangle)) \Big) \Big) \Big) \\
&= \pi^{\mathsf{s}}_1 \wedge \pi^{\mathsf{s}}_{list} \wedge \neg\pi^{\mathsf{minus}}_{list} \wedge \neg\pi^{\mathsf{minus}}_{2} \vee \\
&\quad \Big(\pi^{\mathsf{minus}}_1 \to \pi^{\mathsf{s}}_1 \wedge \pi^{\mathsf{s}}_{list} \Big) \wedge \\
&\quad \neg\pi^{\mathsf{minus}}_2 \wedge \\
&\quad \Big(\pi^{\mathsf{minus}}_{list} \to \Big(\pi^{\mathsf{s}}_{list} \wedge \Big(\tau(\mathsf{s} >_{\Sigma} \mathsf{minus}) \vee (\tau(\mathsf{s} \approx_{\Sigma} \mathsf{minus}) \wedge \tau(\langle x \rangle \succ^{\mathsf{s},\mathsf{minus},\pi}_{rpo} \langle x,y \rangle)) \Big) \Big) \Big)
\end{aligned}
$$

Note that

$$
\begin{aligned}
\tau(\langle x \rangle \succ^{\mathsf{s},\mathsf{minus},\pi}_{rpo} \langle x,y \rangle) = \quad & m_{\mathsf{s}} \wedge \quad m_{\mathsf{minus}} \wedge \tau(\langle x \rangle \succ^{\mathsf{s},\mathsf{minus},\pi}_{mul} \langle x,y \rangle) \\
\vee \quad & \neg m_{\mathsf{s}} \wedge \neg m_{\mathsf{minus}} \wedge \tau(\langle x \rangle \succ^{\mathsf{s},\mathsf{minus},\pi}_{lex} \langle x,y \rangle)
\end{aligned}
$$

Both $\tau(\langle x \rangle \succ^{\mathsf{s},\mathsf{minus},\pi}_{mul} \langle x,y \rangle)$ *and* $\tau(\langle x \rangle \succ^{\mathsf{s},\mathsf{minus},\pi}_{lex} \langle x,y \rangle)$ *imply* $\pi^{\mathsf{s}}_1 \wedge \neg\pi^{\mathsf{minus}}_1 \wedge \neg\pi^{\mathsf{minus}}_2$*. So to summarize,* $\mathsf{s}(x) \succ^{\pi}_{rpo} \mathsf{minus}(x,y)$ *holds iff*

- s *is not filtered and* minus *is collapsed to its first argument; or*
- s *and* minus *are not collapsed,* s *is greater than* minus *in the precedence, the second argument of* minus *is filtered away, and whenever* minus *keeps its first argument then* s *keeps its first argument, too; or*
- s *and* minus *are not collapsed,* s *is equal to* minus *in the precedence and both have either multiset or lexicographic status,* s *keeps its first argument while* minus *filters away both arguments.*

Although the integration of an argument filter π enlarges the formula resulting from the encoding, one still obtains a cubic bound on its size.

Theorem 18 *Let* K *be the size of the dependency pair constraint (2). Then the size of the formula (2″) is in* $\mathcal{O}(K^3)$*.*

Proof To prove this theorem we just refer to the proof of Theorem 10. The reason is that for all Formulas (4′)–(19′), their size has the same asymptotical bound as their counterparts (4)–(19). One only has to care for the $c = |\Sigma|$ new formulas $\tau(\pi^f)$ in Equation (28). Similar to $\tau(\mu^f)$ and $\tau(\mu^{f,\pi})$, the size of these formulas is within $\mathcal{O}(a^3)$ where a is the maximal arity of symbols in Σ, i.e., $a \leqslant K$. □

4.4 Summary

For a pair of terms s and t, we defined the encoding $\tau(s \succ^{\pi}_{rpo} t)$. This is a formula which is satisfiable iff there exists an RPO and an argument filter π such that $s \succ^{\pi}_{rpo} t$. Now we can solve dependency pair constraints of the form (2), where we use reduction pairs based on RPO and argument filters.

$$\boxed{\bigwedge_{\ell \to r \in \mathcal{P} \cup \mathcal{R}} \ell \succsim^{\pi}_{rpo} r \quad \wedge \quad \bigvee_{\ell \to r \in \mathcal{P}} \ell \succ^{\pi}_{rpo} r}$$

So here all rules $\mathcal{R}$ and dependency pairs $\mathcal{P}$ must be weakly oriented and those DPs that are also strictly decreasing are removed. In this way, one can for example prove termination of the TRS in Example 11, cf. Example 15. Here, the resulting Constraint (2) is transformed into a propositional formula whose satisfiability is easily shown by existing SAT solvers.

5 Usable Rules and their SAT Encodings

An important improvement of the reduction pair processor of Theorem 13 is to weaken the condition that *all* rules of $\mathcal{R}$ have to be weakly decreasing. Instead, it is sufficient to require that only the so-called *usable rules* from $\mathcal{R}$ are weakly decreasing, as specified in the usable rule constraint (3). In Section 5.1 we recapitulate the concept of usable rules. Afterwards, Section 5.2 extends our encoding from Section 4.3 to capture this improvement. Again we prove that the size of our encoding is still cubic. We summarize the results of Section 5 in Section 5.3.

5.1 Usable Rules

As already outlined in the introduction (Section 1), the main problem is that the set of usable rules depends on the argument filter used. An argument filter which deletes many arguments has the advantage that it results in only few usable rules (i.e., only few rules of $\mathcal{R}$ have to be weakly decreasing). But on the other hand, such a filter has the disadvantage that satisfying the resulting constraints may be very hard (or even impossible) since those arguments that were crucial for the termination behavior may have been filtered away. Similarly, an argument filter that deletes only few arguments has the advantage that the full information for term comparisons is still available. But at the same time, it results in more usable rules that have to be oriented. Hence, the SAT encoding has to take all these aspects into account, i.e., it has to search simultaneously for an RPO and an argument filter that satisfy the resulting inequalities (where the set of these inequalities again depends on the argument filter used).

Given a DP problem $(\mathcal{P}, \mathcal{R})$, the rules of $\mathcal{R}$ are used to evaluate the arguments between two function calls. Thus, they are only applied to evaluate (instances of) subterms occurring on right-hand sides of dependency pairs from $\mathcal{P}$.

Example 19 Consider again the TRS $\mathcal{R}$ and its dependency pairs from Examples 11 and 12. The rules from $\mathcal{R}$ are applied to evaluate the subterm $\mathsf{minus}(\dots)$ *in an instance of the right-hand side of the dependency pair*

$$\mathsf{DIV}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{DIV}(\mathsf{minus}(x, y), \mathsf{s}(y)). \tag{23}$$

The only defined symbol occurring in the right-hand side of any dependency pair of the example is minus*. Hence, only the* minus*-rules are* usable*. In other words, the two* div*-rules of Example 11 are not usable and thus, they do not need to be weakly decreasing.*

One can further restrict the set of usable rules in the presence of argument filters. Consider for example an argument filter π which ignores the first argument of DIV *by setting $\pi(\mathsf{DIV}) = [2]$. Now the* minus *symbol is filtered away and no longer "occurs" in the right-hand side of the dependency pair (it only occurs in positions which are filtered away). Consequently, then also the* minus*-rules would become non-usable since an evaluation with these rules is only possible in subterms which are ignored by the reduction pair $(\succsim^{\pi}, \succ^{\pi})$.*

Definition 20 recapitulates the definition of usable rules w.r.t. an argument filter from [24], where we choose a formulation that eases the encoding into propositional logic later on. Let $(\mathcal{P}, \mathcal{R})$ be a DP problem and let π be an argument filter. The definition of usable rules is inductive and is determined not only by $\mathcal{P}$ and $\mathcal{R}$, but also by the argument filter π. For each dependency pair $s \to t \in \mathcal{P}$, all those rules of $\mathcal{R}$ are "usable" which are relevant for evaluating the term t unless the rules are only applied in subterms that are filtered away by π. This is captured by Item (c) of the definition below while the notion of usable rules relevant to a term t is captured by Items (a) and (b). Now consider a rule $\ell \to r$ which has already been determined to be "usable". Since $\ell \to r$ is usable, its right-hand side may need to be evaluated. Hence also the rules from $\mathcal{R}$ that are relevant to the term r should be considered "usable". This inductive part of the definition is captured by Item (d).

In the definition, $Rls_{\mathcal{R}}(f)$ denotes those rules of $\mathcal{R}$ which define a function symbol f, i.e., which are rooted by f:

$$Rls_{\mathcal{R}}(f) = \{\ell \to r \in \mathcal{R} \mid root(\ell) = f\}$$

Definition 20 (Usable Rules modulo π) Let $\mathcal{R}$ be a TRS and π be an argument filter. The *usable rules* $\mathcal{U}(t, \mathcal{R}, \pi)$ *of a term* t are defined as

(a) $\mathcal{U}(x, \mathcal{R}, \pi) = \varnothing$ for a variable x
(b) $\mathcal{U}(f(t_1, \ldots, t_n), \mathcal{R}, \pi) = Rls_{\mathcal{R}}(f) \cup \bigcup_{i \in \pi(f)} \mathcal{U}(t_i, \mathcal{R}, \pi)$

The *usable rules of a DP problem* $(\mathcal{P}, \mathcal{R})$ *modulo* π are the least set $\mathcal{U}(\mathcal{P}, \mathcal{R}, \pi)$ such that

(c) $\mathcal{U}(\mathcal{P}, \mathcal{R}, \pi) \supseteq \bigcup_{s \to t \in \mathcal{P}} \mathcal{U}(t, \mathcal{R}, \pi)$ and
(d) $\mathcal{U}(\mathcal{P}, \mathcal{R}, \pi) \supseteq \bigcup_{\ell \to r \in \mathcal{U}(\mathcal{P}, \mathcal{R}, \pi)} \mathcal{U}(r, \mathcal{R}, \pi)$

The following example illustrates the definition.

Example 21 Consider the following alternative TRS $\mathcal{R}$ for division and its dependency pairs.

$$
\begin{array}{rrcl}
\mathcal{R}\colon & \mathsf{minus}(x,\mathsf{0}) & \to & x \\
& \mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{minus}(x,y) \\
& \mathsf{not}(\mathsf{true}) & \to & \mathsf{false} \\
& \mathsf{ge}(x,\mathsf{0}) & \to & \mathsf{true} \\
& \mathsf{ge}(\mathsf{0},\mathsf{s}(y)) & \to & \mathsf{not}(\mathsf{true}) \\
& \mathsf{ge}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{ge}(x,y) \\
& \mathsf{div}(x,y) & \to & \mathsf{if}(\mathsf{ge}(x,y),x,y) \\
& \mathsf{if}(\mathsf{true},\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{div}(\mathsf{minus}(x,y),\mathsf{s}(y))) \\
& \mathsf{if}(\mathsf{false},x,\mathsf{s}(y)) & \to & \mathsf{0}
\end{array}
\qquad
\begin{array}{rrcl}
DP(\mathcal{R})\colon & \mathsf{MINUS}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{MINUS}(x,y) \\
& \mathsf{GE}(\mathsf{0},\mathsf{s}(y)) & \to & \mathsf{NOT}(\mathsf{true}) \\
& \mathsf{GE}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{GE}(x,y) \\
& \mathsf{DIV}(x,y) & \to & \mathsf{IF}(\mathsf{ge}(x,y),x,y) \\
& \mathsf{DIV}(x,y) & \to & \mathsf{GE}(x,y) \\
& \mathsf{IF}(\mathsf{true},\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{DIV}(\mathsf{minus}(x,y),\mathsf{s}(y)) \\
& \mathsf{IF}(\mathsf{true},\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{MINUS}(x,y)
\end{array}
$$

If we use an argument filter which keeps every argument, then the usable rules for the DP problem $(DP(\mathcal{R}),\mathcal{R})$ are the minus- *and* ge*-rules, since* minus *and* ge *occur in the right-hand side of a dependency pair, and the* not*-rules, since* not *occurs in the right-hand side of a (usable)* ge*-rule. But if one chooses an argument filter with $\pi(\mathsf{IF}) = [2]$, then the* ge- *and* not*-rules are no longer usable since* ge *does not occur in the right-hand side of the filtered dependency pair $\mathsf{DIV}(x,y) \to \mathsf{IF}(x)$.*

The following theorem from [24] states that one can refine the reduction pair processor of Theorem 13 by considering usable rules modulo π. As shown in [24,27], such a refinement is only possible for so-called $\mathcal{C}_\varepsilon$*-compatible* reduction pairs $(\succsim,\succ)$ where $\mathsf{c}(x,y) \succsim x$ and $\mathsf{c}(x,y) \succsim y$ hold for some fresh function symbol c. But this is not a restriction in our context, since reduction pairs like $(\succsim^\pi_{rpo},\succ^\pi_{rpo})$ are always $\mathcal{C}_\varepsilon$-compatible by taking $\pi(\mathsf{c}) = [1,2]$.

Theorem 22 (Reduction Pair Processor modulo π) *Let $(\succsim^\pi,\succ^\pi)$ be a $\mathcal{C}_\varepsilon$-compatible reduction pair such that $\mathcal{P} \cup \mathcal{U}(\mathcal{P},\mathcal{R},\pi) \subseteq \succsim^\pi$. If there is no infinite $(\mathcal{P} \setminus \succ^\pi,\mathcal{R})$-chain, then there is no infinite $(\mathcal{P},\mathcal{R})$-chain.*

So a proof of termination based on the reduction pair processor modulo π is obtained by solving a usable rule constraint of the form (3) from the introduction.

$$\boxed{\bigwedge_{\ell\to r \in \mathcal{P}\cup\mathcal{U}(\mathcal{P},\mathcal{R},\pi)} \ell \succsim^\pi r \quad\wedge\quad \bigvee_{\ell\to r \in \mathcal{P}} \ell \succ^\pi r}$$

As demonstrated in [24], the restriction to the usable rules modulo argument filters results in a significant gain of termination proving power. However, identifying the usable rules increases the size of the search space considerably and it is not straightforward to automate using SAT solvers.

5.2 Encoding RPO modulo Argument Filter with Usable Rules

There are two challenges when encoding the usable rule constraint (3) to an equivalent SAT formula. The first challenge is the mutual dependency between

(i) the fact that the argument filter should be chosen in such a way that the usable rules and the dependency pairs can be oriented, and
(ii) the fact that the definition of which rules are "usable" depends also on the argument filter.

As discussed before, an enumeration of all argument filters is hopelessly inefficient. Thus, the idea is to encode the usable rule constraint (3) in terms of a *generic* argument filter. In this way the SAT solver takes care of the facts (i) and (ii) when searching for argument filters. The second challenge is the fact that Definition 20 is inductive and defines the set of usable rules as the *minimal* set satisfying Items (c) and (d) in Definition 20.

Recall Formula ($2''$) from Section 4.3 which encodes the dependency pair constraint (2):

$$\boxed{\bigwedge_{f\in\Sigma}\Big(\tau(\mu^{f,\pi})\wedge\tau(\pi^f)\Big)\wedge\bigwedge_{\ell\to r\,\in\,\mathcal{P}\cup\mathcal{R}}\tau(\ell\succsim^\pi_{rpo} r)\wedge\bigvee_{\ell\to r\,\in\,\mathcal{P}}\tau(\ell\succ^\pi_{rpo} r)}$$

At the end of this section we will present an encoding for the usable rule constraint (3) which has the following (similar) form, where $\tau(\mathcal{U}^\pi)$ encodes (an overapproximation of) the usable rules. Here, $\mathcal{U}^\pi$ is a set of Boolean flags of the form u^π_ρ. For each rule $\rho = \ell\to r\in\mathcal{R}$, we use u^π_ρ to indicate whether the rule ρ needs to be oriented. The propositional formula $\tau(\mathcal{U}^\pi)$ is added to the encoding and ensures that u^π_ρ is *true* whenever the corresponding rule ρ is usable, i.e., whenever $\rho\in\mathcal{U}(\mathcal{P},\mathcal{R},\pi)$.

$$\boxed{\begin{array}{l}\tau(\mathcal{U}^\pi)\quad\wedge\quad\bigwedge_{f\in\Sigma}\Big(\tau(\mu^{f,\pi})\quad\wedge\quad\tau(\pi^f)\Big)\quad\wedge\\ \bigwedge_{\ell\to r\,\in\,\mathcal{R}}(u^\pi_{\ell\to r}\to\tau(\ell\succsim^\pi_{rpo} r))\wedge\bigwedge_{\ell\to r\,\in\,\mathcal{P}}\tau(\ell\succsim^\pi_{rpo} r)\wedge\bigvee_{\ell\to r\,\in\,\mathcal{P}}\tau(\ell\succ^\pi_{rpo} r)\end{array}}\qquad(3')$$

The following definition introduces the SAT encoding of usable rules and is directly derived from Definition 20. We use the propositional variables π^f_i from Section 4.3 which encode "$i\in\pi(f)$", i.e., $\pi^f_i = \mathit{true}$ indicates that the i-th argument of f is not filtered away by π. The first part of the definition introduces a formula $\omega^\pi(t)$ which requires that u^π_ρ is *true* for all those rules ρ that are relevant to a term t. This corresponds directly to Items (a) and (b) of Definition 20. The second part of the definition corresponds to Items (c) and (d) in Definition 20. It states first that for each dependency pair $s\to t\in\mathcal{P}$, all rules relevant to the term t are "usable". Moreover, if the rule $\ell\to r$ is usable then so are all rules that are relevant to the term r.

Definition 23 (Encoding Usable Rules modulo π) Let $\mathcal{R}$ be a TRS and π be an argument filter. For any term t we introduce the formula $\omega^\pi(t)$:

(a) $\omega^\pi(x) = \mathit{true}$ for a variable x

(b) $\omega^\pi(f(t_1,\ldots,t_n)) = \bigwedge_{\rho\,\in\,Rls_\mathcal{R}(f)} u^\pi_\rho\quad\wedge\bigwedge_{1\leqslant i\leqslant n}\Big(\pi^f_i\to\omega^\pi(t_i)\Big)$

Let $(\mathcal{P},\mathcal{R})$ be a DP problem. We introduce the formula $\tau(\mathcal{U}^\pi)$.

$$\tau(\mathcal{U}^\pi) = \underbrace{\bigwedge_{s\to t\,\in\,\mathcal{P}}\omega^\pi(t)}_{\text{(c)}}\quad\wedge\quad\underbrace{\bigwedge_{\ell\to r\,\in\,\mathcal{R}}(u^\pi_{\ell\to r}\to\omega^\pi(r))}_{\text{(d)}}\qquad(29)$$

There is an important difference between Definition 20 which introduces usable rules and Definition 23 which presents their encoding. Part (d) of the encoding states that if $\ell\to r$ is usable then so is every rule relevant to the right-hand side r. This is

equivalent to the statement in Part (d) of Definition 23. However, the encoding does not capture the minimality of the set of usable rules as expressed in Definition 20. Any set of rules satisfying the conditions constitutes a model of the encoding. We will come back to this point after the following example.

Example 24 Consider again the DP problem $(DP(\mathcal{R}), \mathcal{R})$ from Example 21. Using the encoding of Definition 23, for $\mathcal{P} = DP(\mathcal{R})$ we obtain the following conjunction $\tau(\mathcal{U}^\pi)$ after propositional simplification:

$$
\begin{array}{rr}
 & (\pi_1^{\mathsf{IF}} \to u^\pi_{\mathsf{ge}(x,0)\to\mathsf{true}} \wedge u^\pi_{\mathsf{ge}(0,\mathsf{s}(y))\to\mathsf{not}(\mathsf{true})} \wedge u^\pi_{\mathsf{ge}(\mathsf{s}(x),\mathsf{s}(y))\to\mathsf{ge}(x,y)}) \\
\wedge & (\pi_1^{\mathsf{DIV}} \to u^\pi_{\mathsf{minus}(x,0)\to x} \wedge u^\pi_{\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y))\to\mathsf{minus}(x,y)}) \\
\wedge & (u^\pi_{\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y))\to\mathsf{minus}(x,y)} \to u^\pi_{\mathsf{minus}(x,0)\to x} \wedge u^\pi_{\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y))\to\mathsf{minus}(x,y)}) \\
\wedge & (u^\pi_{\mathsf{ge}(0,\mathsf{s}(y))\to\mathsf{not}(\mathsf{true})} \to u^\pi_{\mathsf{not}(\mathsf{true})\to\mathsf{false}}) \\
\wedge & (u^\pi_{\mathsf{ge}(\mathsf{s}(x),\mathsf{s}(y))\to\mathsf{ge}(x,y)} \to u^\pi_{\mathsf{ge}(x,0)\to\mathsf{true}} \wedge u^\pi_{\mathsf{ge}(0,\mathsf{s}(y))\to\mathsf{not}(\mathsf{true})}) \\
\wedge & (u^\pi_{\mathsf{div}(x,y)\to\mathsf{if}(\mathsf{ge}(x,y),x,y)} \to \ldots) \\
\wedge & (u^\pi_{\mathsf{if}(\mathsf{true},\ldots)\to\mathsf{s}(\mathsf{div}(\ldots)} \to \ldots)
\end{array}
$$

Note that we removed conjuncts like $\pi_2^{\mathsf{DIV}} \to \omega^\pi(\mathsf{s}(y))$, *because* $\omega^\pi(\mathsf{s}(y))$ *can be simplified to true. Note also that the variables* $u^\pi_{\mathsf{div}(\ldots)\to\ldots}$, $u^\pi_{\mathsf{if}(\mathsf{true},\ldots)\to\ldots}$, *and* $u^\pi_{\mathsf{if}(\mathsf{false},\ldots)\to\ldots}$ *can all be set to false, because* div *and* if *do not occur on any right-hand side of a dependency pair. Hence, the corresponding propositional variables* $u^\pi_{\mathsf{div}(\ldots)\to\ldots}$, $u^\pi_{\mathsf{if}(\mathsf{true},\ldots)\to\ldots}$, *and* $u^\pi_{\mathsf{if}(\mathsf{false},\ldots)\to\ldots}$ *do not occur on right-hand sides of implications in* $\tau(\mathcal{U}^\pi)$ *except those implications where the left-hand side is also* $u^\pi_{\mathsf{div}(\ldots)\to\ldots}$ *or* $u^\pi_{\mathsf{if}(\mathsf{true},\ldots)\to\ldots}$.

As already mentioned, the issue of minimality of the set of usable rules is not captured by the encoding of Definition 23. In particular, assigning the value *true* to all variables u^π_ρ will always satisfy $\tau(\mathcal{U}^\pi)$ (for any argument filter). There are many techniques to obtain a minimal model for an inductive definition such as the one in Equation (29). In fact there is a whole branch of SAT solvers for inductive definitions, for example SAT(ID) [39]. However, in our application we do not have to enforce minimality, which simplifies the encoding considerably. In the usable rule constraint (3), it is sufficient to make just the usable rules ρ weakly decreasing, but it is not a problem if additional rules ρ are weakly decreasing as well. Our encoding leaves it up to the SAT solver to decide which variables u^π_ρ are set to *true*. The only condition that has to be ensured (by the formula $\tau(\mathcal{U}^\pi)$) is that u^π_ρ is at least *true* for every usable rule ρ.

Hence, the encoding for argument filters and usable rules can now be done by the formula we already presented at the beginning of this section.

$$
\boxed{\begin{array}{l}
\tau(\mathcal{U}^\pi) \quad \wedge \quad \bigwedge_{f\in\Sigma} \Big(\tau(\mu^{f,\pi}) \quad \wedge \quad \tau(\pi^f)\Big) \quad \wedge \\
\bigwedge_{\ell\to r\in\mathcal{R}} (u^\pi_{\ell\to r} \to \tau(\ell \succsim^\pi_{rpo} r)) \wedge \bigwedge_{\ell\to r\in\mathcal{P}} \tau(\ell \succsim^\pi_{rpo} r) \wedge \bigvee_{\ell\to r\in\mathcal{P}} \tau(\ell \succ^\pi_{rpo} r)
\end{array}} \qquad (3')
$$

We observe that the size of the overall formula is the same as for Formula $(2'')$ with the addition of $\tau(\mathcal{U}^\pi)$ and the additional premises for the inequalities $\ell \succsim^\pi_{rpo} r$.

Theorem 25 *Let K be the size of the usable rule constraint (3). Then the size of Formula $(3')$ is in $\mathcal{O}(K^3)$.*

Proof Theorem 18 shows that the size of Formula ($2''$) is cubic in K. Hence, it remains to prove that the size of $\tau(\mathcal{U}^\pi)$ is at most cubic in K. In fact, for $\tau(\mathcal{U}^\pi)$ one can even derive a smaller bound: The size of $\omega^\pi(t)$ is in $\mathcal{O}(|t| \cdot |\mathcal{R}|)$ for every term t. Hence, the size of $\tau(\mathcal{U}^\pi)$ is at most $\sum_{s \to t \in \mathcal{P} \cup \mathcal{R}} |t| \cdot |\mathcal{R}|$, which is in $\mathcal{O}(K^2)$ since both $\sum_{s \to t \in \mathcal{P} \cup \mathcal{R}} |t| \leqslant K$ and $|\mathcal{R}| \leqslant K$. □

5.3 Summary

For any TRS $\mathcal{R}$ and any set of DPs $\mathcal{P}$, we have shown how to encode the usable rule constraint (3):

$$\bigwedge_{\ell \to r \in \mathcal{P} \cup \mathcal{U}(\mathcal{P},\mathcal{R},\pi)} \ell \succsim^\pi_{rpo} r \quad \wedge \quad \bigvee_{\ell \to r \in \mathcal{P}} \ell \succ^\pi_{rpo} r$$

Here, we use reduction pairs based on RPO and argument filters. The formula resulting from our encoding is satisfiable iff there exist an RPO and an argument filter π such that all usable rules and all dependency pairs are weakly decreasing and at least one DP is strictly decreasing. Then all strictly decreasing DPs can be removed. For example, in this way one can easily prove termination of the TRS from Example 21.

6 Implementation and Experiments

In order to assess the impact of our contributions, we performed extensive experiments comparing an implementation based on our results to a dedicated implementation without SAT solving.

To measure the performance of termination tools, since 2004 there is an annual *International Termination Competition*.[3] Here, the tools are tested against each other on a large data base of examples (the so-called *Termination Problem Data Base*, TPDB). Since AProVE was the most powerful tool for proving termination of TRSs in all competitions so far (from 2004–2010), it provides the ideal setting to evaluate the impact of our new SAT encoding.

We tested our implementation on all 1391 TRSs from the TPDB (version 5.0.2). The experiments were run on a 2.67 GHz Intel Core i7 and as in the *International Termination Competition*, we used a time-out of 60 seconds for each example.

The results of our experiments are summarized in Table 1. The columns of the table describe our experiments to measure the effect of SAT solving for three configurations of AProVE corresponding to Constraints (1)–(3) separately. In this way, the contributions of Sections 3–5 are evaluated consecutively.

In the first configuration, for Constraint (1), we prove termination by orienting rules repeatedly with a path order and by removing those rules that are strictly decreasing. This configuration is used to evaluate the encoding of RPO from Section 3. In the second configuration, for Constraint (2), we first build the initial DP problem and then apply the reduction pair processor (Theorem 13) repeatedly. Here, we we also apply the *dependency graph processor* [1, 22, 26] (using the approximation of [21]) to treat sets of "mutually recursive dependency pairs" separately. With this configuration we evaluate

[3] `http://www.termination-portal.org/wiki/Termination_Competition`

order		Constraint (1) dedicated	Constraint (1) SAT	Constraint (2) dedicated	Constraint (2) SAT	Constraint (3) dedicated	Constraint (3) SAT
LPO	proved	**147**	**147**	264	**304**	421	**435**
	time-out	51	**0**	464	**9**	187	**0**
	runtime	4158.3	**26.7**	29508.0	**1269.1**	12653.2	**202.1**
QLPO	proved	**169**	**169**	284	**362**	453	**489**
	time-out	141	**0**	502	**24**	241	**9**
	runtime	9597.6	**36.8**	31885.8	**2811.2**	15898.7	**1503.6**
LPOS	proved	**167**	**167**	275	**316**	427	**440**
	time-out	62	**0**	481	**7**	192	**0**
	runtime	5148.9	**26.9**	30751.8	**1194.2**	13354.0	**196.3**
QLPOS	proved	193	**194**	298	**412**	467	**530**
	time-out	212	**0**	524	**22**	275	**13**
	runtime	14528.6	**62.3**	33717.4	**2719.9**	18307.1	**1554.5**
MPO	proved	**107**	**107**	231	**261**	425	**433**
	time-out	77	**0**	491	**19**	217	**0**
	runtime	6048.9	**33.9**	31501.0	**1876.6**	14643.8	**232.5**
QMPO	proved	**120**	**120**	251	**335**	467	**502**
	time-out	176	**0**	521	**36**	259	**15**
	runtime	12308.2	**78.9**	33724.4	**3768.8**	17186.4	**1840.4**
RPO	proved	**172**	**172**	280	**324**	441	**454**
	time-out	82	**0**	503	**17**	219	**0**
	runtime	7160.6	**38.3**	32714.6	**1831.4**	14972.6	**216.8**
QRPO	proved	200	**202**	297	**429**	485	**552**
	time-out	234	**1**	562	**36**	292	**17**
	runtime	16632.4	**157.5**	36418.0	**4014.6**	19489.4	**1999.1**

Table 1 Experimental Results.

our encoding of RPO with argument filters from Section 4. The third configuration, for Constraint (3), is similar to the second one, but applies the reduction pair processor of Theorem 22. In this way we evaluate our encoding of RPO with argument filters and usable rules from Section 5.

For each of these configurations we compare an implementation based on encoding to SAT with a dedicated implementation without SAT solving. For the dedicated implementation, we use the solver of AProVE where the search for RPO and argument filters is performed by a vastly improved version of the algorithm described in [24].

For the SAT-based implementation, we extended AProVE by the encodings from this paper. We present results using the SAT4J [35] SAT solver and its implementation of Tseitin's algorithm [42] for transformation into CNF. The implementation uses sharing of subformulas to keep the size of the resulting propositional formulas polynomial, as discussed in Section 3.10.

Each of the rows in Table 1 summarizes the experiments for one of the path orders, where we consider that path order for strict and non-strict (or "quasi-") precedences on symbols. These include lexicographic path orders (LPO/QLPO), lexicographic path orders with status (LPOS/QLPOS), multiset path orders (MPO/QMPO), and recursive path orders (RPO/QRPO). For each experiment, we present the number of TRSs proved terminating ("proved"), the number of time-outs given a limit of 60 seconds ("time-out"), and the analysis time in seconds for running AProVE on all 1391 TRSs ("runtime"). The "best" numbers are always printed in **bold**.

SAT vs. Dedicated: The table indicates that the SAT-based implementation improves performance by orders of magnitude over the dedicated solver for all configurations. Precision is also improved due to the significant decrease in the number of time-outs. The largest decrease is a reduction from 562 time-outs for QRPO using Constraint (2) compared to 36 for the SAT-based implementation.

Strict vs. Quasi-Precedence: For all configurations and path orders, considering quasi-precedences improves precision. In the past, dedicated solvers often did not use quasi-precedences as the additional cost was prohibitive (observe the increase in the number of time-outs in each transition from strict to quasi-precedences). When using SAT solving, the additional cost is considerably less prohibitive. Consider for example the transition from RPO to QRPO in the configuration for Constraint (2). Using the dedicated solver we obtain only 17 additional proofs of termination but suffer 59 additional time-outs (a total of 562). In contrast, using a SAT solver we obtain 105 additional proofs while only introducing 18 additional time-outs (a total of 36).

Introducing Permutations: The introduction of permutations to lexicographic path orders (resulting in (Q)LPOS) offers a noticeable increase in precision. For all three configurations, this increase comes at essentially no additional cost. Note that while also for dedicated solvers, total runtimes do not exhibit a significant increase in costs, this is primarily due to the high number of time-outs.

Introducing Multisets: The rows for (Q)MPO illustrate that multiset path orders on their own are relatively weak when compared to the other path orders. However, to appreciate the impact of multiset extensions one should compare (Q)RPO to (Q)LPOS where the only difference is the introduction of multiset comparisons. Consider for example the difference between QLPOS and QRPO for the configuration with Constraint (3), where we obtain additional 22 proofs of termination with only 4 additional time-outs.

Introducing Argument Filters: Comparing the columns for Constraint (1) and Constraint (2) illustrates the effect of introducing argument filters. While the number of proofs obtained increases considerably (more than 50% when using SAT solvers) the additional cost for dedicated solvers is prohibitive. Even for the most simple LPO and MPO, we obtain more than 400 additional time-outs. In contrast, when applying a SAT solver, in the most complex QRPO we obtain only 35 additional time-outs while gaining 227 proofs of termination (a total of 429). Note also that for this path order, the dedicated solver only gives 297 proofs of termination.

Introducing Usable Rules: Comparing the columns for Constraint (2) and Constraint (3) illustrates the effect of introducing usable rules. For all path orders and both dedicated and SAT-based solvers, the addition of usable rules significantly improves precision while at the same time runtimes and time-outs are reduced. For example, for QRPO using the SAT-based implementation we obtain 123 additional proofs of termination and decrease runtimes and time-outs by more than 50%.

Size of the Formulas: To give an impression of the size of the resulting propositional formulas in CNF, we measure the average and maximal numbers of clauses, variables, and literals for the most powerful configuration (the configuration with Constraint (3) using QRPO). Note that this is also the configuration resulting in the longest

	no RPO	*dedicated RPO*	*SAT RPO*
proved	977	984	**1008**
disproved	**234**	226	**234**
time-out	175	177	**144**
runtime	14626.4	15981.5	**13410.2**

Table 2 Experiments for Termination Proofs with a Full Version of AProVE.

formulas. Here, the average (maximal) number of clauses in a CNF is 15062 (2015091), the average (maximal) number of different variables in a CNF is 4310 (543690), and the average (maximal) number of occurrences of literals in a CNF is 37507 (5279871).

Full Impact: In order to assess the impact of our contributions, we consider a full version of AProVE as used in the termination competitions which is allowed to use all techniques available in the tool. In Table 2, we present results running three variants on all 1391 examples of the TPDB.

The first column ("*no RPO*") applies the strategy used in the last termination competition, but where all path orders were disabled. The second column ("*dedicated RPO*") applies an identical strategy except that it uses a dedicated non-SAT-based QRPO solver. Finally, the column ("*SAT RPO*") applies the same strategy but uses a SAT-based QRPO solver. All three versions of AProVE can also prove non-termination. The rows in the table give the numbers of termination proofs ("proved"), non-termination proofs ("disproved"), time-outs ("time-out"), and total runtimes in seconds ("runtime").

Prior to the introduction of SAT-based solvers for path orders, AProVE did not apply these orders in competition strategies (as in column "no RPO"). This is reflected in the table where we see that when introducing a dedicated QRPO solver (column "dedicated RPO") we obtain only 7 additional proofs of termination while losing 8 proofs of non-termination and gaining 2 time-outs. In contrast, when introducing a SAT-based QRPO solver (column "SAT RPO"), we gain 31 proofs of termination, lose no proofs of non-termination, and reduce the number of time-outs by 31.

7 Summary

In this paper, we demonstrated the power of propositional encoding and application of SAT solving to termination analysis with RPO and dependency pairs. We started in Section 3 with a SAT encoding for RPO. Section 4 extended the SAT-based approach to the more realistic setting of DP problems with RPO and argument filters. In Section 5, we improved the approach further by also considering usable rules. The main challenge derives from the strong dependencies between the notions of RPO, argument filters, and the set of rules which need to be oriented. The key to a solution was to introduce and encode in SAT all of the constraints originating from these notions into a single search process.

We introduced such an encoding and through implementation and experimentation showed that it yields 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 we refer

to `http://aprove.informatik.rwth-aachen.de/eval/SATDP/`. There, we also provide data on experiments with other (DPLL-based) SAT solvers like glucose [2], PrecoSAT [6], and MiniSAT2 [13]. These experiments indicate that the choice of the SAT solver does not have a significant influence on the overall performance of the termination prover.

Instead of encoding into SAT, one can obviously also use SAT modulo theories (SMT) as an encoding target. This approach has already been investigated for other reduction orders like Knuth-Bendix orders [44] and polynomial orders [8]. However, numbers play a key role in these orders, whereas most parts in the RPO definition are purely propositional. So for Contributions (A)–(D), encoding to SMT instead of SAT would not simplify the encoding substantially. Only the cardinality constraints and the precedence constraints can be expressed more succinctly in SMT than in SAT by using linear integer arithmetic or even integer difference logic. However, the overall encoding would still have the same asymptotic (cubic) bound for both SAT and SMT. As this paper is not about comparing SAT and SMT, we omitted most low level descriptions and presented more abstract formulations instead, which can then be encoded in either SAT or SMT.

Acknowledgements We thank the referees for many helpful suggestions.

References

1. T. Arts and J. Giesl. Termination of Term Rewriting using Dependency Pairs. *Theoretical Computer Science*, 236(1-2):133–178, 2000.
2. G. Audemard and L. Simon. glucose. `http://www.lri.fr/~simon/glucose/`.
3. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, Cambridge, 1998.
4. A. M. Ben-Amram and M. Codish. A SAT-Based Approach to Size Change Termination with Global Ranking Functions. In *TACAS '08*, LNCS 4963, pages 218–232, 2007.
5. A. Ben Cherifa and P. Lescanne. Termination of Rewriting Systems by Polynomial Interpretations and Its Implementation. *Science of Computer Programming*, 9(2):137–159, 1987.
6. A. Biere. PrecoSAT. `http://fmv.jku.at/precosat/`.
7. M. Bofill, D. Busquets, and M. Villaret. A Declarative Approach to Robust Weighted Max-SAT. In *PPDP '10*, pages 67–76. ACM, 2010.
8. C. Borralleras, S. Lucas, R. Navarro-Marset, E. Rodríguez-Carbonell, and A. Rubio. Solving Non-linear Polynomial Arithmetic via SAT Modulo Linear Arithmetic. In *CADE '09*, LNAI 5663, pages 294–305, 2009.
9. M. Codish, P. Schneider-Kamp, V. Lagoon, R. Thiemann, and J. Giesl. SAT Solving for Argument Filterings. In *LPAR '06*, LNAI 4246, pages 30–44, 2006.
10. M. Codish, V. Lagoon, and P. J. Stuckey. Solving Partial Order Constraints for LPO Termination. *Journal on Satisfiability, Boolean Modeling and Computation*, 5:193–215, 2008.
11. M. Codish, S. Genaim, and P. J. Stuckey. A Declarative Encoding of Telecommunications Feature Subscription in SAT. In *PPDP '09*, pages 255–266. ACM, 2009.
12. N. Dershowitz. Orderings for Term-Rewriting Systems. *Theoretical Computer Science*, 17:279–301, 1982.
13. N. Eén and N. Sörensson. MiniSAT. `http://minisat.se/`.
14. N. Eén and N. Sörensson. Translating Pseudo-Boolean Constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26, 2006.
15. J. Endrullis, J. Waldmann, and H. Zantema. Matrix Interpretations for Proving Termination of Term Rewriting. *Journal of Automated Reasoning*, 40(2-3):195–220, 2008.
16. T. Feydy, A. Schutt, and P. J. Stuckey. Global Difference Constraint Propagation for Finite Domain Solvers. In *PPDP '08*, pages 226–235. ACM, 2008.

17. C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. SAT Solving for Termination Analysis with Polynomial Interpretations. In *SAT '07*, LNCS 4501, pages 340–354, 2007.
18. C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. Maximal Termination. In *RTA '08*, LNCS 5117, pages 110–125, 2008.
19. C. Fuhs, R. Navarro-Marset, C. Otto, J. Giesl, S. Lucas, and P. Schneider-Kamp. Search Techniques for Rational Polynomial Orders. In *AISC '08*, LNAI 5144, pages 109–124, 2008.
20. A. Geser. *Relative Termination*. PhD thesis, University of Passau, Germany, 1990.
21. J. Giesl, R. Thiemann, and P. Schneider-Kamp. Proving and Disproving Termination of Higher-Order Functions. In *FroCoS '05*, LNAI 3717, pages 216–231, 2005.
22. J. Giesl, R. Thiemann, and P. Schneider-Kamp. The Dependency Pair Framework: Combining Techniques for Automated Termination Proofs. In *LPAR '04*, LNAI 3452, pages 301–331, 2005.
23. J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic Termination Proofs in the Dependency Pair Framework. In *IJCAR '06*, LNAI 4130, pages 281–286, 2006.
24. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and Improving Dependency Pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.
25. A. Gotlieb. TCAS Software Verification using Constraint Programming. *The Knowledge Engineering Review*, to appear, 2010.
26. N. Hirokawa and A. Middeldorp. Automating the Dependency Pair Method. *Information and Computation*, 199(1,2):172–199, 2005.
27. N. Hirokawa and A. Middeldorp. Tyrolean Termination Tool: Techniques and Features. *Information and Computation*, 205(4):474–511, 2007.
28. C. Jefferson, N. C. A. Moore, P. Nightingale, and K. E. Petrie. Implementing Logical Connectives in Constraint Programming. *Artificial Intelligence*, 174:1407–1429, 2010.
29. S. Kamin and J. J. Lévy. Two Generalizations of the Recursive Path Ordering. Unpublished Manuscript, University of Illinois, IL, USA, 1980.
30. A. Koprowski and A. Middeldorp. Predictive Labeling with Dependency Pairs using SAT. In *CADE '07*, LNAI 4603, pages 410–425, 2007.
31. M. S. Krishnamoorthy and P. Narendran. On Recursive Path Ordering. *Theoretical Computer Science*, 40:323–328, 1985.
32. M. Kurihara and H. Kondo. Efficient BDD Encodings for Partial Order Constraints with Application to Expert Systems in Software Verification. In *IEA/AIE '04*, LNCS 3029, pages 827–837, 2004.
33. K. Kusakari, M. Nakamura, and Y. Toyama. Argument Filtering Transformation. In *PPDP '99*, LNCS 1702, pages 47–61, 1999.
34. D. Lankford. On Proving Term Rewriting Systems are Noetherian. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA, 1979.
35. D. Le Berre and A. Parrain. SAT4J. `http://www.sat4j.org`.
36. P. Lescanne. Computer Experiments with the REVE Term Rewriting System Generator. In *POPL '83*, pages 99–108. ACM Press, 1983.
37. S. Lescuyer and S. Conchon. Improving Coq Propositional Reasoning Using a Lazy CNF Conversion Scheme. In *FroCoS '09*, LNCS 5749, pages 287–303, 2009.
38. Z. Manna and S. Ness. On the Termination of Markov Algorithms. In *3rd Hawaii International Conference on System Science*, pages 789–792, 1970.
39. M. Mariën, J. Wittocx, M. Denecker, and M. Bruynooghe. SAT(ID): Satisfiability of Propositional Logic Extended with Inductive Definitions. In *SAT '08*, LNCS 4996, pages 211–224, 2008.
40. P. Schneider-Kamp, R. Thiemann, E. Annov, M. Codish, and J. Giesl. Proving Termination using Recursive Path Orders and SAT Solving. In *FroCoS '07*, LNAI 4720, pages 267–282, 2007.
41. N. Tamura, A. Taga, S. Kitagawa, and M. Banbara. Compiling Finite Linear CSP into SAT. *Constraints*, 14(2):254–272, 2009.
42. G. Tseitin. On the Complexity of Derivation in Propositional Calculus. In *Studies in Constructive Mathematics and Mathematical Logic*, pages 115–125. 1968. Reprinted in J. Siekmann and G. Wrightson (editors), *Automation of Reasoning*, 2:466-483, 1983.
43. H. Zankl, N. Hirokawa, and A. Middeldorp. Constraints for Argument Filterings. In *SOFSEM '07*, LNCS 4362, pages 579–590, 2007.

44. H. Zankl, N. Hirokawa, and A. Middeldorp. KBO Orientability. *Journal of Automated Reasoning*, 43(2):173–201, 2009.
45. H. Zankl and A. Middeldorp. Increasing Interpretations. *Annals of Mathematics and Artificial Intelligence*, 56(1):87–108, 2009.